On the Multi-User Security of Short Schnorr Signatures with Preprocessing

Jeremiah Blocki and Seunghoon Lee

Purdue University, West Lafayette, IN, 47906, USA
{jblocki,lee2856}@purdue.edu

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 $p \approx 2^{2k}$ consists of a tuple (s, e), where $e \in \{0,1\}^{2k}$ is a hash output and $s \in \mathbb{Z}_p$ must be computed using the secret key. While the hash output e requires $2k$ bits to encode, Schnorr proposed that it might be possible to truncate the hash value without adversely impacting security.

In this paper, we prove that *short* Schnorr signatures of length $3k$ bits provide k bits of multi-user security in the (Shoup's) generic group model and the programmable random oracle model. We further analyze the multi-user security of key-prefixed short Schnorr signatures against preprocessing attacks, showing that it is possible to obtain secure signatures of length $3k + \log S + \log N$ bits. Here, N denotes the number of users and S denotes the size of the hint generated by our preprocessing attacker, e.g., if $S = 2^{k/2}$, then we would obtain secure $3.75k$-bit signatures for groups of up to $N \leq 2^{k/4}$ users.

Our techniques easily generalize to several other Fiat-Shamir-based signature schemes, allowing us to establish analogous results for Chaum-Pedersen signatures and Katz-Wang signatures. As a building block, we also analyze the 1-out-of-N discrete-log problem in the generic group model, with and without preprocessing.

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 $G = \langle g \rangle$ of prime order p and pick a random secret key $\mathsf{sk} \in \mathbb{Z}_p$. To sign a message m, we pick $r \in \mathbb{Z}_p$ uniformly at random, compute $I = g^r$, $e = \mathsf{H}(I \| m)$, and $s = r + \mathsf{sk} \cdot e \mod p$. Then, the final signature is $\sigma = (s, e)$.

We 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 $\varepsilon_t = t/2^k$ in the (multi-user) signature forgery game, and this should hold for all time bounds $t \leq 2^k$. To achieve k bits of security, we select a hash function H with $2k$-bit outputs, and we select p to be a random $2k$-bit prime so that the length of a signature is $4k$ bits.

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. We refer to this signature scheme as the *short* Schnorr signature scheme. In this paper, we investigate the following questions:

> *Does the short Schnorr signature scheme achieve k bits of (multi-user) security? If so, is the short Schnorr signature scheme also secure against preprocessing attacks?*

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 [BR93]. However, their reduction is not tight, i.e., they show that $\mathsf{Adv}_{\mathsf{sig}} \leq \mathsf{Adv}_{\mathsf{dlog}} \times q_{\mathsf{H}}$ for any attacker making at most q_{H} queries to the random oracle. The loss of the factor q_{H}, 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. 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. 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 the generic group model. While they show that the scheme provides k bits of security, they also consider another version of the generic group model which is different from the definition proposed by Shoup [Sho97]. The reason is that the version they consider is not expressive enough to capture all known attacks, e.g., any attack that requires the ability to hash group elements including preprocessing attacks of Corrigan-Gibbs and Kogan [CK18] cannot be captured in their generic group model. See Appendix B for further discussion if interested.

Galbraith et al. [GMLS02] claimed to have a tight reduction showing that single-user security implies multi-user security of the regular Schnorr signature scheme. However, Bernstein [Ber15] identified an error in the security proof in [GMLS02], proposed a modified "key-prefixed" version of the original Schnorr signature scheme (including the public key as a hash input), and proved that the "key-prefixed" version does provide multi-user security. Derler and Slamanig [DS19] later showed a tight reduction from single-user security to "key-prefixed" multi-user security for a class of key-homomorphic signature schemes including Schnorr signatures. The Internet Engineering Task Force (IETF) adopted the key-prefixed modification of Schnorr signatures to ensure multi-user security [Hao17]. Kiltz et al. [KMP16] later gave a tight security reduction establishing

multi-user security of regular Schnorr signatures in the programmable random oracle model plus (another version of) the generic group model without key-prefixing. Our results imply that key-prefixing is not even necessary to establish tight multi-user security of short Schnorr signatures[1]. On the other hand key-prefixing is both necessary and sufficient to establish multi-user security of (short) Schnorr signatures against preprocessing attackers.

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 (Shoup's) generic group model. We further analyze the *multi-user* security of key-prefixed short Schnorr signature against preprocessing attacks. The preprocessing attacker outputs a hint of size S after making as many as 2^{3k} queries to the random oracle and examining the entire generic group oracle. Later on, the online attacker can use the hint to help win the *multi-user* signature forgery game by forging a signature for *any one* of the N users. By tuning the parameters of the key-prefixed short Schnorr signature scheme appropriately, we can obtain $(3k + \log S + \log N)$-bit signatures with k bits of security against a preprocessing attacker.

Single-User Security of Short Schnorr Signatures. As a warm-up, we first consider the single-user security of the *short* Schnorr Signature scheme without preprocessing, showing that short Schnorr signatures provide k bits of security in the (Shoup's) generic group model and the random oracle model.

Theorem 1 (informal). *Any attacker making at most q queries wins the signature forgery game (chosen message attack) against the short Schnorr signature scheme with probability at most $\mathcal{O}\left(q/2^k\right)$ in the generic group model (of order $p \approx 2^{2k}$) plus programmable random oracle model (See Definition 1 and Theorem 4).*

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 $4k$. 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. 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 is given N public keys $\mathsf{pk}_1, \ldots, \mathsf{pk}_N$, s/he can forge a signature σ which is valid under any one of these public keys (it does not matter which).

[1] The authors of [KMP16] pointed out that their analysis can be adapted to demonstrate multi-user security of short Schnorr signatures (private communication) though the paper itself never discusses short Schnorr signatures. Furthermore, their proof is in a different version of the generic group model which is not suitable for analyzing preprocessing attacks. See discussion in Remark 1.

Theorem 2 (informal). *Let N denote the number of distinct users/public keys. Then any attacker making at most q queries wins the multi-user signature forgery game (chosen message attack) against the short Schnorr signature scheme with probability at most $\mathcal{O}\left((q+N)/2^k\right)$ in the generic group model (of order $p \approx 2^{2k}$) plus programmable random oracle model (See Definition 2 and Theorem 6).*

Theorem 2 guarantees that breaking multi-user security of short Schnorr signatures in the 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 $N \leq q$, 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 $\varepsilon_{\mathsf{MU}}$ can be used to win the single-user forgery game with probability $\varepsilon \geq \varepsilon_{\mathsf{MU}}/N$. For example, suppose that $p \approx 2^{224}$ (i.e., $k = 112$), and there are $N = 2^{32}$ instances of short Schnorr signatures, which is more than the half of the entire world population. In the original single-user security game, an attacker wins with probability at most $\varepsilon \leq \mathcal{O}\left(t/2^k\right)$, so an attacker running in time $t = 2^{80}$ would succeed with probability at most $\varepsilon \approx 2^{-32}$. This only allows us to conclude that an attacker succeeds with probability at most $\varepsilon_{\mathsf{MU}} \leq N\varepsilon \approx 1$ in the multi-user security game! Our security proof implies that the attacker will succeed with probability $\varepsilon_{\mathsf{MU}} \approx \varepsilon \approx 2^{-32}$ in the above example. In particular, we don't lose a factor of N in the security reduction.

Security of Key-Prefixed Short Schnorr Signatures against Preprocessing Attacks. We further show that *key-prefixed short* Schnorr signatures are also secure against preprocessing attacks. Here, we consider a key-prefixed version of Schnorr signatures, because regular Schnorr signatures are trivially vulnerable to preprocessing attacks, e.g., if a preprocessing attacker finds some message m and an integer r such that $e = \mathsf{H}(g^r \| m) = 0$, then $\sigma = (r, 0)$ is *always* a valid signature for *any* public key $\mathsf{pk} = g^{\mathsf{sk}}$ since $g^{r-\mathsf{sk}\cdot 0} = g^r$.

We consider a preprocessing attacker who may query the random oracle at up to 2^{3k} points and may also examine the entire generic group oracles before outputting an S-bit hint for the online attacker. We leave it as an interesting open question whether or not the restriction on the number of random oracle queries is necessary. However, from a practical standpoint, we argue that a preprocessing adversary will never be able to make 2^{2k} queries, e.g., if $k \geq 128$, then 2^{2k} operations is already far too expensive for even a nation-state attacker.

Theorem 3 (informal). *Let N denote the number of distinct users/public keys. Then any preprocessing attacker making at most q_{pre} queries and outputs an S-bit hint during the preprocessing phase and making at most q_{on} queries during the online phase wins the multi-user signature forgery game (chosen message attack) against the short Schnorr signature scheme with probability at most $\widetilde{\mathcal{O}}\left(SN(q_{\mathsf{on}}+N)^2/p + q_{\mathsf{on}}/2^k + Nq_{\mathsf{pre}}q_{\mathsf{on}}/p^2\right)$ in the generic group model of order $p > 2^{2k}$ plus programmable random oracle model (see Theorem 8).*

Theorem 3 tells us that with suitable parameter setting, key-prefixed short Schnorr signatures also achieve k bits of multi-user security even against pre-

processing attacks. In particular, by setting $p \approx 2^{2k}SN$ and maintaining k-bit hash outputs, the short Schnorr signature scheme still maintains k bits of multi-user security against our preprocessing attacker. For example, if $S = 2^{k/2}$ and $N = 2^{k/4}$, then setting $p \approx 2^{2.75k}$ yields signatures of length $k + \log p = 3.75k$. Up to a factor N, the results from Theorem 3 are tight as a preprocessing attacker can succeed with probability at least Sq_{on}^2/p.

Other Fiat-Shamir Signatures. Using similar reductions, we establish similar security bounds for the full-domain hash variant of (key-prefixed) Chaum-Pedersen signatures [CP93] and for Katz-Wang signatures [KW03] with truncated hash outputs. In particular, a preprocessing attacker wins the multi-user signature forgery game with probability at most $\mathcal{O}\left(SNq^2/p + q/2^k\right)$ — see Theorem 10 and Theorem 12 in Section 6. Short Katz-Wang signatures [KW03] have the same length as short Schnorr signatures with equivalent security guarantees, while Chaum-Pedersen signatures are a bit longer.

1.2 Our Techniques

The Multi-User Bridge-Finding Game. We introduce an intermediate problem called the 1-*out-of-N bridge-finding game*. Oversimplifying a bit, in a cyclic group $G = \langle g \rangle$, the attacker is given N inputs $g^{x_1}, \ldots, g^{x_N}$, and the goal of the attacker is to produce a non-trivial linear dependence, i.e., $a_1, \ldots, a_N$ and b such that $a_1x_1 + \ldots + a_Nx_N = b$, and $a_i \neq 0$ for at least one $i \leq N$. We then show that in the generic group model, an attacker making at most q generic group queries can succeed with probability at most $\mathcal{O}\left(q^2/p + qN/p\right)$, where $p \approx 2^{2k}$ is the size of the group.

We also show that a preprocessing attacker wins the 1-out-of-N bridge-finding game with probability at most $\mathcal{O}\left(SNq^2 \log p/p\right)$ when given an arbitrary S-bit hint fixed a priori before $x_1, \ldots, x_N$ are chosen. Our proof adapts a compression argument of [CK18] which was used to analyze the security of the regular discrete logarithm problem. In particular, if the probability that our preprocessing attacker wins is $\omega\left(SNq^2 \log p/p\right)$, then we could derive a contradiction by compressing our random injective map τ, mapping group elements to binary strings.

An interesting corollary of these results is that the 1-out-of-N discrete-log problem is hard even for a preprocessing attacker. Intuitively, if a discrete-log attacker can successfully compute x_i for any $i \leq N$, then s/he can also win the 1-out-of-N bridge-finding game.

Restricted Discrete-Log Oracle. In fact, we consider a stronger attacker $\mathcal{A}$ who may query the usual generic group oracle, and is additionally given access to a restricted discrete-log oracle $\texttt{DLog}$. The oracle $\texttt{DLog}$ will solve the discrete-log problem but only for "fresh" inputs, i.e., given N inputs $g^{x_1}, \ldots, g^{x_N}$, if $h = g^{a_1x_1 + \ldots + a_Nx_N}$ for known values of $a_1, \ldots, a_N$, then this input would not be considered fresh.

We remark that by restricting the discrete-log oracle to fresh queries, we can rule out the trivial attack where the attacker simply queries $\texttt{DLog}(g^{x_i})$ for

any $1 \leq i \leq N$. The restriction also rules out other trivial attacks, where the attacker simply queries $\mathtt{DLog}(g^{x_i+r})$ after computing g^{x_i+r} for some $i \leq N$ and some known value r.

Security Reduction. We then give a reduction showing that any attacker $\mathcal{A}_{\mathsf{sig}}$ that breaks multi-user security of short Schnorr signatures can be used to win the 1-out-of-N bridge-finding game. In the reduction, we interpret the bridge inputs $g^{x_1}, \ldots, g^{x_N}$ as public signing keys, and we simulate the attacker $\mathcal{A}_{\mathsf{sig}}$. The reduction uses the restricted discrete log oracle to ensure that any group element that is submitted as an input to the random oracle has the form $g^{b+a_1x_1+\ldots a_Nx_N}$ for values $a_1, \ldots, a_N$, and b that are known to the bridge attacker. The reduction also makes use of a programmable random oracle to forge signatures whenever $\mathcal{A}_{\mathsf{sig}}$ queries the signing oracle for a particular user $i \leq N$.

One challenge with carrying out this reduction in the preprocessing setting is that we need to ensure that the hint does not allow the attacker to detect when the random oracle has been programmed. We rely on the observation that the reduction programs the random oracle at random inputs which are unknown to the preprocessing a priori.

A similar reduction allows us to establish multi-user security of other Fiat-Shamir-based signatures such as Chaum-Pedersen signatures [CP93] and Katz-Wang signatures [KW03], with and without preprocessing.

1.3 Related Work

Security Proofs in the Generic Group Model. The generic group model goes back to Nechaev [Nec94] and Shoup [Sho97]. 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,WZ11,BL12,GWZ15]. It is well known that in Shoup's generic group model [Sho97], an attacker requires $\Omega(\sqrt{p})$ queries to solve the discrete-log problem in a group of prime order p and the same lower bound holds for other classical problems like Computational Diffie-Hellman (CDH) and Decisional Diffie-Hellman (DDH). This bound is tight as discrete-log algorithms such as the Baby-Step Giant-Step algorithm by Shanks [Sha71], Pollard's Rho and Kangaroo algorithms [Pol78], and the Pohlig-Hellman algorithm [PH06] can all be described generically in Shoup's model. However, there are some exceptions for other elliptic curves and subgroups of $\mathbb{Z}_p^*$, where the best discrete-log algorithms are not generic and are much more efficient than any generic discrete log algorithms, e.g., see [GHS02,MVO91,Sma99].

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 counterexample. Similar to the random oracle model, experience suggests that protocols with security proofs in the generic group model do not 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.

Corrigan-Gibbs and Kogan [CK18] analyzed the security of several key cryptographic problems (e.g., discrete-log, computational/decisional Diffie-Hellman, etc.) against preprocessing attacks in the generic group model. We extend their analysis to analyze the multi-user security of key-prefixed short Schnorr signatures against preprocessing attacks. See Section 5 for the further details.

Schnorr Signatures and Multi-Signatures. Bellare and Dai [BD20] recently showed that the (single-user) security of Schnorr signatures could be based on the Multi-Base Discrete Logarithm problem which in turn is similar in flavor to the One More Discrete Log Problem [BNPS03]. There has also been an active line of work on adapting the Schnorr signature scheme to design compact multi-signature schemes, e.g., see [BN06,BCJ08,DEF+19,MPSW19]. The goal is for multiple parties collaborate to generate a single Schnorr signature which is signed using an aggregate public key which can be (publicly) derived from the individually public keys of each signing party. A very recent line of work has reduced the interaction to generate a Schnorr multi-signature to two-rounds without pairings [NRSW20,NRS20,AB20].

Other Short Signatures. Boneh et al. [BLS04] proposed even shorter signatures called BLS signatures, which is as short as $2k$ bits to yield k bits of security 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].

2 Preliminaries

Let $\mathbb{N}$ be the set of positive integers, and we define $[N] := \{1, \ldots, N\}$ for $N \in \mathbb{N}$. Throughout the paper, we denote the security parameter by k. We say that $\vec{x} = (x_1, \ldots, x_N) \in \mathbb{Z}_p^N$ is an N-dimensional vector over $\mathbb{Z}_p^N$, and for each $i \in [N]$, we define $\hat{u}_i$ to be the i^{th} N-dimensional unit vector, i.e., the i^{th} element of $\hat{u}_i$ is 1, and all other elements are 0 elsewhere. For simplicity, we let $\log(\cdot)$ be a log with base 2, i.e., $\log x := \log_2 x$. The notation $\leftarrow\!\!\$$ denotes a uniformly random sampling, e.g., we say $x \leftarrow\!\!\$ \mathbb{Z}_p$ when x is sampled uniformly at random from $\mathbb{Z}_p$.

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 multiplicative cyclic group of prime order p. In the generic group model, since G is isomorphic to $\mathbb{Z}_p$, we select a random injective map $\tau : \mathbb{Z}_p \to \mathbb{G}$, where $\mathbb{G}$ is the set of bit strings of length ℓ (with $2^\ell \geq p$) and we encode the discrete log of a group element instead of the group element itself.

The key idea in the generic group model is that the map τ does not need to be a group homomorphism, because any adversary against a cryptographic scheme is only available to see a randomly chosen encoding of the discrete log of each group element, not the group element itself. 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 discrete log of the generator g given as $\mathfrak{g} = \tau(1)$. More precisely, for an input $(\mathfrak{a}, \mathfrak{b}) \in \mathbb{G} \times \mathbb{G}$, the oracles $\mathtt{Mult}(\mathfrak{a}, \mathfrak{b})$, and $\mathtt{Inv}(\mathfrak{a})$ act as following:

$$\mathtt{Mult}(\mathfrak{a}, \mathfrak{b}) = \tau(\tau^{-1}(\mathfrak{a}) + \tau^{-1}(\mathfrak{b})), \text{ and}$$
$$\mathtt{Inv}(\mathfrak{a}) = \tau(-\tau^{-1}(\mathfrak{a})),$$

if $\tau^{-1}(\mathfrak{a}), \tau^{-1}(\mathfrak{b}) \in G$. We 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 $\mathtt{Mult}(\mathfrak{a}, \mathfrak{b})$ in $\mathbb{G}$ even if s/he sees the oracle output.

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

2.2 The Schnorr Signature Scheme

The Schnorr signature scheme is a digital signature scheme, which consists of a tuple of probabilistic polynomial-time algorithms $\Pi = (\mathsf{Kg}, \mathsf{Sign}, \mathsf{Vfy})$, where $\mathsf{Kg}(1^k)$ is a key-generation algorithm to generate a secret key $\mathsf{sk} \in \mathbb{Z}_p$ and a public key $\mathsf{pk} = \mathtt{Pow}(\mathfrak{g} = \tau(1), \mathsf{sk}) = \tau(\mathsf{sk})$. The size of the prime number p will be tied to the security parameter k, e.g., $p \approx 2^{2k}$. $\mathsf{Sign}(\mathsf{sk}, m)$ is a signing algorithm which generates a signature σ on a message $m \in \{0,1\}^*$, and $\mathsf{Vfy}(\mathsf{pk}, m, \sigma)$ is a verification algorithm which outputs 1 if the signature is valid, and 0 otherwise.

Throughout the paper, we will consider the notion of the generic group model in the Schnorr signature scheme as described in Figure 1. We remark that verification works for a correct signature $\sigma = (s, e)$, because $R = \mathtt{Mult}(\tau(s), \mathtt{Pow}(\mathtt{Inv}(\mathsf{pk}), e)) = \tau(s - \mathsf{sk} \cdot e) = \tau(r) = I$ if the signature is valid.

Short Schnorr Signatures. Typically, it is assumed that the random oracle $\mathsf{H}(I \| m)$ outputs a uniformly random element $e \in \mathbb{Z}_p$, where p 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 $e \in \mathbb{Z}_{2^k}$ with just k bits. In practice, the shorter random oracle is *easier* to implement, 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 of our random binary string. The result is a signature $\sigma = (s, e)$, which can be encoded in $3k$ bits – $2k$ bits to encode $s \in \mathbb{Z}_p$ plus k bits to encode e. This natural modification is straightforward and

$\mathsf{Kg}(1^k)$:	$\mathsf{Sign}(\mathsf{sk}, m)$:	$\mathsf{Vfy}(\mathsf{pk}, m, \sigma)$:
1 : $\mathsf{sk} \leftarrow\!\!\$\ \mathbb{Z}_p$	1 : $r \leftarrow\!\!\$\ \mathbb{Z}_p$	1 : Parse $\sigma = (s, e)$
2 : $\mathsf{pk} \leftarrow \mathtt{Pow}(\mathfrak{g}, \mathsf{sk})$	2 : $I \leftarrow \mathtt{Pow}(\mathfrak{g}, r)$	2 : $R \leftarrow \mathtt{Mult}(\mathtt{Pow}(\mathfrak{g}, s), \mathtt{Pow}(\mathtt{Inv}(\mathsf{pk}), e))$
3 : **return** $(\mathsf{pk}, \mathsf{sk})$	3 : $e \leftarrow \mathsf{H}(I \| m)$	3 : **if** $\mathsf{H}(R \| m) = e$ **then**
	4 : $s \leftarrow r + \mathsf{sk} \cdot e \mod p$	4 : **return** 1
	5 : **return** $\sigma = (s, e)$	5 : **else return** 0

Figure 1. The Schnorr signature scheme in the generic group model. Note that instead of the direct group operation in G, we use the encoding by the map τ and the generic group oracle queries.

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 $\mathsf{SigForge}^{\tau}_{\mathcal{A},\Pi}(k)$ in the generic group model and the random oracle model. Here, an attacker is given the public key $\mathsf{pk} = \tau(\mathsf{sk})$ along with $\mathfrak{g} = \tau(1)$ (the encoding of the group generator 1 of $\mathbb{Z}_p$). 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. Fixing an encoding map $\tau : \mathbb{Z}_p \to \mathbb{G}$, $\mathfrak{g} = \tau(1)$ and the random oracle H, and an adversary $\mathcal{A}$, consider the following experiment defined for a signature scheme $\Pi = (\mathsf{Kg}, \mathsf{Sign}, \mathsf{Vfy})$:

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

(1) $\mathsf{Kg}(1^k)$ is run to obtain the public and the secret keys $(\mathsf{pk}, \mathsf{sk})$. Here, sk is chosen randomly from the group $\mathbb{Z}_p$ where p is a $2k$-bit prime, and $\mathsf{pk} = \mathtt{Pow}(\mathfrak{g}, \mathsf{sk}) = \tau(\mathsf{sk})$.
(2) Adversary $\mathcal{A}$ is given $(\mathfrak{g} = \tau(1), \mathsf{pk}, p)$ 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) We define $\mathsf{SigForge}^{\tau,\mathsf{H}}_{\mathcal{A},\Pi}(k) = \mathsf{Vfy}(\mathsf{pk}, m, \sigma)$, i.e., the output is 1 when $\mathcal{A}$ succeeds, and 0 otherwise.

Definition 1 formalizes this argument in the sense that an attacker forges a signature if and only if $\mathsf{SigForge}^{\tau,\mathsf{H}}_{\mathcal{A},\Pi}(k) = 1$.

Definition 1. *Consider the generic group model with an encoding map* $\tau : \mathbb{Z}_p \to \mathbb{G}$*. A signature scheme* $\Pi = (\mathsf{Kg}, \mathsf{Sign}, \mathsf{Vfy})$ *is said to be* $(q_{\mathsf{H}}, q_{\mathtt{G}}, q_{\mathtt{S}}, \varepsilon)$-UF-CMA secure (unforgeable against chosen message attack) *if for every adversary* $\mathcal{A}$ *making at most* q_{H} *(resp.* $q_{\mathtt{G}}$*,* $q_{\mathtt{S}}$*) queries to the random oracle (resp. generic group, signing oracles), the following bound holds:*

$$\Pr\left[\mathsf{SigForge}^{\tau,\mathsf{H}}_{\mathcal{A},\Pi}(k) = 1\right] \leq \varepsilon,$$

where the randomness is taken over the selection of τ*, the random coins of* $\mathcal{A}$*, the random coins of* Kg*, and the selection of random oracle* H*.*

3.1 Discrete Log Problem with Restricted Discrete Log Oracle

Restricted Discrete-Log Oracle in the Generic Group Model. In the discrete log problem we pick a random $x \in \mathbb{Z}_p$ and the attacker is challenged to recover x given $\mathfrak{g} = \tau(1)$ and $\mathfrak{h} = \mathtt{Pow}(\mathfrak{g}, x)$ after making queries to the generic group oracles $\mathtt{Mult}$ and $\mathtt{Inv}$. As we mentioned in Section 1.2, we analyze the discrete log problem in a stronger setting where the attacker is additionally given access to a restricted discrete-log oracle $\mathtt{DLog}$. Given the map $\tau : \mathbb{Z}_p \to \mathbb{G}$ and $y \in \mathbb{Z}_p$, $\mathtt{DLog}(\tau(y))$ will output y as long as $\tau(y)$ is a "fresh" group element. More specifically, we say $\tau(y)$ is "fresh" if (1) $\tau(y)$ is not equal to $\mathfrak{h}$, and (2) $\tau(y)$ has not been the output of a previous generic group query.

The requirement that $\tau(y)$ is fresh rules out trivial attacks where the attacker picks $a, b \in \mathbb{Z}_p$, computes $\tau(ax + b) = \mathtt{Mult}(\mathtt{Pow}(\mathfrak{h}, a), \mathtt{Pow}(\mathfrak{g}, b))$ and queries $\mathtt{DLog}(\tau(ax + b))$ and solves for $x = a^{-1}(\mathtt{DLog}(\tau(ax + b)) - b) \mod p$.

The Generic Discrete-Log Game. The formal definition of the discrete log experiment $\mathsf{DLogChal}^{\tau}_{\mathcal{A}}(k)$ is given below:

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

(1) The adversary $\mathcal{A}$ is given $(\mathfrak{g} = \tau(1), \tau(x))$ for a random value of $x \in \mathbb{Z}_p$. Here, $\tau : \mathbb{Z}_p \to \mathbb{G}$ is a map from $\mathbb{Z}_p$ to a generic group $\mathbb{G}$ with a $2k$-bit prime p.
(2) $\mathcal{A}$ is allowed to query the usual generic group oracles $(\mathtt{Mult}, \mathtt{Inv})$ and is additionally allowed to query $\mathtt{DLog}(\tau(y))$, but *only* if $\tau(y)$ is "fresh", i.e., $\tau(y)$ is not $\tau(x)$, 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 $\mathsf{DLogChal}^{\tau}_{\mathcal{A}}(k) = 1$ if $x' = x$, and 0 otherwise.

Lemma 1 upper bounds the probability that an attacker wins the generic discrete-log game $\mathsf{DLogChal}^{\tau}_{\mathcal{A}}(k)$. Intuitively, the proof works by maintaining a list $\mathcal{L}$ of tuples $(\tau(y), a, b)$ such that $y = ax + b$ for every oracle output $\tau(y)$.

Initially, the list $\mathcal{L}$ contains two items $(\tau(x), 1, 0)$ and $(\tau(1), 0, 1)$, and the list is updated after every query to the generic group oracles, e.g., if $(\tau(y_1), a_1, b_1) \in$

$\mathcal{L}$ and $(\tau(y_2), a_2, b_2) \in \mathcal{L}$, then querying $\mathtt{Mult}(\tau(y_1), \tau(y_2))$ will result in the addition of $(\tau(y_1 + y_2), a_1 + a_2, b_1 + b_2)$ into $\mathcal{L}$. If $\mathcal{L}$ already contained a tuple of the form $(\tau(y_1 + y_2), a', b')$ with $a' \neq a_1 + a_2$ or $b' \neq b_1 + b_2$, then we say that the event BRIDGE occurs.

We can use the restricted discrete log oracle to maintain the invariant that every output of our generic group oracles $\mathtt{Mult}$ and $\mathtt{Inv}$ can be added to $\mathcal{L}$. In particular, if we every encounter an input $\mathfrak{y} = \tau(b)$ that does not already appear in $\mathcal{L}$, then $\mathfrak{y}$ is fresh and we can simply query the restricted discrete log oracle to extract $b = \mathtt{DLog}(\mathfrak{y})$, ensuring that the tuple $(\mathfrak{y}, 0, b)$ is added to $\mathcal{L}$ before the generic group query is processed.

The key component of the proof is to upper bound the probability of the event BRIDGE. This is sufficient as any attacker that can recover x will also be able to ensure that $(\tau(x), 0, x)$ is added to $\mathcal{L}$, which would immediately cause the event BRIDGE to occur, since we already have $(\tau(x), 1, 0) \in \mathcal{L}$. We defer the full proof of Lemma 1 to Appendix C.1 for readers who are interested.

Lemma 1. *The probability the attacker making at most $q_{\mathtt{G}}$ generic group oracle queries wins the generic discrete-log game $\mathsf{DLogChal}^{\tau}_{\mathcal{A}}(k)$ (even with access to the restricted $\mathtt{DLog}$ oracle) is at most*

$$\Pr\left[\mathsf{DLogChal}^{\tau}_{\mathcal{A}}(k) = 1\right] \leq \frac{6q_{\mathtt{G}}(q_{\mathtt{G}} + 1) + 12}{4p - (3q_{\mathtt{G}} + 2)^2},$$

in the generic group model of prime order p, where the randomness is taken over the selection of τ, the challenge x, as well as any random coins of $\mathcal{A}$.

3.2 Security Reduction

Given Lemma 1, we are now ready to describe our security reduction for short Schnorr signatures of length $3k$. 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 $y = ax + b$ 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 $\mathtt{DLog}(\tau(y))$ so that we can add $\tau(y)$ to the list $\mathcal{L}$.

Theorem 4 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 4. *The* short *Schnorr signature scheme $\Pi_{\mathsf{short}} = (\mathsf{Kg}, \mathsf{Sign}, \mathsf{Vfy})$ of length $3k$ is $(q_{\mathsf{H}}, q_{\mathtt{G}}, q_{\mathtt{S}}, \varepsilon)$-$\mathsf{UF\text{-}CMA}$ secure with*

$$\varepsilon = \frac{6q_{\mathtt{G}}(q_{\mathtt{G}} + 1) + 12}{4p - (3q_{\mathtt{G}} + 2)^2} + \frac{q_{\mathtt{S}}(q_{\mathsf{H}} + q_{\mathtt{S}})}{p} + \frac{q_{\mathsf{H}} + q_{\mathtt{S}}}{p - (3q_{\mathtt{G}} + 2)} + \frac{q_{\mathsf{H}} + 1}{2^k} = \mathcal{O}\left(\frac{q}{2^k}\right),$$

in the generic group model of prime order $p \approx 2^{2k}$ and the programmable random oracle model, where q denotes the total number of queries made by an adversary.

Proof Sketch of Theorem 4: Here, we only give the intuition of how the proof works. The full proof of Theorem 4 is similar to that of Theorem 6 and we defer it to Appendix C.1.

We give the proof by reduction, i.e., given an adversary $\mathcal{A}_{\mathsf{sig}}$ attacking short Schnorr signature scheme, we construct an efficient algorithm $\mathcal{A}_{\mathsf{dlog}}$ which solves the discrete-log problem. During the reduction, we simulate the signature signing process without secret key x by programming the random oracle, i.e., to sign a message m we can pick s and e randomly, compute $I = \tau(s - xe)$ by querying adequate generic group oracles[2], and see if the random oracle has been previously queried at $\mathsf{H}(I\|m)$. If not, then we can program the random oracle as $\mathsf{H}(I\|m) := e$ and output the signature (s, e). Otherwise, the reduction simply outputs $\perp$ for failure. Since s and e are selected randomly, we can argue that the probability that we output $\perp$ because $\mathsf{H}(I\|m)$ is already defined is small, i.e., $\approx q_{\mathsf{H}}/p$.

We can use the oracle $\mathtt{DLog}$ to maintain the invariant that before processing any random oracle query of the form $\mathsf{H}(\mathfrak{y}\|\cdot)$ that we know a, b such that $\tau(ax + b) = \mathfrak{y}$. In particular, if $\mathfrak{y}$ is a fresh string that has not previously been observed, then we can simply set $a = 0$ and query the restricted discrete log oracle to find b such that $\tau(b) = \mathfrak{y}$. We say that the random oracle query is lucky if $\mathsf{H}(\mathfrak{y}\|m) = -a$, or $\mathsf{H}(\mathfrak{y}\|m) = 0$. Assuming that the event BRIDGE does not occur, it is straightforward to upper bound the probability of a lucky query as $\mathcal{O}(q_{\mathsf{H}}/2^k)$. Similarly, it is straightforward to show that the probability that $\mathcal{A}_{\mathsf{sig}}$ gets lucky and guesses a valid signature (s, e) for a message m without first querying $\mathsf{H}(\tau(s - xe)\|m)$ is $\mathcal{O}(2^{-k})$. Assuming that there are no lucky queries or guesses but the attacker still outputs a successful signature forgery (s, e). In this case we have $I = \tau(s - xe) = \tau(ax + b)$, which allows for us to solve for x using the equation $(a + e)x = (s - b)$. Thus, we can argue that the probability $\mathcal{A}_{\mathsf{dlog}}$ solves the discrete-log challenge correctly is lower bounded by the probability that $\mathcal{A}_{\mathsf{sig}}$ forges a signature minus $\mathcal{O}(q/2^k)$. Finally, by applying Lemma 1 we can upper bound the probability $\mathcal{A}_{\mathsf{sig}}$ wins the generic signature forgery game $\mathsf{SigForge}^{\tau,\mathsf{H}}_{\mathcal{A},\Pi}(k)$ to be $\mathcal{O}(q/2^k)$. □

4 Multi-User Security of Short Schnorr Signatures

In this section, we prove that short Schnorr signatures also provide k bits of security in the multi-user setting. The reduction uses similar ideas, but requires us to introduce and analyze a game called the *1-out-of-N generic* BRIDGE^N*-finding game*. We first define the *1-out-of-N generic signature forgery game*, where an adversary is given N independent public keys $(\mathsf{pk}_1, \ldots, \mathsf{pk}_N) = (\tau(\mathsf{sk}_1), \ldots, \tau(\mathsf{sk}_N))$ along with oracle access to the signing oracles $\mathsf{Sign}(\mathsf{sk}_1, \cdot), \ldots, \mathsf{Sign}(\mathsf{sk}_N, \cdot)$, the random oracle H, and the generic group oracles. The attacker can succeed if s/he can output a forgery (σ, m) which is valid under *any one* public key, e.g., for some public key pk_j we have $\mathsf{Vfy}(\mathsf{pk}_j, m, \sigma) = 1$, while the query m was never submitted to the jth signing oracle $\mathsf{Sign}(\mathsf{sk}_j, \cdot)$. In our reduction, we show that any attacker that wins the 1-out-of-N generic signature forgery game can be

[2] We can compute I without knowledge of x because $\tau(x)$ is given as public key.

used to win the 1-out-of-N generic BRIDGE^N-finding game. We separately upper bound the probability that a generic attacker can win the 1-out-of-N generic BRIDGE^N-finding game.

1-out-of-N Generic Signature Forgery Game. Fixing the injective mapping $\tau : \mathbb{Z}_p \to \mathbb{G}$, a random oracle H, and an adversary $\mathcal{A}$, consider the following experiment defined for a signature scheme $\Pi = (\mathsf{Kg}, \mathsf{Sign}, \mathsf{Vfy})$:

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

(1) $\mathsf{Kg}(1^k)$ is run N times to obtain the public and the secret keys $(\mathsf{pk}_i, \mathsf{sk}_i)$ for each $i \in [N]$. Here, for each $i \in [N]$, sk_i is chosen randomly from the group $\mathbb{Z}_p$, where p is a $2k$-bit prime, and $\mathsf{pk}_i = \tau(\mathsf{sk}_i)$.
(2) Adversary $\mathcal{A}$ is given $(\mathfrak{g} = \tau(1), \mathsf{pk}_1, \cdots, \mathsf{pk}_N, p)$, 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 $\mathsf{Sign}(\mathsf{sk}_1, \cdot), \ldots, \mathsf{Sign}(\mathsf{sk}_N, \cdot)$. The experiment ends when the adversary outputs $(m, \sigma = (s, e))$.
(3) $\mathcal{A}$ succeeds to forge a signature if and only if there exists some $j \in [N]$ such that $\mathsf{Vfy}(\mathsf{pk}_j, m, \sigma) = 1$ and the query m was never submitted to the oracle $\mathsf{Sign}(\mathsf{sk}_j, \cdot)$. The output of the experiment is $\mathsf{SigForge}^{\tau,\mathsf{H},N}_{\mathcal{A},\Pi}(k) = 1$ when $\mathcal{A}$ succeeds; otherwise $\mathsf{SigForge}^{\tau,\mathsf{H},N}_{\mathcal{A},\Pi}(k) = 0$.

Definition 2. *Consider the generic group model with an encoding map $\tau : \mathbb{Z}_p \to \mathbb{G}$. A signature scheme $\Pi = (\mathsf{Kg}, \mathsf{Sign}, \mathsf{Vfy})$ is $(N, q_\mathsf{H}, q_\mathtt{G}, q_\mathsf{S}, \varepsilon)$-*MU-UF-CMA *secure (multi-user unforgeable against chosen message attack) if for every adversary $\mathcal{A}$ making at most q_H (resp. $q_\mathtt{G}$, q_S) queries to the random oracle (resp. generic group, signing oracles), the following bound holds:*

$$\Pr\left[\mathsf{SigForge}^{\tau,\mathsf{H},N}_{\mathcal{A},\Pi}(k) = 1\right] \leq \varepsilon,$$

where the randomness is taken over the selection of τ, the random coins of $\mathcal{A}$, the random coins of Kg, and the selection of random oracle H.

The Discrete-Log Solution List $\mathcal{L}$ 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 $(\mathfrak{y}, \vec{a}, b)$ in a list $\mathcal{L}$ where $\mathtt{DLog}(\mathfrak{y}) = \vec{a} \cdot \vec{x} + b$ (here, $\vec{a} = (a_1, \ldots, a_N), \vec{x} = (x_1, \ldots, x_N) \in \mathbb{Z}_p^N$). Note that the restricted oracle $\mathtt{DLog}(\cdot)$ will solve $\mathtt{DLog}(\mathfrak{y})$ for any fresh group element $\mathfrak{y}$ such that $\mathfrak{y} \notin \{\tau(1), \tau(x_1), \ldots, \tau(x_N)\}$, and $\mathfrak{y}$ has not been the output of a prior generic group query.

- Initially, $\mathcal{L}$ contains $(\tau(1), \vec{0}, 1)$ and $(\tau(x_i), \hat{u}_i, 0)$ for $1 \leq i \leq N$.
- If the attacker ever submits a fresh group element $\mathfrak{y}$ which was not previously an output of a generic group oracle query, then we can query $b = \mathtt{DLog}(\mathfrak{y})$, and add $(\mathfrak{y}, \vec{0}, b)$ to our list. Thus, without loss of generality, we can assume that all query inputs to $\mathtt{Mult}, \mathtt{Inv}$ were first added to $\mathcal{L}$.
- If $(\mathfrak{y}_1, \vec{a}_1, b_1), (\mathfrak{y}_2, \vec{a}_2, b_2) \in \mathcal{L}$, and the attacker queries $\mathtt{Mult}(\mathfrak{y}_1, \mathfrak{y}_2)$, then add $(\mathtt{Mult}(\mathfrak{y}_1, \mathfrak{y}_2), \vec{a}_1 + \vec{a}_2, b_1 + b_2)$ to $\mathcal{L}$.
- If $(\mathfrak{y}, \vec{a}, b) \in \mathcal{L}$, and $\mathtt{Inv}(\mathfrak{y})$ is queried, then add $(\mathtt{Inv}(\mathfrak{y}), -\vec{a}, -b)$ to $\mathcal{L}$.

4.1 The Multi-User Bridge-Finding 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 $\tau(1)$, as well as $\tau(x_1), \ldots, \tau(x_N)$ for N randomly selected values $\vec{x} = (x_1, \ldots, x_N) \in \mathbb{Z}_p^N$. The key difference between this game and the 1-out-of-N discrete-log game is that the attacker's 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 Section 3, we will assume that we have access to $\mathtt{DLog}(\cdot)$, and we will maintain the invariant that for every output $\tau(y)$ of some generic group query, we have $y = \vec{a} \cdot \vec{x} + b$ for known values $\vec{a} = (a_1, \ldots, a_N) \in \mathbb{Z}_p^N$ and $b \in \mathbb{Z}_p$, i.e., by querying the restricted oracle $\mathtt{DLog}(\tau(y))$ whenever we encounter a fresh input.

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

(1) The challenger initializes the list $\mathcal{L} = \{(\tau(1), \vec{0}, 1), (\tau(x_1), \hat{u}_1, 0), \ldots, (\tau(x_N), \hat{u}_N, 0)\}$, and $\vec{x} = (x_1, \cdots, x_N)$.
(2) The adversary $\mathcal{A}$ is given $\mathfrak{g} = \tau(1)$ and $\tau(x_i)$ for each $i \in [N]$.
(3) $\mathcal{A}$ is allowed to query the usual generic group oracles $(\mathtt{Mult}, \mathtt{Inv})$.
  (a) 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 $b_y = \mathtt{DLog}(\mathfrak{y})$, and adds the tuple $(\mathfrak{y}, \vec{0}, b_y)$ to the list $\mathcal{L}$.
  (b) Whenever $\mathcal{A}$ submits a query $\mathfrak{y}_1, \mathfrak{y}_2$ to $\mathtt{Mult}(\cdot, \cdot)$, we are ensured that there exist tuples $(\mathfrak{y}_1, \vec{a}_1, b_1), (\mathfrak{y}_2, \vec{a}_2, b_2) \in \mathcal{L}$. The challenger adds the tuple $(\mathtt{Mult}(\mathfrak{y}_1, \mathfrak{y}_2), \vec{a}_1 + \vec{a}_2, b_1 + b_2)$ to the list $\mathcal{L}$.
  (c) Whenever $\mathcal{A}$ submits a query $\mathfrak{y}$ to $\mathtt{Inv}(\cdot)$, we are ensured that some tuple $(\mathfrak{y}, \vec{a}_y, b_y) \in \mathcal{L}$. The challenger adds the tuple $(\mathtt{Inv}(\mathfrak{y}), -\vec{a}_y, -b_y)$ to the list $\mathcal{L}$.
(4) If at any point in time we have a collision, i.e., two distinct tuples $(\mathfrak{y}, \vec{a}_1, b_1), (\mathfrak{y}, \vec{a}_1, b_1) \in \mathcal{L}$ with $(\vec{a}_1, b_1) \neq (\vec{a}_2, b_2)$, then the event BRIDGE^N occurs, and the output of the game is 1. If BRIDGE^N never occurs, then the output of the game is 0.

We further define the game $\mathsf{BridgeChal}_{\mathcal{A}}^{\tau,N}(k)$ in which $\vec{x} = (x_1, \ldots, x_N)$ are first sampled uniformly at random, and then we run $\mathsf{BridgeChal}_{\mathcal{A}}^{\tau,N}(k, \vec{x})$. Thus, we have

$$\Pr_{\mathcal{A},\tau}[\mathsf{BridgeChal}_{\mathcal{A}}^{\tau,N}(k) = 1] := \Pr_{\mathcal{A},\tau,\vec{x}}[\mathsf{BridgeChal}_{\mathcal{A}}^{\tau,N}(k, \vec{x}) = 1] \ .$$

As long as the event BRIDGE^N has not occurred, we can (essentially) view $x_1, \ldots, x_N$ as uniformly random values that are 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 $\mathfrak{f}_1 = \tau(\vec{a}_1 \cdot \vec{x} + b_1) \neq \mathfrak{f}_2 = \tau(\vec{a}_2 \cdot \vec{x} + b_2)$ then we have the constraint that $\vec{a}_1 \cdot \vec{x} + b_1 \neq \vec{a}_2 \cdot \vec{x} + b_2$.

Theorem 5. *For any attackers $\mathcal{A}$ making at most $q_{\mathtt{G}} := q_{\mathtt{G}}(k)$ queries to the generic group oracles,*

$$\Pr\left[\mathsf{BridgeChal}_{\mathcal{A}}^{\tau,N}(k)=1\right] \leq \frac{q_{\mathtt{G}}N + 3q_{\mathtt{G}}(q_{\mathtt{G}}+1)/2}{p-(N+3q_{\mathtt{G}}+1)^2-N},$$

in the generic group model of prime order p where the randomness is taken over the selection of $x_1,\ldots,x_N,\tau$ as well as any random coins of $\mathcal{A}$.

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 $\overline{\mathsf{BRIDGE}}^N_{<i}$ that the event has not yet occurred, i.e., the event BRIDGE^N has not been occurred until the $(i-1)^{th}$ query. Before we even receive the output $\mathfrak{y}_i$, we already know the values $\vec{a}_i, b_i$ such that the tuple $(\mathfrak{y}_i, \vec{a}_i, b_i)$ 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 $(\mathfrak{y}_i, \vec{a}_i, b_i)$, then we are interested in the event B_i that some other tuple $(\mathfrak{y}_i, \vec{a}'_i, b'_i)$ has been recorded with $(\vec{a}'_i, b'_i) \neq (\vec{a}_i, b_i)$. Observe that B_i occurs if and only if there exists a tuple of the form $(\cdot, \vec{a}, b)$ with $(\vec{a}-\vec{a}_i)\cdot\vec{x} = b_i - b$ and $(\vec{a}, b) \neq (\vec{a}_i, b_i)$. If we pick $\vec{x}$ randomly, the probability that $(\vec{a}-\vec{a}_i)\cdot\vec{x} = b_i - b$ would be $1/p$. However, we cannot quite view $\vec{x}$ as random due to the restrictions, i.e., because we condition of the event $\overline{\mathsf{BRIDGE}}^N_{<i}$ we know that for any distinct pair $(\mathfrak{y}_i, \vec{a}_i, b_i)$ and $(\mathfrak{y}_i, \vec{a}_j, b_j)$ we know that $\vec{a}_i\cdot\vec{x} + b_i \neq \vec{a}_j\cdot\vec{x} + b_j$.

Consider sampling $\vec{x}$ uniformly at random subject to this restriction. Let $r \leq N$ be an index such that $\vec{a}[r] - \vec{a}_i[r] \neq 0$ and suppose that $x_r = \vec{x}[r]$ 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 $p - |\mathcal{L}|^2 - (N-1)$ remaining values, subject to all of the restrictions. We also observe that $|\mathcal{L}| \leq N + 3q_{\mathtt{G}} + 1$ since each generic group oracle query adds *at most* three new tuples to $\mathcal{L}$ — exactly three in the case that we query $\mathtt{Mult}(\mathfrak{y}_1, \mathfrak{y}_2)$ on two fresh elements. Thus, the probability that $(\vec{a}-\vec{a}_i)\cdot\vec{x} = b_i - b$ is at most $\frac{1}{p-(N+3q_{\mathtt{G}}+1)^2-(N-1)}$. Union bounding over all tuples $(\cdot, \vec{a}, b) \in \mathcal{L}$, we have

$$\Pr\left[B_i : \overline{\mathsf{BRIDGE}}^N_{<i}\right] \leq \frac{N+3i}{p-(N+3q_{\mathtt{G}}+1)^2-N}\ .$$

To complete the proof, we observe that

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

□

As an immediate corollary of Theorem 5, we can show that an attacker wins the 1-out-of-N discrete log game with (approximately) the same probability as in the multi-user bridge-finding game. In particular, given any attacker $\mathcal{A}'$

in the 1-out-of-N discrete-log game $\mathsf{1ofNDLog}^{\tau,N}_{\mathcal{A}}(k)$, where the attacker's goal is to output *any* $x \in \{x_1, \ldots, x_N\}$ given input $\tau(1), \tau(x_1), \ldots, \tau(x_N)$, we can construct an attacker $\mathcal{A}$ in the game $\mathsf{BridgeChal}^{\tau,N}_{\mathcal{A}}(k)$. $\mathcal{A}$ simply runs $\mathcal{A}'$ to obtain an output x, and then computes $\tau(x)$ using at most $2\log p$ 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 $(\tau(x), \vec{0}, x) \in \mathcal{L}$ and $(\tau(x), \hat{u}_i, 0) \in \mathcal{L}$ for some $i \in [N]$.

Corollary 1. *For any attacker $\mathcal{A}$ making at most $q_{\mathtt{G}} + 2\log p$ queries,*

$$\Pr\left[\mathsf{1ofNDLog}^{\tau,N}_{\mathcal{A}}(k) = 1\right] \leq \frac{q_{\mathtt{G}}N + 3q_{\mathtt{G}}(q_{\mathtt{G}}+1)/2}{p - (N + 3q_{\mathtt{G}} + 1)^2 - N},$$

in the generic group model of prime order p, where the randomness is taken over the selection of τ, the challenges $x_1, \ldots, x_N$, and any random coins of $\mathcal{A}$.

4.2 Security Reduction

Theorem 6. *The* short *Schnorr signature scheme $\Pi_{\mathsf{short}} = (\mathsf{Kg}, \mathsf{Sign}, \mathsf{Vfy})$ of length $3k$ is $\left(N, q_{\mathsf{H}}, q_{\mathtt{G}}, q_{\mathtt{S}}, \varepsilon = \mathcal{O}\left(\frac{q+N}{2^k}\right)\right)$-MU-UF-CMA secure with*

$$\varepsilon = \frac{q_{\mathtt{G}}N + 3q_{\mathtt{G}}(q_{\mathtt{G}}+1)/2}{p - (N + 3q_{\mathtt{G}} + 1)^2 - N} + \frac{q_{\mathtt{S}}(q_{\mathsf{H}} + q_{\mathtt{S}})}{p} + \frac{q_{\mathsf{H}} + q_{\mathtt{S}}}{p - (N + 3q_{\mathtt{G}} + 1)} + \frac{q_{\mathsf{H}} + 1}{2^k},$$

in the generic group model of prime order $p \approx 2^{2k}$ and the programmable random oracle model, where q denotes the total number of queries made by an adversary.

Proof. Given an adversary $\mathcal{A}_{\mathsf{sig}}$ attacking short 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 $\mathsf{BridgeChal}^{\tau,N}_{\mathcal{A}_{\mathsf{bridge}}}(k)$:

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

1. Initialize the list $\mathcal{L} = \{(\tau(1), \vec{0}, 1), (\tau(x_i), \hat{u}_i, 0)$ for each $i \in [N]\}$, and $\mathsf{H}_{\mathsf{resp}} = \{\}$, where $\mathsf{H}_{\mathsf{resp}}$ 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 2. Now we consider the following cases:
   (a) Whenever $\mathcal{A}_{\mathsf{sig}}$ submits a query w to the random oracle H:
      – If there is a pair $(w, R) \in \mathsf{H}_{\mathsf{resp}}$ for some string R, then return R.
      – Otherwise, select $R \leftarrow\!\!\$\, \mathbb{Z}_{2^k}$, and add (w, R) to the set $\mathsf{H}_{\mathsf{resp}}$.
      – If w has the form $w = (\mathfrak{a} \| m_i)$, where the value $\mathfrak{a}$ has not been observed previously (i.e., is not in the list $\mathcal{L}$), then we query $b = \mathtt{DLog}(\mathfrak{a})$, and add $(\mathfrak{a}, \vec{0}, b)$ 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 $\mathcal{L}$ then we immediately query $b = \mathtt{DLog}(\mathfrak{a})$ and add $(\mathfrak{a}, \vec{0}, b)$ to $\mathcal{L}$.

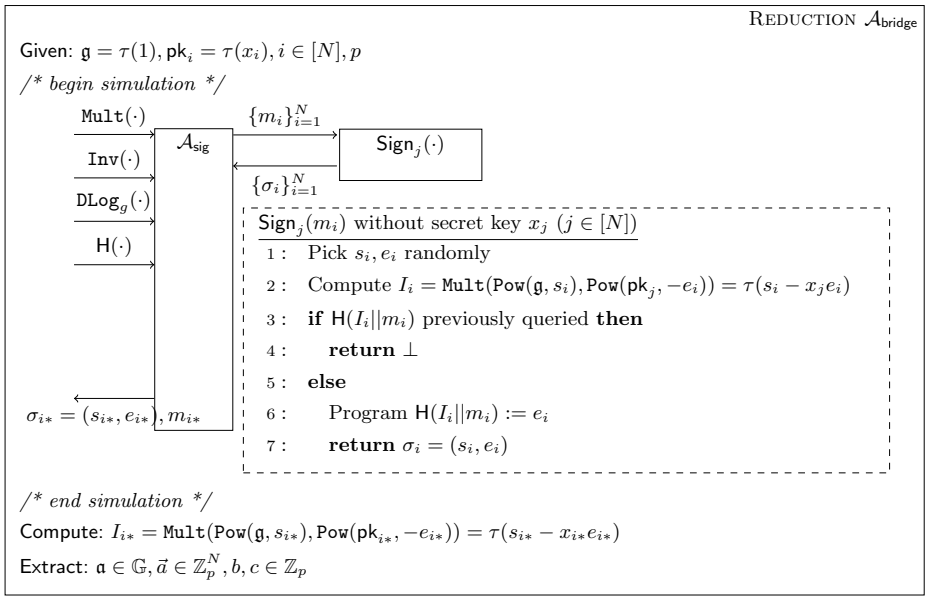

Figure 2. A reduction to the $\mathsf{BridgeChal}^{\tau,N}_{\mathcal{A}_{\mathsf{bridge}}}(k)$ attacker $\mathcal{A}_{\mathsf{bridge}}$ from the short Schnorr signature attacker $\mathcal{A}_{\mathsf{sig}}$.

  - Otherwise, $(\mathfrak{a}, \vec{a}, b) \in \mathcal{L}$. Then we query $\mathtt{Inv}(\mathfrak{a}) = \tau(-\vec{a} \cdot \vec{x} - b)$, output the result and add the result $(\tau(-\vec{a} \cdot \vec{x} - b), -\vec{a}, -b) \in \mathcal{L}$.

(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 $b_0 = \mathtt{DLog}(\mathfrak{a})$ (resp. $b_1 = \mathtt{DLog}(\mathfrak{b})$), and add the element $(\mathfrak{a}, \vec{0}, b_0)$ (resp. $(\mathfrak{b}, \vec{0}, b_1)$) to $\mathcal{L}$.
  - Otherwise, both elements $(\mathfrak{a}, \vec{a}_0, b_0), (\mathfrak{b}, \vec{a}_1, b_1) \in \mathcal{L}$. Then we return $\mathtt{Mult}(\mathfrak{a}, \mathfrak{b}) = \tau((\vec{a}_0 + \vec{a}_1) \cdot \vec{x} + b_0 + b_1)$, and add $(\tau((\vec{a}_0 + \vec{a}_1) \cdot \vec{x} + b_0 + b_1), \vec{a}_0 + \vec{a}_1, b_0 + b_1) \in \mathcal{L}$.

(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 2 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 tuples $(\tau(s_i), \vec{0}, s_i)$, $(\tau(x_j e_i), e_i \hat{u}_i, 0)$ and $(\tau(s_i - x_j e_i), -e_i \hat{u}_i, s_i)$ to $\mathcal{L}$, since these values are computed using the generic group oracles $\mathtt{Inv}$ and $\mathtt{Mult}$.

(e) If at any point we find some string $\mathfrak{y}$ such that $(\mathfrak{y}, \vec{a}, b) \in \mathcal{L}$ and $(\mathfrak{y}, \vec{c}, d) \in \mathcal{L}$ for $(\vec{a}, b) \neq (\vec{c}, d)$, then we can immediately have a BRIDGE^N instance $(\tau((\vec{a} - \vec{c}) \cdot \vec{x}), \vec{a} - \vec{c}, 0) \in \mathcal{L}$ and $(\tau(d - b), \vec{0}, d - b) \in \mathcal{L}$ since $\tau((\vec{a} - \vec{c}) \cdot \vec{x}) = \tau(d - b)$.[3] Thus, without loss of generality, we can assume that each string $\mathfrak{y}$ occurs at most once in the list $\mathcal{L}$.

[3] Note that $(\vec{a}, b) \neq (\vec{c}, d)$ implies $\vec{a} \neq \vec{c}$ since if $\vec{a} = \vec{c}$ then $\vec{a} \cdot \vec{x} + b = \vec{a} \cdot \vec{x} + d$ implies $b = d$ as $b, d \in \mathbb{Z}_p$.

3. After $\mathcal{A}_{\mathsf{sig}}$ outputs $\sigma_{i*} = (s_{i*}, e_{i*})$ and m_{i*}, identify the index $i* \in [N]$ such that $\mathsf{Vfy}(\mathsf{pk}_{i*}, m_{i*}, \sigma_{i*}) = 1$.
4. Compute $\tau(-e_{i*}x_{i*}) = \mathtt{Inv}(\mathtt{Pow}(\tau(x_{i*}), e_{i*}))$ and $\mathfrak{s}_{i*} = \mathtt{Pow}(\mathfrak{g}, s_{i*})$. This will ensure that the elements $(\tau(-e_{i*}x_{i*}), -e_{i*}\hat{u}_{i*}, 0)$ and $(\tau(e_{i*}x_{i*}), e_{i*}\hat{u}_{i*}, 0)$, and $(\mathfrak{s}_{i*}, \vec{0}, s_{i*})$ are all added to $\mathcal{L}$.
5. Compute $I_{i*} = \mathtt{Mult}(\mathfrak{s}_{i*}, \tau(-e_{i*}x_{i*})) = \tau(s_{i*} - x_{i*}e_{i*})$ which ensures that $(I_{i*}, -e_{i*}\hat{u}_{i*}, s_{i*}) \in \mathcal{L}$. Finally, we can check to see if we previously had any tuple of the form $(I_{i*}, \vec{a}, b) \in \mathcal{L}$.

Analysis. We first remark that if the signature is valid then we must have $e_{i*} = \mathsf{H}(I_{i*} \| m_{i*})$ and $\mathtt{DLog}(I_{i*}) = s_{i*} - x_{i*}e_{i*} = \vec{a} \cdot \vec{x} + b$.

We now define failure events $\mathsf{FailtoFind}(I_{i*})$ and $\mathsf{BadQuery}$. $\mathsf{FailtoFind}(I_{i*})$ denotes the event that we find that the signature is valid, but I_{i*} was not previously recorded in our list $\mathcal{L}$ before we computed $\mathtt{Mult}(\mathfrak{s}_{i*}, \tau(-e_{i*}x_{i*}))$ in the last step. Similarly, let $\mathsf{BadQuery}$ denote the event that the signature is valid but for the only prior tuple $(I_{i*}, \vec{a}, b) \in \mathcal{L}$ recorded in $\mathcal{L}$ we have that $\vec{a} = -e_{i*}\hat{u}_{i*}$. If the signature is valid and neither of the events $\mathsf{FailtoFind}(I_{i*})$ and $\mathsf{BadQuery}$ occur, then the bridge event BRIDGE^N must have occurred and we immediately win the game since $(I_{i*}, \vec{a}, b) \in \mathcal{L}$, $(I_{i*}, -e_{i*}\hat{u}_{i*}, s_{i*}) \in \mathcal{L}$ and $\vec{a} \neq -e_{i*}\hat{u}_{i*}$.

We additionally consider then the event $\mathsf{FailtoSign}$ where our reduction outputs $\perp$ in Step 2.(d) due signing oracle failure i.e., because $\mathsf{H}(I_i \| m_i)$ has been queried previously. Intuitively, the attacker will output a valid signature forgery with probability at least $\Pr[\mathsf{SigForge}^{\tau,N}_{\mathcal{A}_{\mathsf{sig}}, \Pi_{\mathsf{short}}}(k) = 1] - \Pr[\mathsf{FailtoSign}]$ after we replace the signing oracle with the procedure Sign_j described in Figure 2.

Claim 1, Claim 2, and Claim 3 upper bound the probability of our events $\mathsf{FailtoSign}$, $\mathsf{FailtoFind}$ and $\mathsf{BadQuery}$ respectively. We defer the proofs to Appendix C.2.

Claim 1. $\Pr[\mathsf{FailtoSign}] \leq \dfrac{q_{\mathsf{S}}(q_{\mathsf{H}} + q_{\mathsf{S}})}{p}$.

Claim 2. $\Pr[\mathsf{FailtoFind}(I_{i*})] \leq \dfrac{q_{\mathsf{H}} + q_{\mathsf{S}}}{p - |\mathcal{L}|} + \dfrac{1}{2^k}$.

Claim 3. $\Pr[\mathsf{BadQuery}] \leq \dfrac{q_{\mathsf{H}}}{2^k}$.

Since we have $|\mathcal{L}| \leq N + 3q_{\mathsf{G}} + 1$, we can apply Theorem 5 to conclude that

$$\begin{aligned}
&\Pr[\mathsf{SigForge}^{\tau,N}_{\mathcal{A}_{\mathsf{sig}}, \Pi_{\mathsf{short}}}(k) = 1] \\
&\leq \Pr[\mathsf{BridgeChal}^{\tau,N}_{\mathcal{A}_{\mathsf{bridge}}}(k) = 1] + \Pr[\mathsf{FailtoSign}] + \Pr[\mathsf{FailtoFind}(I_{i*})] + \Pr[\mathsf{BadQuery}] \\
&\leq \frac{q_{\mathsf{G}}N + 3q_{\mathsf{G}}(q_{\mathsf{G}} + 1)/2}{p - (N + 3q_{\mathsf{G}} + 1)^2 - N} + \frac{q_{\mathsf{S}}(q_{\mathsf{H}} + q_{\mathsf{S}})}{p} + \frac{q_{\mathsf{H}} + q_{\mathsf{S}}}{p - (N + 3q_{\mathsf{G}} + 1)} + \frac{q_{\mathsf{H}} + 1}{2^k} \\
&= \mathcal{O}\left(\frac{q + N}{2^k}\right) .
\end{aligned}$$

□

Remark 1. Kiltz et al. [KMP16] proved that multi-user security of *regular* Schnorr signatures is tightly equivalent to the Q-IDLOG problem in the random oracle model. Here, the Q-IDLOG problem is defined as follows: an adversary is given g^x and access to a challenge oracle CH, where $\text{CH}(g^{r_i})$ returns a uniformly random element $h_i \in \mathbb{Z}_p$. After up to Q queries to CH, the adversary wins if it returns a value s from the set $\{xh_i + r_i \mid i \in [Q]\}$. Kiltz et al. [KMP16, Theorem A.1] also proved that a generic attacker making q generic group queries and Q queries to CH wins with probability proportional to $(q^2 + Q)/p$.

In private communication, the authors of [KMP16] pointed out that Theorem A.1 could be updated to give the bound $q^2/p + Q/2^k$ when CH returns a random element $h_i \in \{0,1\}^k$ as in short Schnorr signatures. This would also imply tight multi-user security of short Schnorr signatures in the random oracle model and (their version of) the generic group model. However, short Schnorr signatures are not discussed in [KMP16]. We prefer to work in Shoup's generic group model as it is more suitable for analyzing protocol composition and preprocessing attacks.

5 Multi-User Security of Short Schnorr Signatures with Key-Prefixing against Preprocessing Attacks

In this section, we analyze the security of short Schnorr signatures against a preprocessing attacker who first outputs an S-bit hint after making (a very large number of) preprocessing queries to the generic group oracles `Mult` and `Inv`, as well as the random oracle H. After the public/secret keys are chosen, the signature forgery attacker will try use the hint to help win the signature forgery game. The hint must be fixed *before* the public/secret keys for our signature scheme are selected, otherwise the preprocessing attacker can generate forged signatures and embed them in the hint.

We first observe that Schnorr signatures are trivially broken against a preprocesssing attack, e.g., if the preprocessing attacker finds some message m and an integer r such that $e = \mathsf{H}(\tau(r)\|m) = 0$, then the attacker can simply include the tuple (m, r) as part of the S-bit hint. Observe that the hint is completely independent of the public key pk. In fact, for any public key pk, we have that $\sigma' = (s = r, e = 0)$ is a valid signature for the message m! To see this, note that $R = \tau(s - \mathsf{sk} \cdot 0) = \tau(r) = 0$ and that, by assumption, $\mathsf{H}(R\|m) = \mathsf{H}(\tau(r)\|m) = 0 = e$.

The above attack can easily be addressed with key-prefixing, i.e., to sign a message m, we pick an integer r, compute $e = \mathsf{H}(\mathsf{pk}\|\tau(r)\|m)$, and output the signature $\sigma = (s = r + \mathsf{sk} \cdot e, e)$. Intuitively, since the preprocessing attacker does not know the public key pk in advance, s/he is unlikely to have stored a tuple of the form $(\mathsf{pk}, m, r, \tau(r))$. The key question is whether or not *short Schnorr signatures with key-prefixing* are secure against *any* preprocessing attack. To prove that short Schnorr signatures with key-prefixing are secure against preprocessing attacks, we revisit the 1-out-of-N bridge-finding game in the preprocessing setting.

5.1 Security of BRIDGE^N-Finding Game with Preprocessing

We analyze the BRIDGE^N-finding game in the setting with preprocessing attacks. In particular, an attacker consists of a pair of algorithms $(\mathcal{A}_{\mathsf{pre}}, \mathcal{A}_{\mathsf{on}})$. The basic idea is that we split the attack into two phases, *preprocessing* and *online* phase, so that the attacker has (exponential) time to make preprocessing queries before playing the bridge game. Specifically,

- Algorithm $\mathcal{A}_{\mathsf{pre}}$ runs a preprocessing phase, where it takes as input $\mathfrak{g} = \tau(1)$ and outputs a hint str_τ, which is a binary string after making queries to the generic group oracles $\mathtt{GO} = (\mathtt{Mult}(\cdot,\cdot), \mathtt{Inv}(\cdot))$. Without loss of generality, we can assume that $\mathcal{A}_{\mathsf{pre}}$ is deterministic, and we simply use str_τ to refer to the hint when the random mapping τ is fixed.
- The online attacker $\mathcal{A}_{\mathsf{on}}$ attempts to win the BRIDGE^N-finding game. The online attacker $\mathcal{A}_{\mathsf{on}}$ is given the hint str_τ (which was produced in the preprocessing phase), as well as $(\tau(x_1), \ldots, \tau(x_N))$. However, the challenger picks $(x_1, \ldots, x_N) \in \mathbb{Z}_p^N$ *after* the hint str_τ is fixed. For convenience, we will write $\mathcal{A}_{\mathsf{on},\mathsf{str}_\tau}$ to denote the online attacker with the hint str_τ hardcoded.

We are interested in the setting where the preprocessing algorithm $\mathcal{A}_{\mathsf{pre}}$ can make $q_{\mathsf{G}}^{\mathsf{pre}} \geq 2^{2k}$ queries to the generic group oracles. In other words, the preprocessing algorithm $\mathcal{A}_{\mathsf{pre}}$ can examine the entire input/output table of the mapping τ. However, the length of the hint str_τ given to the online attacker is bounded by S, and the online attacker can make at most $q_{\mathsf{G}}^{\mathsf{on}} < 2^k$ queries to the generic group oracles. Theorem 7 says that the probability of a successful preprocessing attack is at most $\widetilde{\mathcal{O}}(SN(q_{\mathsf{G}}^{\mathsf{on}})^2/p)$.

Theorem 7. *Let $p > 2^{2k}$ be a prime number and $N \in \mathbb{N}$ be a parameter. Let $(\mathcal{A}_{\mathsf{pre}}, \mathcal{A}_{\mathsf{on}})$ be a pair of generic algorithms with an encoding map $\tau : \mathbb{Z}_p \to \mathbb{G}$ such that $\mathcal{A}_{\mathsf{pre}}$ outputs an S-bit hint and $\mathcal{A}_{\mathsf{on}}$ makes at most $q_{\mathsf{G}}^{\mathsf{on}} := q_{\mathsf{G}}^{\mathsf{on}}(k)$ queries to the generic group oracles. Then*

$$\Pr\left[\mathsf{BridgeChal}^{\tau,N}_{\mathcal{A}_{\mathsf{on},\mathsf{str}_\tau}}(k) = 1\right] \leq \widetilde{\mathcal{O}}\left(\frac{SN(q_{\mathsf{G}}^{\mathsf{on}} + N)(q_{\mathsf{G}}^{\mathsf{on}} + 2N)}{p}\right),$$

where the randomness is taken over the selection of τ, the random coins of $\mathcal{A}_{\mathsf{on}}$, and the random coins used by the challenger in the bridge game (the hint $\mathsf{str}_\tau = \mathcal{A}_{\mathsf{pre}}^{\mathtt{GO}}(\mathfrak{g})$ is selected independently of the random coins used by the challenger). In particular, if $q_{\mathsf{G}}^{\mathsf{on}} \geq 10N(1 + 2\log p)$ and $S \geq 10\log(8p)$, then

$$\Pr\left[\mathsf{BridgeChal}^{\tau,N}_{\mathcal{A}_{\mathsf{on},\mathsf{str}_\tau}}(k) = 1\right] \leq \frac{12SN(q_{\mathsf{G}}^{\mathsf{on}})^2 \log p}{p}.$$

Remark 2. The upper bound is essentially tight as a preprocessing attacker can solve a random 1-out-of-N discrete-log challenge with probability $\widetilde{\Omega}((q_{\mathsf{G}}^{\mathsf{on}})^2 S/p)$ which would trivially allow the attacker to win the bridge-finding game. In particular, even when $N = 1$, there is a preprocessing with success probability $\Omega((q_{\mathsf{G}}^{\mathsf{on}})^2 S/p)$, e.g., see [CK18, Section 7.1]. Thus, our upper bound is tight up to a factor of N. ◁

The proof of Theorem 7 closely follows [CK18, Theorem 2] with a few minor modifications, and the full proof can be found in Appendix C.3. One small difference is that we need to extend the proofs of [CK18] to handle queries to the inverse oracle $\mathtt{Inv}(\cdot)$, and the restricted discrete log oracle $\mathtt{DLog}$. The proof of Theorem 7 relies on Lemma 2, which is similar to [CK18, Lemma 4]. Intuitively, if the preprocessing attack is too successful, then one can derive a contradiction by compressing the random mapping τ.

Lemma 2. *Let $\mathbb{G}$ be the set of binary strings of length ℓ such that $2^\ell \geq p$ for a prime p. Let $\mathcal{T} = \{\tau_1, \tau_2, \ldots\}$ be a subset of the labeling functions from $\mathbb{Z}_p$ to $\mathbb{G}$. Let $(\mathcal{A}_{\mathsf{pre}}, \mathcal{A}_{\mathsf{on}})$ be a pair of generic algorithms for $\mathbb{Z}_p$ on $\mathbb{G}$ such that for every $\tau \in \mathcal{T}$ and every $\vec{x} = (x_1, \ldots, x_N) \in \mathbb{Z}_p^N$, $\mathcal{A}_{\mathsf{pre}}$ outputs an S-bit advice string, $\mathcal{A}_{\mathsf{on}}$ makes at most q_{on} oracle queries, and $(\mathcal{A}_{\mathsf{pre}}, \mathcal{A}_{\mathsf{on}})$ satisfy $\Pr_{\mathcal{A}_{\mathsf{on}}}\left[\mathsf{BridgeChal}^{\tau,N}_{\mathcal{A}_{\mathsf{on}},\mathsf{str}_\tau}(k,\vec{x}) = 1\right] \geq \varepsilon$, where $\mathsf{str}_\tau = \mathcal{A}^{\mathtt{GO}}_{\mathsf{pre}}(\tau(1))$. Then, there exists a randomized encoding scheme that compresses elements of $\mathcal{T}$ to bitstrings of length at most*

$$\log \frac{|\mathbb{G}|!}{(|\mathbb{G}|-p)!} + S + 1 - \frac{\varepsilon p}{6q_{\mathsf{on}}(q_{\mathsf{on}}+N)(N\log p + 1)},$$

and succeeds with probability at least $1/2$.

The full proof of Lemma 2 can be found in Appendix C.3. Here, we only give the brief idea as follows. To compress τ, our encoding algorithm first runs $\mathcal{A}_{\mathsf{pre}}$ to extract an S-bit hint str_τ. We then execute $\mathcal{A}_{\mathsf{on}}(\mathsf{str}_\tau, \tau(x_1), \ldots, \tau(x_N))$ multiple times with different challenges $x_1, \ldots, x_N$. During each execution we record the responses to the new generic group oracle queries, so that the decoder can also execute $\mathcal{A}_{\mathsf{on}}(\mathsf{str}_\tau, \tau(x_1), \ldots, \tau(x_N))$. Intuitively, whenever the BRIDGE^N event occurs, the decoder can save a few bits by simply recording the index of prior query involved in the collision. This requires just $\log q_{\mathsf{on}}$ bits to encode instead of $\log p$ bits.

5.2 Multi-User Security of Key-Prefixed Short Schnorr Signatures with Preprocessing

Theorem 7 upper bounds the probability that a preprocessing attacker wins the multi-user bridge-finding game. In this setting, we observe that the hint $\mathsf{str} := \mathsf{str}_{\tau,\mathsf{H}}$ that the preprocessing attacker outputs may depend both on the random oracle H as well as the encoding map τ. We show how to adapt our prior reduction to establish the multi-user security of key-prefixed short Schnorr signatures against preprocessing attackers. Recall that in our reduction, we simulated a signature forgery attacker for (non key-prefixed) short Schnorr signatures responding to queries to the signing oracle by programming the random oracle. In the preprocessing setting without key-prefixing, the reduction breaks down immediately. For example, the probability of a lucky random oracle query $\mathsf{H}(\tau(r)\|m) = 0$ is no longer $\approx q_{\mathsf{H}}^{\mathsf{on}}/2^k$, since the preprocessing attacker can simply hardcode the pair (r, m) as part of the hint $\mathsf{str} := \mathsf{str}_{\tau,\mathsf{H}}$. Similarly, the

hint $\mathsf{str} := \mathsf{str}_{\tau,\mathsf{H}}$ may be correlated with particular input/output pairs from the random oracle, making it infeasible to program those points.

We address this challenge by considering a model where a preprocessing attacker is *time-bounded*, i.e., the preprocessing attacker can look at the entire generic group oracles but only allowed to query the random oracle at *up to* $q_{\mathsf{H}}^{\mathsf{pre}} = 2^{3k}$ points during the preprocessing phase. We leave it as an interesting theoretical challenge whether or not the bounds can be extended to unbounded preprocessing attacks. However, we would argue that in practice, 2^{3k} greatly overestimates the running time of any preprocessing attacker, e.g., if $k = 112$, then $2^{3k} = 2^{336}$. Intuitively, the signing oracle for key-prefixed short Schnorr signatures involves two random points: a public key $\mathsf{pk} \in \mathbb{G}$ and a random value $r \leftarrow_{\$} \mathbb{Z}_p$. The probability that a preprocessing attacker submitted a query of the form $\mathsf{H}(\mathsf{pk}, \tau(r), \cdot)$ is at most $q_{\mathsf{H}}^{\mathsf{pre}} p^{-2} \leq 2^{-k}$, since the $q_{\mathsf{H}}^{\mathsf{pre}}$ random oracle queries are fixed before pk and r are sampled.

In our analysis, we consider the bad event that the signing oracle queries the random oracle at a point $\mathsf{H}(\mathsf{pk}_i \| \tau(r_j) \| m)$, which was previously queried by the preprocessing attacker. Note that if this bad event never occurs, then we can view $\mathsf{H}(\mathsf{pk}_i \| \tau(r_j) \| m)$ as a uniformly random string that is uncorrelated with the attacker's state. The probability of this bad event occurring on any single query to the signing oracle is at most $N q_{\mathsf{H}}^{\mathsf{pre}} / p^2$. In particular, fixing an arbitrary set of $q_{\mathsf{H}}^{\mathsf{pre}}$ random oracle queries and then sampling $\mathsf{pk}_1, \ldots, \mathsf{pk}_N \in \mathbb{G}$ and $r \in \mathbb{Z}_p$, we can apply union bounds to argue that the probability that the preprocessing attacker previously submitted some query of the form $\mathsf{H}(\mathsf{pk}_i, \tau(r), \cdot)$ for any i is at most $N q_{\mathsf{H}}^{\mathsf{pre}} / p^2$. Union bounding over the $q_{\mathsf{S}}^{\mathsf{on}}$ online queries to the signing oracle, the probability of the bad event ever occurring on any query to the signing oracle is at most $N q_{\mathsf{H}}^{\mathsf{pre}} q_{\mathsf{S}}^{\mathsf{on}} / p^2$. Assuming that the bad event never occurs, we can safely program the random oracle to simulate queries to the signing oracle when we simulate our signature forgery attacker.

The other challenge that arises in the preprocessing setting is upper bounding the probability of the bad event that the attacker forges a signature without causing the bridge event to occur. Previously, our argument relied on the observation that for "fresh" group elements $r \in \mathbb{Z}_p$, we can effectively view $\tau(r)$ as random bit string that is yet to be fixed. This intuition does not carry over into the preprocessing setting, as the hint str might be correlated with $\tau(r)$. We address these challenges by applying a random oracle compression argument. In particular, if the attacker can generate forged signatures without causing the bridge event to occur, we can use this attacker to predict random oracle outputs, allowing us to derive a contraction by compressing the random oracle.

Theorem 8. *Let $\Pi = (\mathsf{Kg}, \mathsf{Sign}, \mathsf{Vfy})$ be a key-prefixed Schnorr signature scheme and $p > 2^{2k}$ be a prime number. Let $N \in \mathbb{N}$ be a parameter and $(\mathcal{A}_{\mathsf{sig}}^{\mathsf{pre}}, \mathcal{A}_{\mathsf{sig}}^{\mathsf{on}})$ be a pair of generic algorithms with an encoding map $\tau : \mathbb{Z}_p \to \mathbb{G}$ such that $\mathcal{A}_{\mathsf{sig}}^{\mathsf{pre}}$ makes at most $q_{\mathsf{H}}^{\mathsf{pre}}$ queries to the random oracle $\mathsf{H} : \{0,1\}^* \to \{0,1\}^{k_1}$ and outputs an S-bit hint $\mathsf{str}_{\tau,\mathsf{H}}$, and $\mathcal{A}_{\mathsf{sig}}^{\mathsf{on}}$ makes at most $q_{\mathsf{G}}^{\mathsf{on}} := q_{\mathsf{G}}^{\mathsf{on}}(k)$ queries to the generic group oracles and at most $q_{\mathsf{H}}^{\mathsf{on}}$ queries to the random oracle. Then*

$$\Pr\left[\mathsf{SigForge}^{\tau,N}_{\mathcal{A}^{\mathsf{on}}_{\mathsf{sig}},\mathsf{str}_{\tau,\mathsf{H}},\Pi}(k)=1\right] \le \varepsilon, \text{ with}$$

$$\varepsilon = \widetilde{\mathcal{O}}\left(\frac{SN(q_{\mathsf{G}}^{\mathsf{on}}+N)(q_{\mathsf{G}}^{\mathsf{on}}+2N)}{p}\right) + \frac{Nq_{\mathsf{H}}^{\mathsf{pre}}q_{\mathsf{S}}^{\mathsf{on}}}{p^2} + \frac{q_{\mathsf{S}}^{\mathsf{on}}(q_{\mathsf{S}}^{\mathsf{on}}+q_{\mathsf{H}}^{\mathsf{on}})}{p} + \frac{4(q_{\mathsf{H}}^{\mathsf{on}}+1)}{2^{k_1}} + \frac{N^2(S+k_1)}{p},$$

where $q_{\mathsf{S}}^{\mathsf{on}}$ denotes the number of queries to the signing oracle and the randomness is taken over the selection of τ and the random coins of $\mathcal{A}_{\mathsf{sig}}^{\mathsf{on}}$ (the hint $\mathsf{str}_{\tau,\mathsf{H}} = \mathcal{A}_{\mathsf{sig}}^{\mathsf{pre},\mathsf{GD}}(\mathsf{g})$ is selected independently of the random coins used by the challenger). In particular, if $q_{\mathsf{G}}^{\mathsf{on}} \ge 10N(1+2\log p)$ and $S \ge 10\log(8p)$, then

$$\varepsilon = \frac{12SN(q_{\mathsf{G}}^{\mathsf{on}})^2\log p}{p} + \frac{Nq_{\mathsf{H}}^{\mathsf{pre}}q_{\mathsf{S}}^{\mathsf{on}}}{p^2} + \frac{q_{\mathsf{S}}^{\mathsf{on}}(q_{\mathsf{S}}^{\mathsf{on}}+q_{\mathsf{H}}^{\mathsf{on}})}{p} + \frac{4(q_{\mathsf{H}}^{\mathsf{on}}+1)}{2^{k_1}} + \frac{N^2(S+k_1)}{p}.$$

Remark 3. The upper bound in Theorem 8 is essentially tight (up to a factor of N), because of the following observations:

- Making the reasonable assumption that $Nq_{\mathsf{H}}^{\mathsf{pre}}q_{\mathsf{S}}^{\mathsf{on}} < 2^{3k}$ and $q_{\mathsf{G}}^{\mathsf{on}} > \sqrt{N}$, the dominating terms in ε are $\widetilde{\mathcal{O}}(SN(q_{\mathsf{G}}^{\mathsf{on}})^2/p)$ and/or $\mathcal{O}\left(q_{\mathsf{H}}^{\mathsf{on}}/2^{k_1}\right)$.
- A preprocessing attacker can simply solve one of the discrete-log challenges with probability at least $\Omega(S(q_{\mathsf{G}}^{\mathsf{on}})^2/p)$ which would recover a secret key and make it trivial to forge a signature.
- Any attacker who makes $q_{\mathsf{H}}^{\mathsf{on}} \ge q_{\mathsf{S}}^{\mathsf{on}}$ queries to the random oracle can fix an arbitrary message m and pick random numbers $r_1,\ldots,r_{q_{\mathsf{H}}^{\mathsf{on}}}$ hoping that $\mathsf{H}(\mathsf{pk}_1\|\tau(r_j)\|m) = 0$ for some $j \le q_{\mathsf{H}}^{\mathsf{on}}$. In this case, $(r_j, 0)$ is a valid forged signature for m under public key pk_1. Thus, the attacker can succeed with probability $\approx q_{\mathsf{H}}^{\mathsf{on}}/2^{k_1}$. ◁

The full proof of Theorem 8 can be found in Appendix C.3. The key idea is that we can repeat the essentially same reduction from Section 4, i.e., we can build a bridge-finding game attacker $(\mathcal{A}_{\mathsf{bridge}}^{\mathsf{pre}}, \mathcal{A}_{\mathsf{bridge}}^{\mathsf{on}})$ *with preprocessing* from the signature forgery attacker $(\mathcal{A}_{\mathsf{sig}}^{\mathsf{pre}}, \mathcal{A}_{\mathsf{sig}}^{\mathsf{on}})$ *with preprocessing*, except that when we program a random oracle, we define an additional bad event that we program a random oracle at a point the attacker has already queried the point during the preprocessing phase. We observe that such probability is negligibly small. As long as the failure event does not occur we can program the random oracle and the attacker will not notice the difference.

Instantiating Key-Prefixed Short Schnorr Signatures. We would like to have the success probability in Theorem 8 bounded by $\mathcal{O}\left(q/2^k\right)$ for any $q \le 2^k$, where $q = q_{\mathsf{G}}^{\mathsf{on}} + q_{\mathsf{H}}^{\mathsf{on}} + q_{\mathsf{S}}^{\mathsf{on}}$ is the total number of *online* queries made by a preprocessing attacker. To achieve k bits of multi-user security for key-prefixed short Schnorr signatures with preprocessing, we can fix p such that $p \approx 2^{2k}SN\log p$, and set the length of our hash output to be $k_1 = k$. With these parameters, Theorem 8 tells us that a preprocessing attacker wins the signature forgery game with probability at most $\varepsilon = \mathcal{O}\left((q_{\mathsf{H}}^{\mathsf{on}} + q_{\mathsf{G}}^{\mathsf{on}})/2^k\right)$. The length of the signatures we obtain will be $k + \log p = 3k + \log N + \log S + \log\log p$.

As a concrete example, if $N \le 2^{k/4}$ and $S \le 2^{k/2}$, then we obtain signatures of length $\approx 3.75k + \log 2.75k$. If we want $k \ge 128$ bits of security, then the

assumption that $N < 2^{k/4}$ seems quite reasonable, since $2^{32}(\approx 4.3$ billion) is over half of the current global population, and 2^{64} bits exceeds the storage capacity of Facebook's data warehouse[4]. As a second example, if we take $S \leq 2^{80}$ as an upper bound on the storage capacity of any nation state and $N \approx 2^{40}$, then we obtain signatures of length $\approx 3k + 120 + \log(2k + 120)$.

6 Multi-User Security of Other Fiat-Shamir Signatures

In this section, we show that our techniques from Section 4 and Section 5 apply to other Fiat-Shamir-based signature schemes. We apply our reductions to analyze the multi-user security of the full-domain hash variant of Chaum-Pedersen signatures [CP93], and (short) Katz-Wang signatures [KW03], with and without preprocessing. In practice, the full-domain hash variant of Chaum-Pedersen would be used to ensure that our signature scheme supports the message space $m \in \{0,1\}^*$ instead of requiring that m is a group element. We begin by introducing regular Chaum-Pedersen signatures in the next paragraph before describing the full-domain hash variant (Chaum-Pedersen-FDH) that we analyze.

Security Analysis of Chaum-Pedersen-FDH Signatures. The Chaum-Pedersen signature scheme [CP93] is obtained by applying the Fiat-Shamir transform [FS87] to the Chaum-Pedersen identification scheme and works as follows.

- Given a cyclic group $G = \langle g \rangle$ of prime order p, the key generation algorithm picks $\mathsf{sk} \leftarrow_{\$} \mathbb{Z}_p$ and sets $\mathsf{pk} = g^{\mathsf{sk}}$.
- To sign a message $m \in G$ with the secret key sk, we sample $r \leftarrow_{\$} \mathbb{Z}_p$ and compute $y = m^{\mathsf{sk}}$, $a = g^r$, $b = m^r$, and $e = \mathsf{H}(m\|y\|a\|b)$. Finally, we output a signature $\sigma = (y, a, b, s)$, where $s := r + \mathsf{sk} \cdot e \mod p$.
- The verification algorithm takes as inputs a signature $\sigma' = (y', a', b', s')$ and computes $e' = \mathsf{H}(m\|y'\|a'\|b')$, $A = g^{s'}$, $B = a' g^{\mathsf{sk} \cdot e'}$, $C = m^{s'}$ and $D = b' y'^{e'}$. Finally, we verify that $(A = B)$ and $(C = D)$ before accepting the signature.

The *full-domain hash variant* of Chaum-Pedersen signature, say *Chaum-Pedersen-FDH signature*, is obtained by hashing a message m into a group element so that we can perform generic group operations when signing the message. That is, in the generic group model, we compute $h = \mathsf{H}'(\mathsf{pk}\|m) := \mathtt{Pow}(\mathfrak{g}, \mathsf{H}(\mathsf{pk}\|m))$ and compute $\mathfrak{y} = \mathtt{Pow}(h, \mathsf{sk})$ and $\mathfrak{b} = \mathtt{Pow}(h, r)$ (which corresponds to $y = h^{\mathsf{sk}}$ and $b = h^r$ when instantiated with a cyclic group $G = \langle g \rangle$) during the signing procedure. Note that key-prefixing is necessary as otherwise an attacker can always forge a signature for a message m, e.g., simply find $m \neq m'$ such that $\mathsf{H}(m) = \mathsf{H}(m')$. The full description for each of these algorithms can be found in Figure 4 in Appendix A.

Our reduction in Section 4 naturally extends to Chaum-Pedersen-FDH signature scheme by using signing oracle in Figure 3. The signing oracle is able go

[4] See the link: https://engineering.fb.com/2014/04/10/core-data/scaling-the-facebook-data-warehouse-to-300-pb/ (Retrieved 2/20/2021)

generate valid signatures *without* the secret key by programming the random oracle. This allows us to prove Theorem 9. We remark that a Chaum-Pedersen-FDH signature with k bits of security has length $8k$ — each group element requires $2k$ bits to encode since $p \approx 2^{2k}$. Note that reducing the length of the hash output does not have any effect on Chaum-Pedersen-FDH signature length. Thus, we assume that H is a random oracle with $2k$-bit outputs. The proof of Theorem 9 can be found in Appendix C.4.

Theorem 9. *The Chaum-Pedersen-FDH signature scheme is* $\left(N, q_{\mathsf{H}}, q_{\mathfrak{G}}, q_{\mathfrak{S}}, \mathcal{O}\left(\frac{q+N}{2^k}\right)\right)$-$\mathsf{MU\text{-}UF\text{-}CMA}$ *secure under the generic group model of prime order* $p \approx 2^{2k}$ *and the programmable random oracle model, where* q *denotes the total number of queries made by an adversary.*

We can also show that the *key-prefixed* Chaum-Pedersen-FDH signature scheme is secure against proprocessing attacks. That is, we apply key-prefixing when computing e, i.e. $e \leftarrow \mathsf{H}(\mathsf{pk}\|h\|\mathfrak{y}\|\mathfrak{a}\|\mathfrak{b})$ during the signing procedure and $e' \leftarrow \mathsf{H}(\mathsf{pk}\|h\|\mathfrak{y}'\|\mathfrak{a}'\|\mathfrak{b}')$ during the verification (see Figure 4). During the online phase we can request a signature σ for m and output $\sigma' = \sigma$ as our forgery for m'. We defer the full proof of Theorem 10 to Appendix C.4.

Theorem 10. *Let* $\Pi = (\mathsf{Kg}, \mathsf{Sign}, \mathsf{Vfy})$ *be a key-prefixed Chaum-Pedersen-FDH signature scheme and* $p > 2^{2k}$ *be a prime number. Let* $N \in \mathbb{N}$ *be a parameter and* $(\mathcal{A}^{\mathsf{pre}}_{\mathsf{sig}}, \mathcal{A}^{\mathsf{on}}_{\mathsf{sig}})$ *be a pair of generic algorithms with an encoding map* $\tau : \mathbb{Z}_p \to \mathbb{G}$ *such that* $\mathcal{A}^{\mathsf{pre}}_{\mathsf{sig}}$ *makes at most* $q^{\mathsf{pre}}_{\mathsf{H}} < 2^{3k}$ *queries to the random oracle* $\mathsf{H} : \{0,1\}^* \to \{0,1\}^{2k}$ *and outputs an* S*-bit hint* $\mathsf{str}_{\tau,\mathsf{H}}$*, and* $\mathcal{A}^{\mathsf{on}}_{\mathsf{sig}}$ *makes at most* $q^{\mathsf{on}}_{\mathfrak{G}} := q^{\mathsf{on}}_{\mathfrak{G}}(k)$ *queries to the generic group oracles and at most* $q^{\mathsf{on}}_{\mathsf{H}}$ *queries to the random oracle. Then* $\Pr\left[\mathsf{SigForge}^{\tau,N}_{\mathcal{A}^{\mathsf{on}}_{\mathsf{sig}},\mathsf{str}_{\tau,\mathsf{H}},\Pi}(k) = 1\right] \le \varepsilon$*, with*

$$\varepsilon = \widetilde{\mathcal{O}}\left(\frac{SN(q^{\mathsf{on}}_{\mathfrak{G}}+N)(q^{\mathsf{on}}_{\mathfrak{G}}+2N)}{p}\right) + \frac{Nq^{\mathsf{pre}}_{\mathsf{H}}q^{\mathsf{on}}_{\mathsf{S}}}{p^2} + \frac{q^{\mathsf{on}}_{\mathsf{S}}(q^{\mathsf{on}}_{\mathsf{S}}+q^{\mathsf{on}}_{\mathsf{H}})}{p} + \frac{4(q^{\mathsf{on}}_{\mathsf{H}}+\widetilde{q}^2_{\mathsf{on}}+1)}{2^{2k}} + \frac{3N^2(S+2k)}{2p},$$

where $q^{\mathsf{on}}_{\mathsf{S}}$ *denotes the number of queries to the signing oracle,* $\widetilde{q}_{\mathsf{on}} = q^{\mathsf{on}}_{\mathsf{H}} + 2q^{\mathsf{on}}_{\mathsf{S}}$*, and the randomness is taken over the selection of* τ *and the random coins of* $\mathcal{A}^{\mathsf{on}}_{\mathsf{sig}}$ *(the hint* $\mathsf{str}_{\tau,\mathsf{H}} = \mathcal{A}^{\mathsf{pre},\mathsf{GD}}_{\mathsf{sig}}(\mathfrak{g})$ *is selected independently of the random coins used by the challenger).*

Applying Theorem 10, we can fix p such that $p \approx 2^{2k}SN\log p$ to achieve k bits of multi-user security. The final signature size would be $\approx 8k + 4\log S + 4\log N + 4\log(2k + \log SN)$.

Security Analysis of Katz-Wang Signatures. The Katz-Wang signature scheme [KW03] is a double generator version of Schnorr signature scheme. In the generic group model on a cyclic group G of prime order p, we have two generators $p_1, p_2 \in \mathbb{Z}_p$ so that we can associate with g^{p_1} and g^{p_2} to the generators of the group G. Here, the message space for m is arbitrary, i.e., $m \in \{0,1\}^*$.

Given our encoding $\tau : \mathbb{Z}_p \to \mathbb{G}$ and $\mathfrak{g} = \tau(1)$, our key generation algorithm picks $\mathsf{sk} \leftarrow_{\$} \mathbb{Z}_p$ and sets $\mathsf{pk} = (p_1, p_2, \mathfrak{h}_1, \mathfrak{h}_2)$, where $\mathfrak{h}_i = \mathtt{Pow}(\tau(p_i), \mathsf{sk})$ for $i = 1, 2$.

To sign a message $m \in \{0,1\}^*$ with the secret key sk, we sample $r \leftarrow\!\!\$\, \mathbb{Z}_p$, and compute $\mathfrak{a}_i = \mathtt{Pow}(\tau(p_i), r)$ for $i = 1, 2$, $e = \mathsf{H}(\mathsf{pk} \| \mathfrak{a}_1 \| \mathfrak{a}_2 \| m)$, and $s = r + \mathsf{sk} \cdot e \mod p$. Finally, we output $\sigma = (s, e)$. The verification algorithm takes as inputs a signature $\sigma' = (s', e')$, $\mathsf{pk} = (p_1, p_2, \mathfrak{h}_1, \mathfrak{h}_2)$ and the message m, and compute $\mathfrak{a}'_i = \mathtt{Mult}(\mathtt{Pow}(\tau(p_i), s'), \mathtt{Pow}(\mathtt{Inv}(\mathfrak{h}_i), e'))$ for $i = 1, 2$. Finally, we verify that $e' = \mathsf{H}(\mathsf{pk} \| \mathfrak{a}'_1 \| \mathfrak{a}'_2 \| m)$ before accepting the signature. The pseudocode for each of these algorithms can be found in Figure 5 in Appendix A.

$\mathsf{Sign}_j(m_i)$ without secret key x_j, $j \in [N]$ (Chaum-Pedersen-FDH)

1 : Pick s_i and $e_i \in \mathbb{Z}_p$ randomly
2 : Compute $\mathfrak{s}_i = \mathtt{Pow}(\mathfrak{g}, s_i)$ and $h_{ij} = \mathtt{Pow}(\mathfrak{g}, \mathsf{H}(\mathsf{pk}_j \| m_i))$
3 : Compute $\mathfrak{y}_i = \mathtt{Pow}(\mathsf{pk}_j, \mathsf{H}(\mathsf{pk}_j \| m_i))$
4 : Compute $\mathfrak{a}_i = \mathtt{Mult}(\mathfrak{s}_i, \mathtt{Pow}(\mathtt{Inv}(\mathsf{pk}_j), e_i))$
5 : Compute $\mathfrak{b}_i = \mathtt{Mult}(\mathtt{Pow}(h_{ij}, s_i), \mathtt{Pow}(\mathtt{Inv}(\mathfrak{y}_i, e_i)))$
6 : **if** $\mathsf{H}(h_{ij} \| \mathfrak{y}_i \| \mathfrak{a}_i \| \mathfrak{b}_i) \in$ prior query **then**
7 : **return** $\perp$
8 : **else** Program $\mathsf{H}(h_{ij} \| \mathfrak{y}_i \| \mathfrak{a}_i \| \mathfrak{b}_i) := e_i$
9 : **return** $\sigma_i = (\mathfrak{y}_i, \mathfrak{a}_i, \mathfrak{b}_i, s_i)$

$\mathsf{Sign}_j(m_i)$ without secret key x_j, $j \in [N]$ (Katz-Wang)

1 : Pick $s_i, e_i \in \mathbb{Z}_p$ randomly
2 : Compute $\mathfrak{a}_{1,i} = \mathtt{Mult}(\mathtt{Pow}(\tau(p_1), s_i), \mathtt{Pow}(\mathtt{Inv}(\mathtt{Pow}(\tau(x_j), p_1)), e_i))$
3 : Compute $\mathfrak{a}_{2,i} = \mathtt{Mult}(\mathtt{Pow}(\tau(p_2), s_i), \mathtt{Pow}(\mathtt{Inv}(\mathtt{Pow}(\tau(x_j), p_2)), e_i))$
4 : **if** $\mathsf{H}(\mathsf{pk}_j \| \mathfrak{a}_{1,i} \| \mathfrak{a}_{2,i} \| m_i) \in$ prior query **then**
5 : **return** $\perp$
6 : **else** Program $\mathsf{H}(\mathsf{pk}_j \| \mathfrak{a}_{1,i} \| \mathfrak{a}_{2,i} \| m_i) := e_i$
7 : **return** $\sigma_i = (s_i, e_i)$

Figure 3. The signing oracle without secret key in the Chaum-Pedersen-FDH scheme (top) and the Katz-Wang scheme (bottom). Note that $\mathsf{pk}_j = \tau(x_j)$ is public in both schemes while the signing oracle has no information about x_j. We further remark that in the key-prefixed Chaum-Pedersen-FDH scheme, the only difference is to do a key-prefixing $\mathsf{pk}_j = \tau(x_j)$ to the input of the random oracle (line 6 and 8).

We remark that the length of a regular Katz-Wang signature is $4k$ bits when $p \approx 2^{2k}$. 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, while we use the signing oracle in Figure 3 without the secret key.

We observe that Katz-Wang signature is already key-prefixed. The security bounds in Theorem 11 and Theorem 12 are equivalent to our bounds for short Schnorr signatures with and without pre-processing. Thus, we obtain $3k$ (resp. $3k + \log N + \log S + \log(2k + \log NS)$)-bit signatures with k bits of security in the multi-user setting without preprocessing (resp. with preprocessing). As before in the preprocessing setting we select our prime number $p \approx 2^{2k} NS \log(2k + \log NS)$

and we fix the length of the hash output to be $k_1 = k$. We defer the full proof of Theorem 11 and Theorem 12 to Appendix C.4.

Theorem 11. *The (short) Katz-Wang signature scheme is* $\left(N, q_{\mathsf{H}}, q_{\mathsf{G}}, q_{\mathsf{S}}, \mathcal{O}\left(\frac{q+N}{2^k}\right)\right)$-MU-UF-CMA *secure under the generic group model of prime order* $p \approx 2^{2k}$ *and the programmable random oracle model, where* q *denotes the total number of queries made by an adversary.*

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 $\varepsilon' \leq t'(4\varepsilon/t + q_{\mathsf{S}}/p + 1/2^k)$. While their result is similar to Theorem 11, our bounds apply to (short) Katz-Wang signatures, with and without preprocessing.

Theorem 12. *Let* $\Pi = (\mathsf{Kg}, \mathsf{Sign}, \mathsf{Vfy})$ *be a Katz-Wang signature scheme and* $p > 2^{2k}$ *be a prime number. Let* $N \in \mathbb{N}$ *be a parameter and* $(\mathcal{A}^{\mathsf{pre}}_{\mathsf{sig}}, \mathcal{A}^{\mathsf{on}}_{\mathsf{sig}})$ *be a pair of generic algorithms with an encoding map* $\tau : \mathbb{Z}_p \to \mathbb{G}$ *such that* $\mathcal{A}^{\mathsf{pre}}_{\mathsf{sig}}$ *makes at most* $q^{\mathsf{pre}}_{\mathsf{H}} < 2^{3k}$ *queries to the random oracle at most* $q^{\mathsf{pre}}_{\mathsf{H}} < 2^{3k}$ *queries to the random oracle* $\mathsf{H} : \{0,1\}^* \to \{0,1\}^{k_1}$ *and outputs an* S*-bit hint* $\mathsf{str}_{\tau,\mathsf{H}}$*, and* $\mathcal{A}^{\mathsf{on}}_{\mathsf{sig}}$ *makes at most* $q^{\mathsf{on}}_{\mathsf{G}} := q^{\mathsf{on}}_{\mathsf{G}}(k)$ *queries to the generic group oracles and at most* $q^{\mathsf{on}}_{\mathsf{H}}$ *queries to the random oracle. Then* $\Pr\left[\mathsf{SigForge}^{\tau,N}_{\mathcal{A}^{\mathsf{on}}_{\mathsf{sig}},\mathsf{str}_{\tau,\mathsf{H}},\Pi}(k) = 1\right] \leq \varepsilon$*, with*

$$\varepsilon = \widetilde{\mathcal{O}}\left(\frac{SN(q^{\mathsf{on}}_{\mathsf{G}}+N)(q^{\mathsf{on}}_{\mathsf{G}}+2N)}{p}\right) + \frac{Nq^{\mathsf{pre}}_{\mathsf{H}}q^{\mathsf{on}}_{\mathsf{S}}}{p^2} + \frac{q^{\mathsf{on}}_{\mathsf{S}}(q^{\mathsf{on}}_{\mathsf{S}}+q^{\mathsf{on}}_{\mathsf{H}})}{p} + \frac{4(q^{\mathsf{on}}_{\mathsf{H}}+1)}{2^{k_1}} + \frac{N^2(S+k_1)}{p},$$

where $q^{\mathsf{on}}_{\mathsf{S}}$ *denotes the number of queries to the signing oracle and the randomness is taken over the selection of* τ *and the random coins of* $\mathcal{A}^{\mathsf{on}}_{\mathsf{sig}}$ *(the hint* $\mathsf{str}_{\tau,\mathsf{H}} = \mathcal{A}^{\mathsf{pre},\mathsf{GO}}_{\mathsf{sig}}(\mathfrak{g})$ *is selected independently of the random coins used by the challenger).*

Acknowledgements

Jeremiah Blocki was supported in part by the National Science Foundation under Awards CNS #1704587, CNS #1755708 and CCF #1910659. Seunghoon Lee was supported in part by NSF Award CNS #1755708 and by the Center for Science of Information (CSoI). The opinions in this paper are those of the authors and do not necessarily reflect the position of the National Science Foundation.

References

AB09. Sanjeev Arora and Boaz Barak. Computational complexity - a modern approach, 2009.

AB20. Handan Kilinc Alper and Jeffrey Burdges. Two-round trip schnorr multi-signatures via delinearized witnesses. Cryptology ePrint Archive, Report 2020/1245, 2020. https://eprint.iacr.org/2020/1245.

AFK87. Martín Abadi, Joan Feigenbaum, and Joe Kilian. On hiding information from an oracle (extended abstract). In Alfred Aho, editor, *19th ACM STOC*, pages 195–203. ACM Press, May 1987.

BCJ08. Ali Bagherzandi, Jung Hee Cheon, and Stanislaw Jarecki. Multisignatures secure under the discrete logarithm assumption and a generalized forking lemma. In Peng Ning, Paul F. Syverson, and Somesh Jha, editors, *ACM CCS 2008*, pages 449–458. ACM Press, October 2008.

BD20. Mihir Bellare and Wei Dai. The multi-base discrete logarithm problem: Tight reductions and non-rewinding proofs for Schnorr identification and signatures. In Karthikeyan Bhargavan, Elisabeth Oswald, and Manoj Prabhakaran, editors, *INDOCRYPT 2020*, volume 12578 of *LNCS*, pages 529–552. Springer, Heidelberg, December 2020.

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

BHK+19. Jeremiah Blocki, Benjamin Harsha, Siteng Kang, Seunghoon Lee, Lu Xing, and Samson Zhou. Data-independent memory hard functions: New attacks and stronger constructions. In Alexandra Boldyreva and Daniele Micciancio, editors, *CRYPTO 2019, Part II*, volume 11693 of *LNCS*, pages 573–607. Springer, Heidelberg, August 2019.

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.

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

BN06. Mihir Bellare and Gregory Neven. Multi-signatures in the plain public-key model and a general forking lemma. In Ari Juels, Rebecca N. Wright, and Sabrina De Capitani di Vimercati, editors, *ACM CCS 2006*, pages 390–399. ACM Press, October / November 2006.

BNPS03. Mihir Bellare, Chanathip Namprempre, David Pointcheval, and Michael Semanko. The one-more-RSA-inversion problems and the security of Chaum's blind signature scheme. *Journal of Cryptology*, 16(3):185–215, June 2003.

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.

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.

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.

DEF+19. Manu Drijvers, Kasra Edalatnejad, Bryan Ford, Eike Kiltz, Julian Loss, Gregory Neven, and Igors Stepanovs. On the security of two-round multi-signatures. In *2019 IEEE Symposium on Security and Privacy*, pages 1084–1101. IEEE Computer Society Press, May 2019.

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.

DKW11. Stefan Dziembowski, Tomasz Kazana, and Daniel Wichs. One-time computable self-erasing functions. In Yuval Ishai, editor, *TCC 2011*, volume 6597 of *LNCS*, pages 125–143. Springer, Heidelberg, March 2011.

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.

DTT10. Anindya De, Luca Trevisan, and Madhur Tulsiani. Time space tradeoffs for attacks against one-way functions and PRGs. In Tal Rabin, editor, *CRYPTO 2010*, volume 6223 of *LNCS*, pages 649–665. Springer, Heidelberg, August 2010.

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.

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.

FS87. Amos Fiat and Adi Shamir. How to prove yourself: Practical solutions to identification and signature problems. In Andrew M. Odlyzko, editor, *CRYPTO'86*, volume 263 of *LNCS*, pages 186–194. Springer, Heidelberg, August 1987.

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

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.

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.

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.

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.

Hao17. Feng Hao. Schnorr Non-interactive Zero-Knowledge Proof. RFC 8235, September 2017.

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.

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.

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

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

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.

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.

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.

MPSW19. Gregory Maxwell, Andrew Poelstra, Yannick Seurin, and Pieter Wuille. Simple schnorr multi-signatures with applications to bitcoin. *Designs, Codes and Cryptography*, 87(9):2139–2164, 2019.

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.

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

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

NRS20. Jonas Nick, Tim Ruffing, and Yannick Seurin. MuSig2: Simple two-round schnorr multi-signatures. Cryptology ePrint Archive, Report 2020/1261, 2020. https://eprint.iacr.org/2020/1261.

NRSW20. Jonas Nick, Tim Ruffing, Yannick Seurin, and Pieter Wuille. MuSig-DN: Schnorr multi-signatures with verifiably deterministic nonces. Cryptology ePrint Archive, Report 2020/1057, 2020. https://eprint.iacr.org/2020/1057.

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.

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

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.

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.

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.

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.

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.

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.

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.

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

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.

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.

A Full Description of Other Fiat-Shamir-Based Signatures

A.1 (Key-Prefixed) Chaum-Pedersen-FDH Signature Scheme

$\mathsf{Kg}(1^k)$:	$\mathsf{Sign}(\mathsf{sk}, m)$:	$\mathsf{Vfy}(\mathsf{pk}, m, \sigma)$:
1 : $\mathsf{sk} \leftarrow\!\!\$\ \mathbb{Z}_p$	1 : $r \leftarrow\!\!\$\ \mathbb{Z}_p$	1 : Parse $\sigma = (\mathfrak{y}', \mathfrak{a}', \mathfrak{b}', s')$
2 : $\mathsf{pk} \leftarrow \mathtt{Pow}(\mathfrak{g}, \mathsf{sk})$	2 : $h \leftarrow \mathtt{Pow}(\mathfrak{g}, \mathsf{H}(\mathsf{pk}\|m))$	2 : $h \leftarrow \mathtt{Pow}(\mathfrak{g}, \mathsf{H}(\mathsf{pk}\|m))$
3 : **return** $(\mathsf{pk}, \mathsf{sk})$	3 : $\mathfrak{y} \leftarrow \mathtt{Pow}(h, \mathsf{sk})$	3 : $e' \leftarrow \mathsf{H}(h\|\mathfrak{y}'\|\mathfrak{a}'\|\mathfrak{b}')$
	4 : $\mathfrak{a} \leftarrow \mathtt{Pow}(\mathfrak{g}, r)$	4 : $A \leftarrow \mathtt{Pow}(\mathfrak{g}, s')$
	5 : $\mathfrak{b} \leftarrow \mathtt{Pow}(h, r)$	5 : $B \leftarrow \mathtt{Mult}(\mathfrak{a}', \mathtt{Pow}(\mathsf{pk}, e'))$
	6 : $e \leftarrow \mathsf{H}(h\|\mathfrak{y}\|\mathfrak{a}\|\mathfrak{b})$	6 : $C \leftarrow \mathtt{Pow}(h, s')$
	7 : $s \leftarrow r + \mathsf{sk} \cdot e \mod p$	7 : $D \leftarrow \mathtt{Mult}(\mathfrak{b}', \mathtt{Pow}(\mathfrak{y}', e'))$
	8 : **return** $\sigma = (\mathfrak{y}, \mathfrak{a}, \mathfrak{b}, s)$	8 : **if** $(A = B) \wedge (C = D)$ **then**
		9 : **return** 1
		10 : **else return** 0

Figure 4. The Chaum-Pedersen-FDH signature scheme in the generic group model. Note that in the *key-prefixed* Chaum-Pedersen-FDH signature scheme, we include the public key in the input to the random oracle, i.e., $e \leftarrow \mathsf{H}(\mathsf{pk}\|h\|\mathfrak{y}\|\mathfrak{a}\|\mathfrak{b})$ on line 6 in $\mathsf{Sign}(\mathsf{sk}, m)$ and $e' \leftarrow \mathsf{H}(\mathsf{pk}\|h\|\mathfrak{y}'\|\mathfrak{a}'\|\mathfrak{b}')$ on line 3 in $\mathsf{Vfy}(\mathsf{pk}, m, \sigma)$.

A.2 Katz-Wang Signature Scheme

$\mathsf{Kg}(1^k)$:	$\mathsf{Sign}(\mathsf{sk}, m)$:	$\mathsf{Vfy}(\mathsf{pk}, m, \sigma)$:
1 : $\mathsf{sk} \leftarrow\!\!\$\ \mathbb{Z}_p$	1 : $r \leftarrow\!\!\$\ \mathbb{Z}_p$	1 : Parse $\sigma = (s', e')$
2 : $\mathfrak{h}_1 \leftarrow \mathtt{Pow}(\tau(p_1), \mathsf{sk})$	2 : $\mathfrak{a}_1 \leftarrow \mathtt{Pow}(\tau(p_1), r)$	2 : Parse $\mathsf{pk} = (p_1, p_2, \mathfrak{h}_1, \mathfrak{h}_2)$
3 : $\mathfrak{h}_2 \leftarrow \mathtt{Pow}(\tau(p_2), \mathsf{sk})$	3 : $\mathfrak{a}_2 \leftarrow \mathtt{Pow}(\tau(p_2), r)$	3 : $\mathfrak{a}'_1 \leftarrow \mathtt{Mult}(\mathtt{Pow}(\tau(p_1), s'), \mathtt{Pow}(\mathtt{Inv}(\mathfrak{h}_1), e'))$
4 : $\mathsf{pk} \leftarrow (p_1, p_2, \mathfrak{h}_1, \mathfrak{h}_2)$	4 : $e \leftarrow \mathsf{H}(\mathsf{pk}\|\mathfrak{a}_1\|\mathfrak{a}_2\|m)$	4 : $\mathfrak{a}'_2 \leftarrow \mathtt{Mult}(\mathtt{Pow}(\tau(p_2), s'), \mathtt{Pow}(\mathtt{Inv}(\mathfrak{h}_2), e'))$
5 : **return** $(\mathsf{pk}, \mathsf{sk})$	5 : $s \leftarrow r + \mathsf{sk} \cdot e \mod p$	5 : **if** $e' = \mathsf{H}(\mathsf{pk}\|\mathfrak{a}'_1\|\mathfrak{a}'_2\|m)$ **then**
	6 : **return** $\sigma = (s, e)$	6 : **return** 1
		7 : **else return** 0

Figure 5. The Katz-Wang Signature Scheme in the generic group model.

B Comparing Different Generic Group Models

In our analysis we show that short Schnorr signatures provide k bits of security in Shoup's 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 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 Shoup's generic group model [Sho97] it does appear to be not possible to describe the attack in other variations of the generic group model.

B.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 i^{th} 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 $(a_1, \cdots, a_{i-1}) \in \mathbb{Z}_p^{i-1}$ 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* generic algorithm *(of Schnorr and Jakobsson) 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 $(a_1, \cdots, a_{i-1}) \in \mathbb{Z}_p^{i-1}$ 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 4. 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 $G_0 = \{h \ : \tau(h) = 0 \mod \sqrt{p}\}$ then it is easy to sample elements in G_0 in Shoup's model, i.e., pick a random x such that $x = 0 \mod \sqrt{p}$ 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 Schnorr and Jakobsson's 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 $x, y \in \mathbb{Z}_p$. 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 $st^2 = \Omega(p\epsilon^2)$. 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 $st^2 = \tilde{\Omega}(p\epsilon^2)$.

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 $\mathsf{H}_{\mathsf{m}} : \mathbb{G}^2 \to \{1, \ldots, t\}$ and $\mathsf{H}_{\mathsf{c}} : \mathbb{G}^2 \to \{1, \ldots, s\}$ 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 $f : \tau(G)^2 \to \mathbb{Z}_p$, 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 $\mathcal{Y} = \{(\tau(g^x), \tau(g^{(x^2)})) : x \in \mathbb{Z}_p\} \subset \tau(G)^2$ 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 $\Omega\left(p/(3t^2)\right)$ marked nodes $\mathcal{T} \subseteq \mathcal{Y}$. Then for each color $c \in \{1, \ldots, s\}$ the preprocessing algorithm computes the advice string $w_c \in \{0,1\}^{\log p}$ such that

$$w_c = \underset{w \in \{0,1\}^{\log p}}{\arg\max} \sum_{\substack{(\mathfrak{m}, c_{\mathfrak{m}}) \in \mathcal{T}: \\ c_{\mathfrak{m}} = c}} H(w, \mathfrak{m}),$$

where $H : \{0,1\}^{\log p} \times \mathbb{G}^2 \to \{0,1\}$ is a random function. Please see [CK18] for the complete discussion.

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 p^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 $\mathsf{H}_{\mathsf{m}} : \mathbb{G}^2 \to \{1, \ldots, t\}$. 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 w_c using the hash function $H : \{0,1\}^{\log p} \times \mathbb{G}^2 \to \{0,1\}$.

B.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 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., $(y, C_y) \in \mathbb{Z}_p \times \mathbb{N}$ 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 $x_i \in \mathbb{Z}_p$ 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 $z = a + b \mod p$.
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 $\alpha_{C_a,C_b} \in \mathbb{Z}_p^*$ such that $\alpha_{C_a,C_b}[j] = 0$ 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 5. Another (minor) limitation is that there is no $\texttt{Inv}$ oracle in the Kiltz et al. [KMP16]. Thus, computing h^{-1} requires us to compute h^{p-1} using $\mathcal{O}(\log p)$ queries to the generic group model. Thus, to simulate an attacker who makes t queries to the $\texttt{Mult}, \texttt{Inv}$ oracles we might require *up to* $t \log p$ queries if we only have the $\texttt{Mult}$ oracle. Such a reduction increases running time by a factor of $\mathcal{O}(\log p)$ so we lose a factor of $\log p$ in the security reduction if their is an efficient algorithm to compute h^{-1} directly. ◁

C Missing Proofs

C.1 Missing Proofs from Section 3

Reminder of Lemma 1. *The probability the attacker making at most $q_{\texttt{G}}$ generic group oracle queries wins the generic discrete-log game $\mathsf{DLogChal}^{\tau}_{\mathcal{A}}(k)$ (even with access to the restricted $\texttt{DLog}$ oracle) is at most*

$$\Pr\left[\mathsf{DLogChal}^{\tau}_{\mathcal{A}}(k) = 1\right] \leq \frac{6q_{\texttt{G}}(q_{\texttt{G}}+1)+12}{4p-(3q_{\texttt{G}}+2)^2},$$

in the generic group model of prime order p, where the randomness is taken over the selection of τ, the challenge x, as well as any random coins of $\mathcal{A}$.

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 $\texttt{DLog}(\tau(y))$ before making the query (We say WLOG because the attacker can always ignore the result.) Thus, we will build up the known sets $\mathcal{K}$ (elements for which we *know* the discrete-log solution and it initially contains $(\tau(1), 0, 1)$) and the partially known set $\mathcal{PK}_x$ (elements for which the discrete-log solution is *partially known* and it initially contains $(\tau(x), 1, 0)$) which are the subsets of the list $\mathcal{L}$. Then we have the following observations for those sets:

- Initially, $\mathcal{K}$ contains $(\tau(1), 0, 1)$ and $\mathcal{PK}_x$ contains $(\tau(x), 1, 0)$.
- 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) $\tau(y) = \tau(r)$ for some known $r \in \mathbb{Z}_p$ and (2) $\tau(y) = \tau(ax+b)$ for known $a, b \in \mathbb{Z}_p$ 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 $p - |\mathcal{K}| \times |\mathcal{PK}_x|$. To see this note that each pair of distinct elements $\tau(ax+b)$ in $\mathcal{PK}_x$ and $\tau(r)$ in $\mathcal{K}$ eliminates at most one possible value of x, i.e., since $\tau(ax+b)$ and $\tau(r)$ are distinct we have $x \neq (r-b)a^{-1}$.

We further note that $|\mathcal{K}| + |\mathcal{PK}_x| \leq 3q_{\mathtt{G}} + 2$ since each query to the generic group oracles adds at most 3 elements to $\mathcal{K} \cup \mathcal{PK}_x$ — equality holds when both inputs to $\mathtt{Mult}(\cdot,\cdot)$ are fresh. The probability the attacker guesses x correctly is at most

$$\begin{aligned}\Pr\left[\mathsf{DLogChal}^{\tau}_{\mathcal{A}}(k) = 1 \;\middle|\; \overline{\mathsf{BRIDGE}}\right] &\leq \frac{1}{p - |\mathcal{K}| \times |\mathcal{PK}_x|} \\ &< \frac{1}{p - (3q_{\mathtt{G}}+2)^2/4}\ .\end{aligned}$$

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 $3q_{\mathtt{G}} + 2$ and the AM-GM inequality.

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 $p - (3q_{\mathtt{G}}+2)^2/4$. 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 $\frac{1}{p-(3q_{\mathtt{G}}+2)^2/4}$. 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

$$\Pr\left[B_i \;\middle|\; \overline{\mathsf{BRIDGE}}_{<i}\right] \leq \frac{3i+2}{p - (3q_{\mathtt{G}}+2)^2/4}\ .$$

Therefore, the probability of BRIDGE is upper bounded by

$$\begin{aligned}\Pr[\mathsf{BRIDGE}] &= \sum_{i \leq q_{\mathtt{G}}} \Pr\left[B_i \;\middle|\; \overline{\mathsf{BRIDGE}}_{<i}\right] \\ &\leq \sum_{i \leq q_{\mathtt{G}}} \frac{3i+2}{p - (3q_{\mathtt{G}}+2)^2/4} \\ &\leq \frac{3(q_{\mathtt{G}}+1)q_{\mathtt{G}}/2 + 2}{p - (3q_{\mathtt{G}}+2)^2/4}\ .\end{aligned}$$

Thus, the probability the attacker succeeds is upper bounded by[5]

$$\Pr\left[\mathsf{DLogChal}^{\tau}_{\mathcal{A}}(k)=1\right] \leq \Pr\left[\mathsf{BRIDGE}\right] + \Pr\left[\mathsf{DLogChal}^{\tau}_{\mathcal{A}}(k)=1 \,\middle|\, \overline{\mathsf{BRIDGE}}\right] \leq \frac{6q_{\mathtt{G}}(q_{\mathtt{G}}+1)+12}{4p-(3q_{\mathtt{G}}+2)^2},$$

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

Reminder of Theorem 4. *The* short *Schnorr signature scheme* $\Pi_{\mathsf{short}} = (\mathsf{Kg}, \mathsf{Sign}, \mathsf{Vfy})$ *of length* $3k$ *is* $(q_{\mathsf{H}}, q_{\mathtt{G}}, q_{\mathtt{S}}, \varepsilon)$-UF-CMA *secure with*

$$\varepsilon = \frac{6q_{\mathtt{G}}(q_{\mathtt{G}}+1)+12}{4p-(3q_{\mathtt{G}}+2)^2} + \frac{q_{\mathtt{S}}(q_{\mathsf{H}}+q_{\mathtt{S}})}{p} + \frac{q_{\mathsf{H}}+q_{\mathtt{S}}}{p-(3q_{\mathtt{G}}+2)} + \frac{q_{\mathsf{H}}+1}{2^k} = \mathcal{O}\left(\frac{q}{2^k}\right),$$

in the generic group model of prime order $p \approx 2^{2k}$ *and the programmable random oracle model, where* q *denotes the total number of queries made by an adversary.*

Proof of Theorem 4: 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 $\mathbb{G}$ using $\mathcal{A}_{\mathsf{sig}}$ as a subroutine:

Algorithm $\mathcal{A}_{\mathsf{dlog}}$:
The algorithm is given $\mathfrak{g} = \tau(1), \tau(x), p$ as input. The goal of $\mathcal{A}_{\mathsf{dlog}}$ is to output the discrete log of $\tau(x)$, i.e., find x.

1. Initialize the list $\mathcal{L} = \{(\tau(x), 1, 0), (\mathfrak{g}, 0, 1)\}$ and $\mathsf{H}_{\mathsf{resp}} = \{\}$ where $\mathsf{H}_{\mathsf{resp}}$ 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 6. Now we consider the following cases:
   (a) Whenever $\mathcal{A}_{\mathsf{sig}}$ submits a query w to the random oracle H:
      – If there is a pair $(w, R) \in \mathsf{H}_{\mathsf{resp}}$ for some string R then return R.
      – Otherwise, select $R \in \mathbb{Z}_{2^k}$ uniformly at random and add (w, R) to the set $\mathsf{H}_{\mathsf{resp}}$.
      – If w has the form $w = (\mathfrak{a} \| m)$ where the value $\mathfrak{a}$ has not been observed previously (i.e., is not in the list $\mathcal{L}$) then we query $b = \mathtt{DLog}(\mathfrak{a})$ and add $(\tau(b), 0, b)$ 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 $b = \mathtt{DLog}(\mathfrak{a})$ and add $(\tau(b), 0, b)$ to $\mathcal{L}$.
      – Otherwise $(\mathfrak{a}, a, b) \in \mathcal{L}$ (i.e., $\mathfrak{a} = \tau(ax+b)$). In this case we query $\mathtt{Inv}(\mathfrak{a}) = \tau(-ax-b)$, output the result and add the result $(\tau(-ax-b), -a, -b) \in \mathcal{L}$.
   (c) Whenever $\mathcal{A}_{\mathsf{sig}}$ submits a query $\mathfrak{a}, \mathfrak{b}$ to the generic group oracle $\mathtt{Mult}(\mathfrak{a}, \mathfrak{b})$:

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

- First, if the element $\mathfrak{a}$ (resp. $\mathfrak{b}$) is not in the list $\mathcal{L}$ then query $b_0 = \mathtt{DLog}(\mathfrak{a})$ (resp. $b_1 = \mathtt{DLog}(\mathfrak{b})$) 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 compute $\mathtt{Mult}(\mathfrak{a}, \mathfrak{b}) = \tau((a_0 + a_1)x + b_0 + b_1)$ and add the element $(\tau((a_0 + a_1)x + b_0 + b_1), a_0 + a_1, b_0 + b_1) \in \mathcal{L}$.

(d) Whenever $\mathcal{A}_{\mathsf{sig}}$ submits a query m to the signing oracle $\mathsf{Sign}(\cdot)$:
- We use the procedure described in Figure 6 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 tuples $(\tau(s), 0, s)$, $(\tau(xe), e, 0)$ and $(\tau(s - xe), -e, s)$ 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 $x = (d-b)(a-c)^{-1}$.[6] 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 $\sigma = (s, e)$ and m we first compute $I_\sigma = \mathtt{Mult}(\mathtt{Pow}(\mathfrak{g}, s), \mathtt{Inv}(\mathtt{Pow}(\tau(x), e))) = \tau(s - xe)$ and then check to see if we previously had any tuple of the form $(I_\sigma, a, b) \in \mathcal{L}$.
   (a) If no such tuple exists we return $\perp$.
   (b) Otherwise, we let a, b be given such that $I_\sigma = \tau(ax + b)$.
   - If $a + e = 0$ then we return $\perp$.
   - Otherwise, we return $x = (s - b)(a + e)^{-1}$.

Analysis. We first remark that if the signature is valid then we must have $e = \mathsf{H}(I_\sigma \| m)$ and $\mathtt{DLog}(I_\sigma) = s - xe = ax + b$ 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 $\mathcal{A}_{\mathsf{sig}}$ outputs a signature as well as the probability our algorithm outputs $\perp$ after $\mathcal{A}_{\mathsf{sig}}$ 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 $\mathsf{FailtoSign}$. Note that every time the attacker queries the signing oracle, we would generate a new query to the random oracle H. Thus, we would have at most $q_{\mathsf{H}} + q_{\mathsf{S}}$ input/output pairs recorded for the random oracle. Since $I = \tau(s - xe)$ represents a fresh/randomly selected group element of size p, the probability that (I, m) is one of the inputs is at most $(q_{\mathsf{H}} + q_{\mathsf{S}})/p$. Applying union bound over q_{S} queries to the signing oracle, we conclude that

$$\Pr[\mathsf{FailtoSign}] \leq \frac{q_{\mathsf{S}}(q_{\mathsf{H}} + q_{\mathsf{S}})}{p}.$$

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

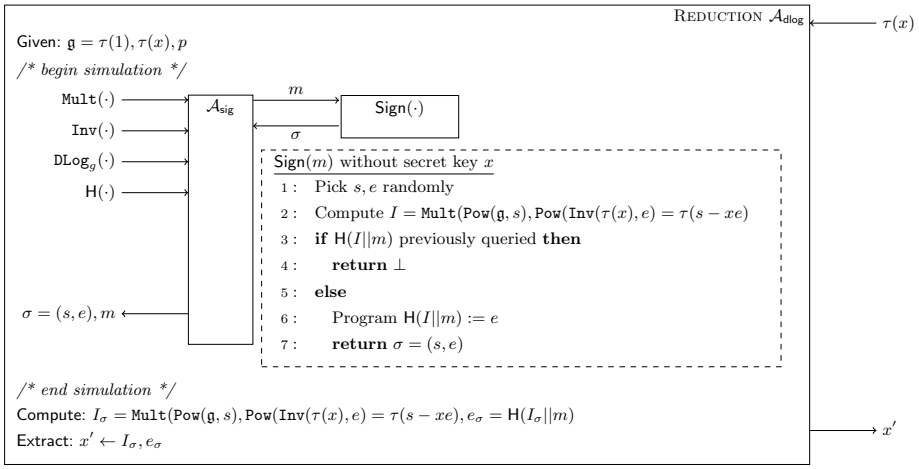

Figure 6. A reduction to the discrete-log attacker $\mathcal{A}_{\mathsf{dlog}}$ from the Schnorr signature attacker $\mathcal{A}_{\mathsf{sig}}$.

(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 $\mathcal{L}$. If $I_\sigma \notin \mathcal{L}$ we can view $I_\sigma = \tau(s - xe)$ as a uniformly random binary string from a set of size at least $p - |\mathcal{L}|$ 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 $(q_{\mathsf{H}} + q_{\mathsf{S}})/(p - |\mathcal{L}|)$. 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,

$$\Pr[\mathsf{FailtoFind}(I_\sigma)] \leq \frac{q_{\mathsf{H}} + q_{\mathsf{S}}}{p - |\mathcal{L}|} + \frac{1}{2^k}.$$

(3) Finally, we could output failure if $a = -e$ (Algorithm 3.(b)). We call this event $\mathsf{BadQuery}$. We first note that by construction we ensure that the tuple (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 $\mathtt{DLog}(I)$ before querying the random oracle. We call a random oracle query $x = I \| m$ "bad" if $\mathsf{H}(x) = -a$ where the tuple (I, a, b) has already been recorded. Recall that if there were two recorded tuples (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

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

Now we have shown that

$$\Pr[\mathsf{DLogChal}^{\mathcal{T}}_{\mathcal{A}_{\mathsf{dlog}}}(k) = 1]$$

$$\geq \Pr[\mathsf{SigForge}^{\tau}_{\mathcal{A}_{\mathsf{sig}},\Pi}(k)=1] - \Pr[\mathsf{FailtoSign}] - \Pr[\mathsf{FailtoFind}(I_\sigma)] - \Pr[\mathsf{BadQuery}]$$
$$\geq \Pr[\mathsf{SigForge}^{\tau}_{\mathcal{A}_{\mathsf{sig}},\Pi}(k)=1] - \frac{q_{\mathsf{S}}(q_{\mathsf{H}}+q_{\mathsf{S}})}{p} - \frac{q_{\mathsf{H}}+q_{\mathsf{S}}}{p-|\mathcal{L}|} - \frac{1}{2^k} - \frac{q_{\mathsf{H}}}{2^k}\ .$$

Since $|\mathcal{L}| \leq 3q_{\mathsf{G}}+2$, we can apply Lemma 1 to conclude that

$$\begin{aligned}&\Pr[\mathsf{SigForge}^{\tau}_{\mathcal{A}_{\mathsf{sig}},\Pi}(k)=1]\\ &\leq \frac{6q_{\mathsf{G}}(q_{\mathsf{G}}+1)+12}{4p-(3q_{\mathsf{G}}+2)^2} + \frac{q_{\mathsf{S}}(q_{\mathsf{H}}+q_{\mathsf{S}})}{p} + \frac{q_{\mathsf{H}}+q_{\mathsf{S}}}{p-(3q_{\mathsf{G}}+2)} + \frac{q_{\mathsf{H}}+1}{2^k}\\ &= \mathcal{O}\left(\frac{q}{2^k}\right)\ ,\end{aligned}$$

where q denotes the total number of queries made by the adversary. Note that for all $q_{\mathsf{G}} < \frac{2^k-2}{3}$ we have that $(3q_{\mathsf{G}}+2)^2 < 2^{2k} \approx p$ and therefore it clearly holds that $\frac{6q_{\mathsf{G}}(q_{\mathsf{G}}+1)+12}{4p-(3q_{\mathsf{G}}+2)^2} = \mathcal{O}\left(\frac{q}{2^k}\right)$. □

C.2 Missing Proofs from Section 4

Reminder of Claim 1. $\Pr[\mathsf{FailtoSign}] \leq \frac{q_{\mathsf{S}}(q_{\mathsf{H}}+q_{\mathsf{S}})}{p}$.

Proof of Claim 1: Note that every time the attacker queries the signing oracle, we would generate a new query to the random oracle H. Thus, we would have at most $q_{\mathsf{H}}+q_{\mathsf{S}}$ input/output pairs recorded for the random oracle. Since $I_i = \tau(s_i - x_i e_i)$ represents a fresh/randomly selected group element of size p, the probability that (I_i, m_i) is one of the inputs is at most $(q_{\mathsf{H}}+q_{\mathsf{S}})/p$. Applying union bound over q_{S} queries to the signing oracle, we conclude that $\Pr[\mathsf{FailtoSign}] \leq \frac{q_{\mathsf{S}}(q_{\mathsf{H}}+q_{\mathsf{S}})}{p}$. □

Reminder of Claim 2. $\Pr[\mathsf{FailtoFind}(I_{i*})] \leq \frac{q_{\mathsf{H}}+q_{\mathsf{S}}}{p-|\mathcal{L}|} + \frac{1}{2^k}$.

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

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

Proof of Claim 3: Recall that the event $\mathsf{BadQuery}$ happens if $\vec{a} = -e_{i*}\hat{u}_{i*}$. Note that by construction we ensure that the tuple $(I, \vec{a}, b)$ 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 $\mathtt{DLog}(I)$ before querying the random oracle. Now define a subset $\widehat{\mathcal{L}} \subset \mathcal{L}$ as the set of tuples $(\mathfrak{a}, \vec{a}, b) \in \mathbb{G} \times \mathbb{Z}_p^N \times \mathbb{Z}_p$ such that $\vec{a}$ has exactly *one* nonzero element. Now we call a random oracle query $x = (I\|m)$ "bad" if $\mathsf{H}(x) = -\widetilde{a}$ where the tuple $(I, \vec{a}, b) \in \widehat{\mathcal{L}}$ has already been recorded and the nonzero element of $\vec{a}$ is $\widetilde{a}$ (Recall that if there were two recorded tuples $(I, \vec{a}, b)$ and $(I, \vec{c}, d)$ then our algorithm would have already found a BRIDGE^N instance). 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 $\Pr[\mathsf{BadQuery}] \leq \frac{q_{\mathsf{H}}}{2^k}$. □

C.3 Missing Proofs from Section 5

Reminder of Theorem 7. *Let $p > 2^{2k}$ be a prime number and $N \in \mathbb{N}$ be a parameter. Let $(\mathcal{A}_{\mathsf{pre}}, \mathcal{A}_{\mathsf{on}})$ be a pair of generic algorithms with an encoding map $\tau : \mathbb{Z}_p \to \mathbb{G}$ such that $\mathcal{A}_{\mathsf{pre}}$ outputs an S-bit hint and $\mathcal{A}_{\mathsf{on}}$ makes at most $q_{\mathsf{G}}^{\mathsf{on}} := q_{\mathsf{G}}^{\mathsf{on}}(k)$ queries to the generic group oracles. Then*

$$\Pr\left[\mathsf{BridgeChal}^{\tau,N}_{\mathcal{A}_{\mathsf{on}},\mathsf{str}_\tau}(k) = 1\right] \leq \widetilde{\mathcal{O}}\left(\frac{SN(q_{\mathsf{G}}^{\mathsf{on}} + N)(q_{\mathsf{G}}^{\mathsf{on}} + 2N)}{p}\right),$$

where the randomness is taken over the selection of τ, the random coins of $\mathcal{A}_{\mathsf{on}}$, and the random coins used by the challenger in the bridge game (the hint $\mathsf{str}_\tau = \mathcal{A}_{\mathsf{pre}}^{\mathsf{GO}}(\mathfrak{g})$ is selected independently of the random coins used by the challenger). In particular, if $q_{\mathsf{G}}^{\mathsf{on}} \geq 10N(1 + 2\log p)$ and $S \geq 10\log(8p)$, then

$$\Pr\left[\mathsf{BridgeChal}^{\tau,N}_{\mathcal{A}_{\mathsf{on}},\mathsf{str}_\tau}(k) = 1\right] \leq \frac{12SN(q_{\mathsf{G}}^{\mathsf{on}})^2 \log p}{p}.$$

Proof of Theorem 7: We follow the same idea from [CK18, Theorem 2], which shows the relationship between the size of the hint S and the probability ε that can win the multi-user bridge-finding game. Say that a map τ is *good* if $(\mathcal{A}_{\mathsf{pre}}, \mathcal{A}_{\mathsf{on}})$ wins the multi-user bridge-finding game with probability at least $\varepsilon/2$ on τ. That is, τ is *good* if we have

$$\Pr\left[\mathsf{BridgeChal}^{\tau,N}_{\mathcal{A}_{\mathsf{on}},\mathsf{str}_\tau}(k) = 1\right] \geq \frac{\varepsilon}{2},$$

where the probability is taken over the selection of $\vec{x} = (x_1, \ldots, x_N) \in \mathbb{Z}_p^N$, the random coins of $\mathcal{A}_{\mathsf{on}}$, and the random coins used by the challenger in the bridge game (the hint $\mathsf{str}_\tau = \mathcal{A}_{\mathsf{pre}}^{\mathsf{GO}}(\mathfrak{g})$ is selected independently of the random coins used by the challenger).

Let $\mathcal{T} = \{\tau_1, \tau_2, \ldots\}$ be the set of good labeling maps. Then by a standard averaging argument ([AB09, Lemma A.12]), an $(\varepsilon/2)$-fraction of injective mappings from $\mathbb{Z}_p$ to $\mathbb{G}$ are good. One could also observe that the number of injective mappings from $\mathbb{Z}_p$ to $\mathbb{G}$ is $|\mathbb{G}|!/(|\mathbb{G}| - p)!$, which implies that $|\mathcal{T}| \geq (\varepsilon/2) \cdot |\mathbb{G}|!/(|\mathbb{G}| - p)!$.

We introduce Lemma 3 which has been adapted from Abadi et al. [AFK87] that an average-case multi-user bridge-finding game implies a worst-case multi-user bridge-finding game as shown below.

Lemma 3. *For a prime p, let $(\mathcal{A}_{\mathsf{pre}}, \mathcal{A}_{\mathsf{on}})$ be a pair of generic algorithms for $\mathbb{Z}_p$ on $\mathbb{G}$ with an encoding map $\tau : \mathbb{Z}_p \to \mathbb{G}$ such that $\mathcal{A}_{\mathsf{pre}}$ outputs an S-bit hint str_τ and $\mathcal{A}_{\mathsf{on}}$ makes at most q_{on} generic group oracle queries. Then there exists a generic algorithm $\mathcal{A}'_{\mathsf{on}}$ that makes at most $q_{\mathsf{on}} + 2N \log p + N$ generic group oracle queries, and, for every $\tau : \mathbb{Z}_p \to \mathbb{G}$, if $\Pr_{\vec{x},\mathcal{A}_{\mathsf{on}}}[\mathsf{BridgeChal}^{\tau,N}_{\mathcal{A}_{\mathsf{on}},\mathsf{str}_\tau}(k,\vec{x}) = 1] \geq \varepsilon$, then for every $\vec{x} = (x_1, \ldots, x_N) \in \mathbb{Z}_p^N$, $\Pr_{\mathcal{A}'_{\mathsf{on}}}[\mathsf{BridgeChal}^{\tau,N}_{\mathcal{A}'_{\mathsf{on}},\mathsf{str}_\tau}(k,\vec{x}) = 1] \geq \varepsilon$.*

Proof. Algorithm $\mathcal{A}'_{\mathsf{on}}(\mathsf{str}_\tau, \tau(x_1), \ldots, \tau(x_N))$ executes the following:

- Sample a random $(r_1, \ldots, r_N) \leftarrow\!\!\$\, \mathbb{Z}_p^N$ and computes $\mathtt{Mult}(\tau(x_i), \tau(r_i))$ for each $i \in [N]$, using at most $2N \log p + N$ group operations, since it would need at most $2 \log p$ group operations to compute $\tau(r_i)$ given each r_i for $i \in [N]$ and we have extra one operation of $\mathtt{Mult}$ to compute $\mathtt{Mult}(\tau(x_i), \tau(r))$ for each $i \in [N]$.
- Run $\mathcal{A}_{\mathsf{on}}(\mathsf{str}_\tau, \tau(x_1 + r_1), \ldots, \tau(x_N + r_N))$, which plays a multi-user bridge-finding game $\mathsf{BridgeChal}^{\tau,N}_{\mathcal{A}_{\mathsf{on}},\mathsf{str}_\tau}(k,\vec{x}')$, where $\vec{x}' = (x_1 + r, \ldots, x_N + r)$.
- When $\mathcal{A}_{\mathsf{on}}$ outputs 1, then algorithm $\mathcal{A}'_{\mathsf{on}}$ also outputs 1, and vice versa.

Observe that during the execution of $\mathcal{A}'_{\mathsf{on}}$, we run $\mathcal{A}_{\mathsf{on}}$ on $(\tau(x_1 + r_1), \ldots, \tau(x_N + r_N))$, which is the image of a uniformly random point in $\mathbb{Z}_p^N$. Since we assume that the bridge-finding game with $\mathcal{A}_{\mathsf{on}}$ succeeds with probability at least ε over the random selection of $\vec{x} = (x_1, \ldots, x_N) \in \mathbb{Z}_p^N$ and its coins, we can conclude that the bridge-finding game with $\mathcal{A}'_{\mathsf{on}}$ also succeeds with probability at least ε only over the selection of its coins. □

Now we are back to the proof of Theorem 7. Lemma 3 implies that there exists a pair of generic algorithms $(\mathcal{A}_{\mathsf{pre}}, \mathcal{A}'_{\mathsf{on}})$ such that for every $\tau \in \mathcal{T}$ and every $\vec{x} = (x_1, \ldots, x_N) \in \mathbb{Z}_p^N$, $\mathcal{A}'_{\mathsf{on}}$ making at most $q_{\mathsf{G}}^{\mathsf{on}'} = q_{\mathsf{G}}^{\mathsf{on}} + 2N \log p + N$ generic group oracle queries and we have $\Pr_{\mathcal{A}'_{\mathsf{on}}}[\mathsf{BridgeChal}^{\tau,N}_{\mathcal{A}'_{\mathsf{on}},\mathsf{str}_\tau}(k,\vec{x}) = 1] \geq \varepsilon/2$.

Now from Lemma 2, we can use $(\mathcal{A}_{\mathsf{pre}}, \mathcal{A}'_{\mathsf{on}})$ to compress any mapping $\tau \in \mathcal{T}$ to a binary string of length at most

$$\log \frac{|\mathbb{G}|!}{(|\mathbb{G}| - p)!} + S + 1 - \frac{(\varepsilon/2)p}{6q_{\mathsf{G}}^{\mathsf{on}'}(q_{\mathsf{G}}^{\mathsf{on}'} + N)(N \log p + 1)},$$

where the encoding scheme works with probability at least $1/2$. The incompressibility argument[7] of De et al. [DTT10] says that this length must be at least $\log |\mathcal{T}| - \log 2$. Hence, we have that

$$\log \frac{|\mathbb{G}|!}{(|\mathbb{G}| - p)!} + S + 1 - \frac{(\varepsilon/2)p}{6q_{\mathsf{G}}^{\mathsf{on}'}(q_{\mathsf{G}}^{\mathsf{on}'} + N)(N \log p + 1)} \geq \log \frac{|\mathbb{G}|!}{(|\mathbb{G}| - p)!} - \log \frac{4}{\varepsilon},$$

[7] [DTT10, Fact 8.1] says that "Suppose there is a randomized encoding procedure $\mathsf{Enc} : \{0,1\}^n \times \{0,1\}^r \to \{0,1\}^m$ and a decoding procedure $\mathsf{Dec} : \{0,1\}^m \times \{0,1\}^r \to \{0,1\}^n$ such that $\Pr_{r \in U_r}[\mathsf{Dec}(\mathsf{Enc}(x,r),r) = x] \geq \delta$. Then $m \geq n - \log(1/\delta)$." Since $\{0,1\}^n$ corresponds to the set of mapping $\mathcal{T}$, setting $\delta = 1/2$, we get the result.

which implies that

$$S \geq \frac{\varepsilon p}{6(q_{\mathsf{G}}^{\mathsf{on}} + 2N\log p + N)(q_{\mathsf{G}}^{\mathsf{on}} + 2N\log p + 2N)(N\log p + 1)} - \log\frac{8}{\varepsilon}.$$

Without loss of generality, we may assume that $\varepsilon \geq 1/p$, since the probability to find a BRIDGE^N instance is not lower than solving 1-out-of-N discrete log problem (see Corollary 1), which is also not lower than $1/p$ when guessing them randomly. Hence, we have that $\log\frac{8}{\varepsilon} \leq \log(8p)$ and we get

$$6\left(S + \log(8p)\right)(q_{\mathsf{G}}^{\mathsf{on}} + 2N\log p + N)(q_{\mathsf{G}}^{\mathsf{on}} + 2N\log p + 2N)(N\log p + 1) \geq \varepsilon p,$$

which implies that

$$\begin{aligned}\varepsilon &\leq \frac{6}{p}\left(S + \log(8p)\right)(q_{\mathsf{G}}^{\mathsf{on}} + N + 2N\log p)(q_{\mathsf{G}}^{\mathsf{on}} + 2N + 2N\log p)(N\log p + 1)\\ &= \widetilde{\mathcal{O}}\left(\frac{SN(q_{\mathsf{G}}^{\mathsf{on}} + N)(q_{\mathsf{G}}^{\mathsf{on}} + 2N)}{p}\right),\end{aligned}$$

i.e., if the size of the hint is bounded then the probability of winning the 1-out-of-N generic bridge-finding game is also bounded correspondingly. In particular, if $q_{\mathsf{on}} \geq 10N(1 + 2\log p)$ and $S \geq 10\log(8p)$, then we observe that

$$\begin{aligned}\varepsilon &\leq \frac{6}{p}\left(S + \log(8p)\right)(q_{\mathsf{G}}^{\mathsf{on}} + N + 2N\log p)(q_{\mathsf{G}}^{\mathsf{on}} + 2N + 2N\log p)(N\log p + 1)\\ &\leq \frac{6}{p}(1.1S)(1.1q_{\mathsf{G}}^{\mathsf{on}})^2(1.5N\log p) \leq \frac{12SN(q_{\mathsf{G}}^{\mathsf{on}})^2\log p}{p},\end{aligned}$$

where we have $N\log p + 1 \leq 1.5N\log p$ because $p > 2^{2k}$ implies $N\log p > 2kN \geq 2k$ and $N\log p + 1 \leq (1 + 1/(2k))N\log p \leq 1.5N\log p$. □

Reminder of Lemma 2. *Let $\mathbb{G}$ be the set of binary strings of length ℓ such that $2^\ell \geq p$ for a prime p. Let $\mathcal{T} = \{\tau_1, \tau_2, \ldots\}$ be a subset of the labeling functions from $\mathbb{Z}_p$ to $\mathbb{G}$. Let $(\mathcal{A}_{\mathsf{pre}}, \mathcal{A}_{\mathsf{on}})$ be a pair of generic algorithms for $\mathbb{Z}_p$ on $\mathbb{G}$ such that for every $\tau \in \mathcal{T}$ and every $\vec{x} = (x_1, \ldots, x_N) \in \mathbb{Z}_p^N$, $\mathcal{A}_{\mathsf{pre}}$ outputs an S-bit advice string, $\mathcal{A}_{\mathsf{on}}$ makes at most q_{on} oracle queries, and $(\mathcal{A}_{\mathsf{pre}}, \mathcal{A}_{\mathsf{on}})$ satisfy $\Pr_{\mathcal{A}_{\mathsf{on}}}\left[\mathsf{BridgeChal}^{\tau,N}_{\mathcal{A}_{\mathsf{on}},\mathsf{str}_\tau}(k,\vec{x}) = 1\right] \geq \varepsilon$, where $\mathsf{str}_\tau = \mathcal{A}^{\mathsf{GO}}_{\mathsf{pre}}(\tau(1))$. Then, there exists a randomized encoding scheme that compresses elements of $\mathcal{T}$ to bitstrings of length at most*

$$\log\frac{|\mathbb{G}|!}{(|\mathbb{G}| - p)!} + S + 1 - \frac{\varepsilon p}{6q_{\mathsf{on}}(q_{\mathsf{on}} + N)(N\log p + 1)},$$

and succeeds with probability at least $1/2$.

Proof of Lemma 2: The proof works largely the same as it from [CK18] except for some modifications. In the proof of [CK18], they build an encoding and a decoding routine to construct a randomized encoding scheme. The encoding

routine takes as input an encoding function $\tau : \mathbb{Z}_p \to \mathbb{G}$ and a random binary string r, and outputs a compression of τ. The decoding routine reverts this operation; it takes as input a compressed representation of τ and the same random string r as used in the encoding routine, and outputs the original τ. One modification from the proof of [CK18] is that in the encoding routine, we also need to handle $\texttt{Inv}(\cdot)$ while [CK18] only considers $\texttt{Mult}(\cdot,\cdot)$. Another difference is that the multi-user bridge-finding game $\mathsf{BridgeChal}^{\tau,N}_{\mathcal{A}_{\mathsf{on}}}(k,\vec{x})$ that $\mathcal{A}_{\mathsf{on}}$ plays does not aim for finding discrete-log solutions but for finding a BRIDGE^N instance.

Encoding Routine:

Input: an encoding function $\tau : \mathbb{Z}_p \to \mathbb{G}$, and parameters $d, R \in \mathbb{Z}^+$, where $\mathbb{Z}^+$ denotes the set of positive integers.

(1) Compute and write the S-bit hint $\mathsf{str}_\tau \leftarrow \mathcal{A}_{\mathsf{pre}}(\mathfrak{g})$ into the encoding.
(2) Write the image of τ using $\log \binom{|\mathbb{G}|}{p}$ bits into the encoding.
(3) Initialize Table to an empty list. It will store the pairs $(y, \tau(y))$, i.e., an element in the image of τ and its discrete log value.
(4) Repeat the following d times in total:
   (a) Choose the first N strings in the lexicographical order of the image of τ that are not in the table. Call these strings $\tau(x_1), \ldots, \tau(x_N)$ and add the pairs $(X_1, \tau(x_1)), \ldots, (X_N, \tau(x_N))$ to the table, where for each $i \in [N]$, X_i is the indeterminate that represents discrete log value x_i of $\tau(x_i)$ that the decoder does not yet know.
   (b) Run $\mathcal{A}_{\mathsf{on}}(\mathsf{str}_\tau, \tau(x_1), \ldots, \tau(x_N))$ up to R times using independent randomness from the encoder's random string in each run. Write the index $r^* \in [R]$ of the successful execution (i.e., it finds a BRIDGE^N instance) into the encoding using $\log R$ bits. If $\mathcal{A}_{\mathsf{on}}$ fails on all R execution, then return $\perp$ and abort the entire routine.
   (c) Write a placeholder of $\log q_{\mathsf{on}}$ zeros in the encoding, and it will be overwritten by the actual number of queries until it finds a BRIDGE^N instance during the next step.
   (d) Rerun $\mathcal{A}_{\mathsf{on}}(\mathsf{str}_\tau, \tau(x_1), \ldots, \tau(x_N))$ using the r^{*th} random tape. It processes each generic group oracle query (see **Handling** $\texttt{Mult}(\mathfrak{y}_1, \mathfrak{y}_2)$ **and Handling** $\texttt{Inv}(\mathfrak{y})$ as described below) and as soon as it finds a BRIDGE^N instance it will mark the actual number of total queries $q^* \leq q_{\mathsf{on}}$ into its placeholder. Note that it will count the number of queries altogether whenever it queries $\texttt{Mult}(\cdot,\cdot)$ or $\texttt{Inv}(\cdot)$.
   (e) For the remaining indeterminates that are not yet resolved, simply replace them with the discrete log values (naïve encoding), since the encoder knows all of τ. If this is the i^{th} entry in Table, then it requires $\log(p - i + 1)$ bits since all the entries that were already in the table are constants.
   (f) Write the remaining values that are not yet in the table to the encoding in lexicographical order.

Handling $\texttt{Mult}(\mathfrak{y}_1, \mathfrak{y}_2)$:

(1) If either of $\mathfrak{y}_1$ or $\mathfrak{y}_2$ is not in the image of τ, reply $\perp$ and continue to the next query.
(2) If either of $\mathfrak{y}_1$ or $\mathfrak{y}_2$ is not in the table, then this is a *fresh* query input. For each such argument $\mathfrak{s} \in \{\mathfrak{y}_1, \mathfrak{y}_2\}$, add the pair $(\texttt{DLog}(\mathfrak{s}), \mathfrak{s})$ to the table and write $\mathfrak{s}$ to the encoding using $\log(p - l)$ bits, where l is the number of labels already in the table.
(3) Otherwise, look up the tuples $(f_1, \mathfrak{y}_1), (f_2, \mathfrak{y}_2)$ in the table where $f_1 = f_1(X_1, \ldots, X_N)$ and $f_2 = f_2(X_1, \ldots, X_N)$ are linear polynomials of N indeterminates $X_1, \ldots, X_N$, and compute $f_1 + f_2$.
   (a) If $(f_1 + f_2, \texttt{Mult}(\mathfrak{y}_1, \mathfrak{y}_2))$ is already in the table, simply reply with $\texttt{Mult}(\mathfrak{y}_1, \mathfrak{y}_2)$.
   (b) If $(f_1 + f_2, \texttt{Mult}(\mathfrak{y}_1, \mathfrak{y}_2))$ is not in the table, then add $\texttt{Mult}(\mathfrak{y}_1, \mathfrak{y}_2)$ to the encoding and reply with $\texttt{Mult}(\mathfrak{y}_1, \mathfrak{y}_2)$. Write $\texttt{Mult}(\mathfrak{y}_1, \mathfrak{y}_2)$ into the encoding requires $\log(p-l)$ bits, where l is the number of labels already in the table.
   (c) If $\texttt{Mult}(\mathfrak{y}_1, \mathfrak{y}_2)$ is in the table but its corresponding discrete log value in the table is a linear polynomial $f = f(X_1, \ldots, X_N)$ such that f is not identical to $f_1 + f_2$ (i.e., the coefficients for X_i's are not all the same), then encode the reply to this query as a pointer to the table entry $(f, \texttt{Mult}(\mathfrak{y}_1, \mathfrak{y}_2))$ and add this pointer to the encoding. Then use the equation $f = f_1 + f_2$ to derive an equation $X_j = g(X_1, \ldots, \overline{X}_j, \ldots, X_N)$ for some j and a linear polynomial g of $X_1, \ldots, X_N$ except for X_j, and replace X_j by $g(X_1, \ldots, \overline{X}_j, \ldots, X_N)$ in the table.
   In this case, we successfully found a BRIDGE^N instance, so stop the execution of the algorithm and indicate the *early stop* by writing the actual number of queries $q^* \leq q_{\mathsf{on}}$ into its placeholder above.

Handling $\texttt{Inv}(\mathfrak{y})$:

(1) If $\mathfrak{y}$ is not in the image of τ, reply $\perp$ and continue to the next query.
(2) If $\mathfrak{y}$ is not in the table, then this is an *fresh* query input. Add the pair $(\texttt{DLog}(\mathfrak{y}), \mathfrak{y})$ to the table.
(3) Otherwise, look up the tuple $(f, \mathfrak{y})$ in the table where $f = f(X_1, \ldots, X_N)$ is a linear polynomial of N indeterminates $X_1, \ldots, X_N$, and compute $-f$.
   (a) If $(-f, \texttt{Inv}(\mathfrak{y}))$ is already in the table, simply reply with $\texttt{Inv}(\mathfrak{y})$.
   (b) If $(-f, \texttt{Inv}(\mathfrak{y}))$ is not in the table, then add $\texttt{Inv}(\mathfrak{y})$ to the encoding and reply with $\texttt{Inv}(\mathfrak{y})$. Write $\texttt{Inv}(\mathfrak{y})$ into the encoding requires $\log(p - l)$ bits, where l is the number of labels already in the list.
   (c) If $\texttt{Inv}(\mathfrak{y})$ is in the table but its corresponding discrete log value in the table is a linear polynomial $\tilde{f} = \tilde{f}(X_1, \ldots, X_N)$ such that f is

not identical to $\tilde{f}$ (i.e., the coefficients for X_i's are not all the same), then encode the reply to this query as a pointer to the table entry $(\tilde{f}, \mathtt{Inv}(\mathfrak{y}))$ and add this pointer to the encoding.
Then use the equation $f = -\tilde{f}$ to derive an equation $X_j = g(X_1, \ldots, \overline{X}_j, \ldots, X_N)$ for some j and a linear polynomial g of $X_1, \ldots, X_N$ except for X_j, and replace X_j by $g(X_1, \ldots, \overline{X}_j, \ldots, X_N)$ in the table.
In this case, we successfully found a BRIDGE^N instance, so stop the execution of the algorithm and indicate the *early stop* by writing the actual number of queries $q^* \leq q_{\mathsf{on}}$ into its placeholder above.

Decoding Routine:

The decoding routine is given the encoded string and recovers τ as follows:

(1) Extract str_τ from the first S bits of the encoding.
(2) Extract the image of τ from the next $\log \binom{|\mathbb{G}|}{p}$ bits from the encoding.
(3) Initialize Table (input/output pairs for τ) as an empty list.
(4) Extract $\tau(1)$ from the encoding and add $(1, \tau(1))$ to table.
(5) Repeat the following d times in total:
  (a) Choose the first N strings in the lexicographical order of the image of τ that are not in Table. Call these strings $\mathfrak{y}_1, \ldots, \mathfrak{y}_N$ and add the pairs $(X_1, \mathfrak{y}_1), \ldots, (X_N, \mathfrak{y}_N)$ to the table, where $X_1, \ldots, X_N$ denotes the indeterminates that represent the discrete log of $\mathfrak{y}_i$'s for $i = 1, \ldots, N$.
  (b) Decode $r^* \in [R]$ after reading $\log R$ bits from the encoding.
  (c) Decode $q^* \in [q_{\mathsf{on}}]$ after reading $\log q_{\mathsf{on}}$ bits from the encoding.
  (d) Run $\mathcal{A}_{\mathsf{on}}(\mathsf{str}_\tau, \mathfrak{y}_1, \ldots, \mathfrak{y}_N)$ for q^* queries using the $r^{*\,th}$ random tape allocated for this instance of $\mathcal{A}_{\mathsf{on}}$.
    i. If $\mathcal{A}_{\mathsf{on}}$ makes the query $\mathtt{Mult}(\mathfrak{s}_1, \mathfrak{s}_2)$:
      A. If either $\mathfrak{s}_1$ or $\mathfrak{s}_2$ is not in the image of τ, reply $\perp$.
      B. If either $\mathfrak{s}_1$ or $\mathfrak{s}_2$ is in the image of τ, but not in the table, then add entries $(S_i, \mathfrak{s}_1)$ and $(S_j, \mathfrak{s}_2)$ to the table, where i and j are the smallest integers such that S_i and S_j are the fresh indeterminates in the table.
      C. Search for the table and find the pairs $(f_1, \mathfrak{s}_1)$ and $(f_2, \mathfrak{s}_2)$, and compute the linear polynomial $f_1 + f_2$.
      (1) If $(f_1 + f_2, \mathfrak{w})$ is already in the table, then reply with $\mathfrak{w}$ and continue to the next query.
      (2) If there is no such $\mathfrak{w}$ in the table, and if this is not the last $(q^{*\,th})$ query of this execution, then read $\log(p - l)$ bits from the encoding, where l is the number of labels in the table, and decode the bits to output $\mathfrak{w}$. Reply with $\mathfrak{w}$, and add $(f_1 + f_2, \mathfrak{w})$ to the table and continue to the next query.

(3) If there is no such $\mathfrak{w}$ in the table, and if this is the last (q^{*th}) query, then we know that we have a bridge instance here. Thus, read a $\log|\mathsf{Table}|$-bit pointer from the encoding and search for the entry in the table with that pointer, say $(f, \mathfrak{w})$. Solve the equation $f = f_1 + f_2 \mod p$ for the first indeterminate S in the equation. Replace every S in the polynomials in the table with the solution of the equation so that there is no longer S in the entire table.

ii. If $\mathcal{A}_{\mathsf{on}}$ makes the query $\mathtt{Inv}(\mathfrak{s})$:
   A. If $\mathfrak{s}$ is not in the image of τ, reply $\perp$.
   B. If $\mathfrak{s}$ is in the image of τ, but not in the table, then add an entry $(S_i, \mathfrak{s})$ to the table, where i is the smallest integer such that S_i is the fresh indeterminate in the table.
   C. Search for the table and find the pair $(f, \mathfrak{s})$, and compute the linear polynomial $-f$.
   (1) If $(-f, \mathfrak{w})$ is already in the table, then reply with $\mathfrak{w}$ and continue to the next query.
   (2) If there is no such $\mathfrak{w}$ in the table, and if this is not the last (q^{*th}) query of this execution, then read $\log(p-l)$ bits from the encoding, where l is the number of labels in the table, and decode the bits to output $\mathfrak{w}$. Now reply with $\mathfrak{w}$, and add $(-f, \mathfrak{w})$ to the table and continue to the next query.
   (3) If there is no such $\mathfrak{w}$ in the table, and if this is the last (q^{*th}) query, then we know that we have a bridge instance here. Thus, read a $\log|\mathsf{Table}|$-bit pointer from the encoding and search for the entry in the table with this pointer, say $(\tilde{f}, \mathfrak{w})$. Solve the equation $\tilde{f} = -f \mod p$ for the first indeterminate S in the equation. Replace every S in the polynomials in the table with the solution of the equation so that there is no longer S in the entire table.

iii. Read the values of all the remaining indeterminates as they appear in the table.

The total encoding length will be calculated as follows:

- the hint τ (S bits),
- the encoding of the image of τ ($\log\binom{|\mathbb{G}|}{p}$ bits),
- the index r^* which denotes the successful execution of $\mathcal{A}_{\mathsf{on}}$ ($d \log R$ bits),
- the actual number of total queries q^* ($d \log q_{\mathsf{on}}$ bits),
- for the l^{th} entry that is added to the table ($0 \le l < |\mathsf{Table}|$), if the entry
  - corresponds to an indeterminate that has been resolved by finding a BRIDGE^N instance (at most $\log|\mathsf{Table}|$ bits),
  - otherwise, $\log(p-l)$ bits, and
- the remaining discrete log values, encoded using $\log(p-l)$ bits each, where $l \in \{|\mathsf{Table}|, \ldots, p-1\}$.

Note that at the end of each of the d executions of $\mathcal{A}_{\mathsf{on}}$, we either learn the discrete log of one label $\tau(\cdot)$, or, we find a bridge instance in the table. In this case, we get $\log(p-l)$ bits of information on τ at a cost of at most $\log R + \log q_{\mathsf{on}} + \log|\mathsf{Table}|$ bits. Since each execution of $\mathcal{A}_{\mathsf{on}}$ adds at most $3q_{\mathsf{on}}+N$ rows to the table (N inputs, q_{on} query replies, and at most $2q_{\mathsf{on}}$ fresh query inputs), we observe that $|\mathsf{Table}| \le d(3q_{\mathsf{on}}+N)$. Since $l \le |\mathsf{Table}|$, the net profit is lower bounded by

$$\log \frac{p-l}{\log R + \log q_{\mathsf{on}} + \log|\mathsf{Table}|} \ge \log \frac{p-|\mathsf{Table}|}{Rq_{\mathsf{on}}|\mathsf{Table}|} \ge \log \frac{p-d(3q_{\mathsf{on}}+N)}{Rdq_{\mathsf{on}}(3q_{\mathsf{on}}+N)}.$$

We further observe that $\log \frac{p-d(3q_{\mathsf{on}}+N)}{Rdq_{\mathsf{on}}(3q_{\mathsf{on}}+N)} \ge 1$ for $d = \lfloor p/((2Rq_{\mathsf{on}}+1)(3q_{\mathsf{on}}+N))\rfloor$. Thus, with this value of d, we have that this net profit becomes at least 1 bit for each of the d executions of $\mathcal{A}_{\mathsf{on}}$. Hence, the total bitlength of the encoding is at most

$$\begin{aligned} S + \log\binom{|\mathbb{G}|}{p} + \sum_{l=0}^{p}\log(p-l) - d &= \log\frac{|\mathbb{G}|!}{(|\mathbb{G}|-p)!} + S - d \\ &\le \log\frac{|\mathbb{G}|!}{(|\mathbb{G}|-p)!} + S - \frac{p}{(2Rq_{\mathsf{on}}+1)(3q_{\mathsf{on}}+N)} + 1 \\ &\le \log\frac{|\mathbb{G}|!}{(|\mathbb{G}|-p)!} + S - \frac{p}{6Rq_{\mathsf{on}}(q_{\mathsf{on}}+N)} + 1. \end{aligned}$$

We want that this randomized encoding routine succeeds with probability at lease $1/2$. In the encoding routine, we run $\mathcal{A}_{\mathsf{on}}$ up to R times and see if there is any successful execution. Since each execution fails with probability at most ε, if we choose $R = (1+N\log p)/\varepsilon$, then such R executions all fail with the probability at most

$$(1-\varepsilon)^R \le e^{-\varepsilon R} \le 2^{-\varepsilon R} \le 2^{-1-N\log p} \le 2^{-1-\log\binom{p}{N}} \le \frac{1}{2}\cdot\binom{p}{N}^{-1},$$

since $p^N \ge \binom{p}{N}$. Union bounding over all $\binom{p}{N}$ different inputs on $\mathcal{A}_{\mathsf{on}}$, the probability that the encoding routine fails is at most $1/2$. Hence, with this value of R, the encoding length is at most

$$\log\frac{|\mathbb{G}|!}{(|\mathbb{G}|-p)!} + S + 1 - \frac{\varepsilon p}{6q_{\mathsf{on}}(q_{\mathsf{on}}+N)(N\log p+1)},$$

which completes the proof. □

Reminder of Theorem 8. *Let $\Pi = (\mathsf{Kg}, \mathsf{Sign}, \mathsf{Vfy})$ be a key-prefixed Schnorr signature scheme and $p > 2^{2k}$ be a prime number. Let $N \in \mathbb{N}$ be a parameter and $(\mathcal{A}^{\mathsf{pre}}_{\mathsf{sig}}, \mathcal{A}^{\mathsf{on}}_{\mathsf{sig}})$ be a pair of generic algorithms with an encoding map $\tau : \mathbb{Z}_p \to \mathbb{G}$ such that $\mathcal{A}^{\mathsf{pre}}_{\mathsf{sig}}$ makes at most $q^{\mathsf{pre}}_{\mathsf{H}}$ queries to the random oracle $\mathsf{H} : \{0,1\}^* \to \{0,1\}^{k_1}$ and outputs an S-bit hint $\mathsf{str}_{\tau,\mathsf{H}}$, and $\mathcal{A}^{\mathsf{on}}_{\mathsf{sig}}$ makes at most $q^{\mathsf{on}}_{\mathsf{G}} := q^{\mathsf{on}}_{\mathsf{G}}(k)$ queries*

to the generic group oracles and at most $q_{\mathsf{H}}^{\mathsf{on}}$ *queries to the random oracle. Then* $\Pr\left[\mathsf{SigForge}^{\tau,N}_{\mathcal{A}^{\mathsf{on}}_{\mathsf{sig}},\mathsf{str}_{\tau,\mathsf{H}},\Pi}(k)=1\right] \le \varepsilon$, *with*

$$\varepsilon = \widetilde{\mathcal{O}}\left(\frac{SN(q_{\mathtt{G}}^{\mathsf{on}}+N)(q_{\mathtt{G}}^{\mathsf{on}}+2N)}{p}\right) + \frac{Nq_{\mathsf{H}}^{\mathsf{pre}}q_{\mathsf{S}}^{\mathsf{on}}}{p^2} + \frac{q_{\mathsf{S}}^{\mathsf{on}}(q_{\mathsf{S}}^{\mathsf{on}}+q_{\mathsf{H}}^{\mathsf{on}})}{p} + \frac{4(q_{\mathsf{H}}^{\mathsf{on}}+1)}{2^{k_1}} + \frac{N^2(S+k_1)}{p},$$

where $q_{\mathsf{S}}^{\mathsf{on}}$ *denotes the number of queries to the signing oracle and the randomness is taken over the selection of* τ *and the random coins of* $\mathcal{A}^{\mathsf{on}}_{\mathsf{sig}}$ *(the hint* $\mathsf{str}_{\tau,\mathsf{H}} = \mathcal{A}^{\mathsf{pre},\mathtt{GO}}_{\mathsf{sig}}(\mathfrak{g})$ *is selected independently of the random coins used by the challenger). In particular, if* $q_{\mathtt{G}}^{\mathsf{on}} \ge 10N(1+2\log p)$ *and* $S \ge 10\log(8p)$, *then*

$$\varepsilon = \frac{12SN(q_{\mathtt{G}}^{\mathsf{on}})^2\log p}{p} + \frac{Nq_{\mathsf{H}}^{\mathsf{pre}}q_{\mathsf{S}}^{\mathsf{on}}}{p^2} + \frac{q_{\mathsf{S}}^{\mathsf{on}}(q_{\mathsf{S}}^{\mathsf{on}}+q_{\mathsf{H}}^{\mathsf{on}})}{p} + \frac{4(q_{\mathsf{H}}^{\mathsf{on}}+1)}{2^{k_1}} + \frac{N^2(S+k_1)}{p}.$$

Proof of Theorem 8: Given a generic adversary with preprocessing $(\mathcal{A}^{\mathsf{pre}}_{\mathsf{sig}}, \mathcal{A}^{\mathsf{on}}_{\mathsf{sig}})$ attacking Schnorr signature scheme, we construct the following efficient generic algorithm with preprocessing $(\mathcal{A}^{\mathsf{pre}}_{\mathsf{bridge}}, \mathcal{A}^{\mathsf{on}}_{\mathsf{bridge}})$ which tries to succeed in the 1-out-of-N generic BRIDGE^N-finding game $\mathsf{BridgeChal}^{\tau,N}_{\mathcal{A}^{\mathsf{on}}_{\mathsf{bridge}},\mathsf{str}_{\tau,\mathsf{H}}}(k,\vec{x})$:

Algorithm $(\mathcal{A}^{\mathsf{pre}}_{\mathsf{bridge}}, \mathcal{A}^{\mathsf{on}}_{\mathsf{bridge}})$**:**
The algorithm is given $p, \mathfrak{g} = \tau(1), \mathsf{pk}_i = \tau(x_i), 1 \le i \le N$ as input.

1. $\mathcal{A}^{\mathsf{pre}}_{\mathsf{bridge}}$ simply runs $\mathcal{A}^{\mathsf{pre}}_{\mathsf{sig}}$ to generate an S-bit hint $\mathsf{str}_{\tau,\mathsf{H}}$.
2. $\mathcal{A}^{\mathsf{on}}_{\mathsf{bridge}}$ initializes the set $\mathsf{H}_{\mathsf{resp}} = \{\}$ which stores the random oracle input/output pairs observed during online processing.
3. $\mathcal{A}^{\mathsf{on}}_{\mathsf{bridge}}$ takes the hint $\mathsf{str}_{\tau,\mathsf{H}}$ and initializes the list $\mathcal{L} = \{(\tau(1), \vec{0}, 1), (\mathsf{pk}_i, \hat{u}_i, 0)$ for $1 \le i \le N\}$, and runs $\mathcal{A}^{\mathsf{on}}_{\mathsf{sig}}$ with the hint $\mathsf{str}_{\tau,\mathsf{H}}$ and a number of access to the generic group oracles $\mathtt{GO} = (\mathtt{Mult}(\cdot,\cdot), \mathtt{Inv}(\cdot)), \mathtt{DLog}(\cdot), \mathsf{Sign}_i(\cdot)$ for $1 \le i \le N$, and the random oracle $\mathsf{H}(\cdot)$. We maintain the invariant that every output of a generic group query during the online phase appears in the list $\mathcal{L}$. We consider the following cases:
   (a) Whenever $\mathcal{A}^{\mathsf{on}}_{\mathsf{sig}}$ submits a query w to the random oracle H:
      – If there is a pair $(w, R) \in \mathsf{H}_{\mathsf{resp}}$ for some string $R \in \{0,1\}^{k_1}$ then return R.
      – Otherwise, select $R \leftarrow_{\$} \{0,1\}^{k_1}$ and add (w, R) to $\mathsf{H}_{\mathsf{resp}}$.
      – If w has the form $w = (\mathsf{pk}_j \| \mathfrak{a} \| m_i)$ where the value $\mathfrak{a}$ has not been observed previously (i.e., is not in the list $\mathcal{L}$ then we query $b = \mathtt{DLog}(\mathfrak{a})$ and add $(\mathfrak{a}, \vec{0}, b)$ to $\mathcal{L}$.
   (b) Whenever $\mathcal{A}^{\mathsf{on}}_{\mathsf{sig}}$ submits a query $\mathfrak{a}$ to the generic group oracle $\mathtt{Inv}(\cdot)$:
      – If $\mathfrak{a}$ is not in $\mathcal{L}$ then we immediately query $b = \mathtt{DLog}(\mathfrak{a})$ and add $(\mathfrak{a}, \vec{0}, b)$ to $\mathcal{L}$.
      – Otherwise, $(\mathfrak{a}, \vec{a}, b) \in \mathcal{L}$ for some $\vec{a}$ and b. Then we query $\mathtt{Inv}(\mathfrak{a}) = \tau(-\vec{a} \cdot \vec{x} - b)$, output the result and add $(\tau(-\vec{a} \cdot \vec{x} - b), -\vec{a}, -b)$ to $\mathcal{L}$.
   (c) Whenever $\mathcal{A}^{\mathsf{on}}_{\mathsf{sig}}$ submits a query $\mathfrak{a}, \mathfrak{b}$ to the generic group oracle $\mathtt{Mult}(\cdot,\cdot)$:
      – If the element $\mathfrak{a}$ (resp. $\mathfrak{b}$) is not in $\mathcal{L}$ then query $b_0 = \mathtt{DLog}(\mathfrak{a})$ (resp. $b_1 = \mathtt{DLog}(\mathfrak{b})$) and add the element $(\mathfrak{a}, \vec{0}, b_0)$ (resp. $(\mathfrak{b}, \vec{0}, b_1)$) to $\mathcal{L}$.

- Otherwise both elements $(\mathfrak{a}, \vec{a}_0, b_0), (\mathfrak{b}, \vec{a}_1, b_1) \in \mathcal{L}$. Then we return $\mathtt{Mult}(\mathfrak{a}, \mathfrak{b}) = \tau((\vec{a}_0 + \vec{a}_1) \cdot \vec{x} + b_0 + b_1)$ and add $(\tau((\vec{a}_0 + \vec{a}_1) \cdot \vec{x} + b_0 + b_1), \vec{a}_0 + \vec{a}_1, b_0 + b_1) \in \mathcal{L}$.

(d) Whenever $\mathcal{A}_{\mathsf{sig}}^{\mathsf{on}}$ submits a query m_i to the signing oracle $\mathsf{Sign}(x_j, \cdot)$:
- The attacker tries to forge a signature without knowledge of the secret key x_j, relying on the ability to program the random oracle as follows:
  i. Pick s_i, e_i randomly and compute $I_i = \tau(s_i - x_j e_i) = \mathtt{Mult}(\mathfrak{s}_i, \mathtt{Pow}(\mathsf{pk}_j, -e_i))$ where $\mathfrak{s}_i = \mathtt{Pow}(\mathfrak{g}, s_i)$.
  ii. If $\mathsf{H}(\mathsf{pk}_j \| I_i \| m_i)$ has been previously queried, then return $\perp$.
  iii. Otherwise, program $\mathsf{H}(\mathsf{pk}_j \| I_i \| m_i) := e_i$ and return $\sigma_i = (s_i, e_i)$.
- We remark that a side effect of querying the Sign_j oracle is the addition of the tuples $(\tau(s_i), \vec{0}, s_i)$, $(\tau(x_j e_i), e_i \hat{u}_i, 0)$ and $(\tau(s_i - x_j e_i), -e_i \hat{u}_i, s_i)$ 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 $(\mathfrak{y}, \vec{a}, b) \in \mathcal{L}$ and $(\mathfrak{y}, \vec{c}, d) \in \mathcal{L}$ for $(\vec{a}, b) \neq (\vec{c}, d)$ then we can immediately have a BRIDGE^N instance $(\tau((\vec{a} - \vec{c}) \cdot \vec{x}), \vec{a} - \vec{c}, 0) \in \mathcal{L}$ and $(\tau(d - b), \vec{0}, d - b) \in \mathcal{L}$ since $\tau((\vec{a} - \vec{c}) \cdot \vec{x}) = \tau(d - b)$. Thus, without loss of generality, we can assume that each string $\mathfrak{y}$ occurs at most once in the list $\mathcal{L}$.

4. After $\mathcal{A}_{\mathsf{sig}}^{\mathsf{on}}$ outputs $\sigma_{i*} = (s_{i*}, e_{i*})$ and m_{i*}, identify the index $i* \in [N]$ such that $\mathsf{Vf}(\mathsf{pk}_{i*}, m_{i*}, \sigma_{i*}) = 1$.
5. Compute $\tau(-e_{i*} x_{i*}) = \mathtt{Inv}(\mathtt{Pow}(\tau(x_{i*}), e_{i*}))$. This will ensure that the elements $(\tau(-e_{i*} x_{i*}), -e_{i*} \hat{u}_{i*}, 0)$ and $(\tau(e_{i*} x_{i*}), e_{i*} \hat{u}_{i*}, 0)$ are both added to $\mathcal{L}$.
6. Compute $\mathfrak{s}_{i*} = \mathtt{Pow}(\mathfrak{g}, s_{i*})$ to ensure that $(\mathfrak{s}_{i*}, \vec{0}, s_{i*}) \in \mathcal{L}$.
7. Finally, compute $I_{i*} = \mathtt{Mult}(\mathfrak{s}_{i*}, \tau(-e_{i*} x_{i*})) = \tau(s_{i*} - x_{i*} e_{i*})$ which ensures that $(I_{i*}, -e_{i*} \hat{u}_{i*}, s_{i*}) \in \mathcal{L}$.

Analysis. We first remark that if the signature is valid then we must have $e_{i*} = \mathsf{H}(I_{i*} \| m_{i*})$ and $\mathtt{DLog}(I_{i*}) = s_{i*} - x_{i*} e_{i*} = \vec{a} \cdot \vec{x} + b$.

We first analyze the probability that the attacker will outputs a valid signature forgery after when we use random oracle programming to simulate the honest signing oracles without the secret keys. We introduce two bad events $\mathsf{FailtoSign}$ and $\mathsf{DetectProgramming}$ such that the probability of a valid signature forgery is at least

$$\Pr\left[\mathsf{SigForge}^{\tau,N}_{\mathcal{A}^{\mathsf{on}}_{\mathsf{sig},\mathsf{str}_{\tau,\mathsf{H}}},\Pi}(k) = 1\right] - \Pr[\mathsf{DetectProgramming}] - \Pr[\mathsf{FailtoSign}],$$

where $\Pr[\mathsf{SigForge}^{\tau,N}_{\mathcal{A}^{\mathsf{on}}_{\mathsf{sig},\mathsf{str}_{\tau,\mathsf{H}}},\Pi}(k) = 1]$ denotes the probability that the attacker is successful when playing with the real signing oracles.

$\mathsf{FailtoSign}$ is the event where our reduction outputs $\perp$ in Step 3.(d) due signing oracle failure, i.e., during some query i, we generate I_i and find that the random oracle query $\mathsf{H}(\mathsf{pk}_j \| I_i \| m_i)$ was made previously during the *online* phase. As before, we can argue that

$$\Pr[\mathsf{FailtoSign}] \leq \frac{q_{\mathsf{S}}^{\mathsf{on}}(q_{\mathsf{S}}^{\mathsf{on}} + q_{\mathsf{H}}^{\mathsf{on}})}{p},$$

i.e., see Claim 1.

Intuitively, $\mathsf{DetectProgramming}$ denotes the event that the random oracle query $\mathsf{H}(\mathsf{pk}_j \| I_i \| m_i)$ was made in the *offline phase*. Now we have the following claim to upper bound the probability of the event $\mathsf{DetectProgramming}$:

Claim 4. $\Pr[\mathsf{DetectProgramming}] \leq \frac{N q_{\mathsf{H}}^{\mathsf{pre}} q_{\mathsf{S}}^{\mathsf{on}}}{p^2}$.

We push the proof of Claim 4 right after the proof of our theorem. If the event $\mathsf{DetectProgramming}$ occurs, then it is possible that the signature the hint $\mathsf{str}_{\tau,\mathsf{H}}$ is somehow correlated with $\mathsf{H}(\mathsf{pk}_j \| I_i \| m_i)$, which could potentially allow the signature attacker to halt early because it guesses that we are programming the random oracle. Assuming that the event $\mathsf{DetectProgramming}$ does not occur, the attacker will not be able to distinguish between a programmed response and the signatures generated via the honest signing oracle.

We now upper bound the probability that the attacker outputs a valid signature forgery without the bridge event BRIDGE^N occurring. Unlike our previous analysis, we rely on a compression argument. $\mathsf{FailtoFind}(I_{i*})$ denotes the event that the signature is valid, but we find that I_{i*} was not previously recorded in our list $\mathcal{L}$ before we computed $\mathtt{Mult}(\mathfrak{s}_{i*}, \tau(-e_{i*}x_{i*}))$. Observe that if the attacker had previously queried $\mathsf{H}(\mathsf{pk}_{i*} \| I_{i*} \| m_{i*})$ during the online phase, then I_{i*} would have been added to $\mathcal{L}$ already. Thus, in this case, a signature forgery allows us to predict the value of the random oracle $e_{i*} = \mathsf{H}(\mathsf{pk}_{i*} \| I_{i*} \| m_{i*})$. Intuitively, if $\mathsf{FailtoFind}(I_{i*})$ is too large then we can derive a contradiction by compressing the random oracle.

Similarly, we define $\mathsf{BadQuery}$ as the event that at some point for an element of the form $(I, -e\hat{u}_i, s) \in \mathcal{L}$, we make the random oracle query $\mathsf{H}(\mathsf{pk}_i \| I \| m)$ and receive the response $\mathsf{H}(\mathsf{pk}_i \| I \| m) = e$. Such a lucky query would allow the attacker to forge a signature without causing BRIDGE^N to occur. Intuitively, if such an event occurs we can also compress the random oracle, i.e., by using $\log q_{\mathsf{H}}^{\mathsf{on}} \leq k_1$ bits to encode the index of the lucky query.

Claim 5. $\Pr[\mathsf{FailtoFind}(I_{i*}) \cup \mathsf{BadQuery}] \leq \frac{4(q_{\mathsf{H}}^{\mathsf{on}} + 1)}{2^{k_1}} + \frac{N^2(S + k_1)}{p}$.

We also push the proof of Claim 5 after the proof of our theorem. Now we have shown that

$$\begin{aligned}
&\Pr\left[\mathsf{BridgeChal}^{\tau,N}_{\mathcal{A}^{\mathsf{on}}_{\mathsf{bridge},\mathsf{str}_{\tau,\mathsf{H}}}}(k) = 1\right] \\
&\geq \Pr\left[\mathsf{SigForge}^{\tau,N}_{\mathcal{A}^{\mathsf{on}}_{\mathsf{sig},\mathsf{str}_{\tau,\mathsf{H}}},\Pi}(k) = 1\right] - \Pr[\mathsf{DetectProgramming}] - \Pr[\mathsf{FailtoSign}] \\
&\qquad\qquad - \Pr[\mathsf{FailtoFind}(I_{i*})] - \Pr[\mathsf{BadQuery}] \\
&\geq \Pr\left[\mathsf{SigForge}^{\tau,N}_{\mathcal{A}^{\mathsf{on}}_{\mathsf{sig},\mathsf{str}_{\tau,\mathsf{H}}},\Pi}(k) = 1\right] - \frac{N q_{\mathsf{H}}^{\mathsf{pre}} q_{\mathsf{S}}^{\mathsf{on}}}{p^2} - \frac{q_{\mathsf{S}}^{\mathsf{on}}(q_{\mathsf{S}}^{\mathsf{on}} + q_{\mathsf{H}}^{\mathsf{on}})}{p} - \frac{4(q_{\mathsf{H}}^{\mathsf{on}} + 1)}{2^{k_1}} - \frac{N^2(S + k_1)}{p}.
\end{aligned}$$

Finally, we can apply Theorem 7 to conclude that

$$\Pr\left[\mathsf{SigForge}^{\tau,N}_{\mathcal{A}^{\mathsf{on}}_{\mathsf{sig},\mathsf{str}_{\tau,\mathsf{H}}},\Pi}(k) = 1\right]$$

$$\leq \widetilde{\mathcal{O}}\left(\frac{SN(q_{\mathsf{G}}^{\mathsf{on}}+N)(q_{\mathsf{G}}^{\mathsf{on}}+2N)}{p}\right)+\frac{Nq_{\mathsf{H}}^{\mathsf{pre}}q_{\mathsf{S}}^{\mathsf{on}}}{p^2}+\frac{q_{\mathsf{S}}^{\mathsf{on}}(q_{\mathsf{S}}^{\mathsf{on}}+q_{\mathsf{H}}^{\mathsf{on}})}{p}+\frac{4(q_{\mathsf{H}}^{\mathsf{on}}+1)}{2^{k_1}}+\frac{N^2(S+k_1)}{p}.$$

Similar to Theorem 7, if $q_{\mathsf{G}}^{\mathsf{on}} \geq 10N(1+2\log p)$ and $S \geq 10\log(8p)$, then

$$\Pr\left[\mathsf{SigForge}^{\tau,N}_{\mathcal{A}^{\mathsf{on}}_{\mathsf{sig},\mathsf{str}_{\tau,\mathsf{H}}},\Pi}(k)=1\right]$$
$$\leq \frac{12SN(q_{\mathsf{G}}^{\mathsf{on}})^2\log p}{p}+\frac{Nq_{\mathsf{H}}^{\mathsf{pre}}q_{\mathsf{S}}^{\mathsf{on}}}{p^2}+\frac{q_{\mathsf{S}}^{\mathsf{on}}(q_{\mathsf{S}}^{\mathsf{on}}+q_{\mathsf{H}}^{\mathsf{on}})}{p}+\frac{4(q_{\mathsf{H}}^{\mathsf{on}}+1)}{2^{k_1}}+\frac{N^2(S+k_1)}{p},$$

which completes the proof. □

Reminder of Claim 4. $\Pr[\mathsf{DetectProgramming}] \leq \frac{Nq_{\mathsf{H}}^{\mathsf{pre}}q_{\mathsf{S}}^{\mathsf{on}}}{p^2}$.

Proof of Claim 4: Let $I_i = \mathtt{Pow}(\mathfrak{g}, r_i)$ be the value generated during the i^{th} query to the signing oracle. Let $B_{i,j}$ denote the bad event that the preprocessing attacker previously submitted a query of the form $\mathsf{H}(\mathsf{pk}_j \| I_i \| \cdot)$. Since pk_j and I_i are selected randomly, we have $\Pr[B_{i,j}] \leq q_{\mathsf{H}}^{\mathsf{pre}}/p^2$. We now define the event $\mathsf{DetectProgramming} = \bigcup_{j\leq N, i\leq q_{\mathsf{S}}} B_{i,j}$. Applying union bounds, we have

$$\Pr[\mathsf{DetectProgramming}] = \Pr[\cup_{i,j} B_{i,j}] \leq \sum_{i,j}\Pr[B_{i,j}] \leq \frac{Nq_{\mathsf{H}}^{\mathsf{pre}}q_{\mathsf{S}}^{\mathsf{on}}}{p^2}. \qquad \square$$

Reminder of Claim 5. $\Pr[\mathsf{FailtoFind}(I_{i*}) \cup \mathsf{BadQuery}] \leq \frac{4(q_{\mathsf{H}}^{\mathsf{on}}+1)}{2^{k_1}} + \frac{N^2(S+k_1)}{p}$.

Proof of Claim 5: Fixing τ and H, and for some parameter t (which we will determine later), we define $\varepsilon_{F,\tau,\mathsf{H}}$ to be

$$\varepsilon_{F,\tau,\mathsf{H}} := \min_{Y:|Y|\leq Nt} \Pr\left[\mathsf{FailtoFind}(I_{i*}) \cup \mathsf{BadQuery}\right],$$

where the probability is taken over the random selection of $x_1,\ldots,x_N \in \mathbb{Z}_p \setminus Y$ and the random coins of $\mathcal{A}^{\mathsf{on}}_{\mathsf{sig},\mathsf{str}_{\tau,\mathsf{H}}}$. We can argue that removing $Nt \approx NS$ points does not impact this probability, i.e., $\varepsilon_{F,\tau,\mathsf{H}} \approx \varepsilon'_{F,\tau,\mathsf{H}}$ where $\varepsilon'_{F,\tau,\mathsf{H}}$ is the same probability when $x_1,\ldots,x_N \in \mathbb{Z}_p$ are chosen without any restrictions. In particular, $\varepsilon'_{F,\tau,H} \leq \varepsilon_{F,\tau,\mathsf{H}} + N^2t/p$ where the term N^2t/p upper bounds the probability that sample some point $x_i \in Y$.

We now sample t uniformly random strings $R_1,\ldots,R_t$, e.g., by querying the random oracle at $\mathcal{O}(t)$ fixed points. We repeat the following experiment t times to extract t input/output predictions, where each iteration $i \in [t]$ uses R_i as its random coin when sampling $\vec{x}^i$ and q^i:

(1) For $i \in [t]$, sample $\vec{x}^i = (x_1^i,\ldots,x_N^i)$ subject to the restriction that $x_j^i \neq x_{j'}^{i'}$ for any pair $(i,j) \neq (i',j')$.
(2) Pick $q^i \in [0, q_{\mathsf{H}}^{\mathsf{on}}]$ uniformly at random.

(3) If $q^i = 0$, then wait for the (attempted) forged signature $\sigma_i = (s_i, e_i)$ to be output, compute $I_i = \tau(s_i - e_i x_i^j)$, and output the prediction $\mathsf{H}(\tau(x_i^j)\|I_i\|m_i) = e_i$.
(4) If instead $q^i > 0$, then we simulate $\mathcal{A}^{\mathsf{on}}_{\mathsf{sig},\mathsf{str}_{\tau,\mathsf{H}}}$ until the query q^i to the random oracle. At this point, the query has the form $\mathsf{H}(\mathsf{pk}_j\|I\|m)$ for some I such that $(I, e\hat{u}_j, 0) \in \mathcal{L}^i$. In this case, we can extract e from $\mathcal{L}^i$ and output the prediction $\mathsf{H}(\mathsf{pk}_j\|I\|m) = e$.

During each iteration i, we output a correct input/output pair with probability at least $\varepsilon_{F,\tau,\mathsf{H}}/(q_{\mathsf{H}}^{\mathsf{on}}+1)$. Thus, the procedure above correctly outputs t input/output pairs with probability at least $(\varepsilon_{F,\tau,\mathsf{H}}/(q_{\mathsf{H}}^{\mathsf{on}}+1))^t$.

Let $\mathsf{SucceedExp}$ denotes an event that the experiment above correctly outputs t correct input/output pairs (without querying the random oracle at these t points) when τ and H are picked randomly.

Now let $\varepsilon_F \coloneqq \mathbb{E}_{\tau,\mathsf{H}}[\varepsilon_{F,\tau,\mathsf{H}}]$ and suppose that $\varepsilon_F > 4(q_{\mathsf{H}}^{\mathsf{on}}+1)2^{-k_1}$ for contradiction, then we can use Markov inequality to argue that

$$\begin{aligned}
\Pr_{\tau,\mathsf{H}}\left[\varepsilon_{F,\tau,\mathsf{H}} > \frac{2(q_{\mathsf{H}}^{\mathsf{on}}+1)}{2^{k_1}}\right] &= 1 - \Pr_{\tau,\mathsf{H}}\left[\varepsilon_{F,\tau,\mathsf{H}} \le \frac{2(q_{\mathsf{H}}^{\mathsf{on}}+1)}{2^{k_1}}\right] \\
&= 1 - \Pr_{\tau,\mathsf{H}}\left[1-\varepsilon_{F,\tau,\mathsf{H}} \ge 1 - \frac{2(q_{\mathsf{H}}^{\mathsf{on}}+1)}{2^{k_1}}\right] \\
&\ge 1 - \frac{1-\varepsilon_F}{1-2(q_{\mathsf{H}}^{\mathsf{on}}+1)2^{-k_1}} \\
&> \frac{\varepsilon_F/2}{1-\varepsilon_F/2} \ge \frac{\varepsilon_F}{2}.
\end{aligned}$$

Thus, if we pick τ and H randomly and run the above procedure, we will succeed with probability at least

$$\Pr[\mathsf{SucceedExp}] \ge \frac{\varepsilon_F}{2}\left(\frac{\varepsilon_{F,\tau,\mathsf{H}}}{q_{\mathsf{H}}^{\mathsf{on}}+1}\right)^t \ge \frac{\varepsilon_F}{2}2^{(1-k_1)t} > 2(q_{\mathsf{H}}^{\mathsf{on}}+1)2^{-k_1}2^{(1-k_1)t}.$$

However, on the other hand, Lemma 4 below tells us that

$$\Pr[\mathsf{SucceedExp}] \le 2^{-k_1 t + S}.$$

Picking $t = S + k_1$, we derive a contradiction, since, with this value of t we have

$$2(q_{\mathsf{H}}^{\mathsf{on}}+1)2^{-k_1 S - k_1^2 + S} < \Pr[\mathsf{SucceedExp}] \le 2^{-k_1 S - k_1^2 + S}.$$

The contradiction comes from the assumption that $\varepsilon_F > 4(q_{\mathsf{H}}^{\mathsf{on}}+1)2^{-k_1}$. Thus, we have

$$\begin{aligned}
\varepsilon'_F &:= \Pr[\mathsf{FailtoFind}(I_{i*}) \cup \mathsf{BadQuery}] \\
&= \mathbb{E}_{\tau,\mathsf{H}}[\varepsilon'_{F,\tau,\mathsf{H}}] \\
&\le \mathbb{E}_{\tau,\mathsf{H}}\left[\varepsilon_{F,\tau,\mathsf{H}} + \frac{N^2(S+k_1)}{p}\right]
\end{aligned}$$

$$= \varepsilon_F + \frac{N^2(S+k_1)}{p} \leq \frac{4(q_{\mathsf{H}}^{\mathsf{on}}+1)}{2^{k_1}} + \frac{N^2(S+k_1)}{p}. \qquad \square$$

Lemma 4 has appeared in prior work in various forms. To the best of our knowledge, the original usage is from [DKW11] though our statement is closer to a form from [BHK+19]. Since we rephrase the lemma slightly, we include a proof of Lemma 4 below for completeness.

Definition 4 (k-bit prediction game). *Let B be a uniformly random bit string and let $\mathcal{A}$ be an algorithm that receives a hint $h = f(B) \in H$ which may depend arbitrarily on B and can additionally query B at specific indices before outputting indices $i_1, \ldots, i_k$ and bits $b_1, \ldots, b_k$. We say that $\mathcal{A}$ wins the* k-bit prediction game *if for all $j \leq k$ we have $B[i_j] = b_j$ and $\mathcal{A}$ did not previously query for $B[i_j]$.*

Lemma 4 ([DKW11,BHK+19]). *Any attacker $\mathcal{A}$ wins the k-bit prediction game with probability at most $|H|/2^k$.*

Proof. Let $\mathsf{Success}_{\mathcal{A}}$ denote the attacker's success probability when $h = f(B)$ depends on B. Suppose for contradiction that $\Pr[\mathsf{Success}_{\mathcal{A}}] > |H|2^{-k}$, then consider the algorithm $\mathcal{A}'$ which takes no hint, samples $h \in H$ uniformly, and simulates $\mathcal{A}$ with hint h. We have

$$\Pr[\mathsf{Success}_{\mathcal{A}'}] \geq \Pr[h = f(B)] \Pr[\mathsf{Success}_{\mathcal{A}}] > \frac{1}{|H|}|H|2^{-k} = 2^{-k}.$$

To obtain a contradiction, we observe that $\mathcal{A}'$ succeeds with probability at most 2^{-k}. In particular, $\mathcal{A}'$ starts with no hint and outputs $i_1, \ldots, i_k$ and bits $b_1, \ldots, b_k$ such that $B[i_j]$ has not been queried for all $j \leq k$. In this case, we can view each $B[i_j]$ as a uniformly random bit sampled *after* $\mathcal{A}'$ outputs. Thus, $\Pr[\forall j \leq k,\ b_j = B[i_j]] = 2^{-k}$, which implies that $\Pr[\mathsf{success}_{\mathcal{A}'}] \leq 2^{-k}$. $\square$

C.4 Missing Proofs from Section 6

Reminder of Theorem 9. *The Chaum-Pedersen-FDH signature scheme is $\left(N, q_{\mathsf{H}}, q_{\mathfrak{G}}, q_{\mathfrak{S}}, \mathcal{O}\left(\frac{q+N}{2^k}\right)\right)$-MU-UF-CMA secure under the generic group model of prime order $p \approx 2^{2k}$ and the programmable random oracle model, where q denotes the total number of queries made by an adversary.*

Proof Sketch of Theorem 9: Let Π be the Chaum-Pedersen-FDH signature scheme. We follow a similar reduction as in Theorem 6 using the signing oracle without knowledge of the secret key x_j in Figure 3 (top). Note that the corresponding public key is $\mathsf{pk}_j = \tau(x_j)$. Whenever the attacker queries the random oracle of the form $\mathsf{H}(h\|\mathfrak{y}\|\mathfrak{a}\|\mathfrak{b})$ we can ensure that $\mathfrak{y}, \mathfrak{a}$ and $\mathfrak{b}$ all appear in our list $\mathcal{L}$ by using the oracle $\mathtt{DLog}(\cdot)$ on any fresh input. Letting $e = \mathsf{H}(h\|\mathfrak{y}\|\mathfrak{a}\|\mathfrak{b})$,

we then compute $\mathtt{Pow}(\mathsf{pk}_j, e)$ and $\mathtt{Inv}(\mathtt{Pow}(\mathsf{pk}_j, e))$ to ensure that both the tuples $(\mathtt{Pow}(\mathsf{pk}_j, e), e\hat{u}_j, 0)$ and $(\mathtt{Inv}(\mathtt{Pow}(\mathsf{pk}_j, e)), -e\hat{u}_j, 0)$ appear in $\mathcal{L}$.

We first consider the event $\mathsf{FailtoSign}$ that our reduction outputs $\perp$ because of a signing oracle failure. Conditioning on this event not occurring the programmed signing oracle is equivalent to the real one. Thus, in our reduction the attacker successfully produces a forged signature with probability at least

$$\Pr[\mathsf{SigForge}^{\tau,N}_{\mathcal{A}_{\mathsf{sig}},\Pi}(k) = 1] - \Pr[\mathsf{FailtoSign}].$$

When $\mathcal{A}_{\mathsf{sig}}$ outputs a (potential) forgery $\sigma_{i*} = (\mathfrak{y}_{i*}, \mathfrak{a}_{i*}, \mathfrak{b}_{i*}, s_{i*})$ and the corresponding message m_{i*} for some $i* \in [N]$, we perform the following computations in hope of causing the bridge event BRIDGE^N (if it has not already):

(1) Query the oracle $\mathtt{DLog}(\cdot)$ on any fresh inputs $\mathfrak{y}_{i*}, \mathfrak{a}_{i*}, \mathfrak{b}_{i*}$ (if applicable),
(2) Compute $h_{i*} = \mathsf{H}'(\mathsf{pk}_{i*} \| m_{i*}) = \mathtt{Pow}(\mathfrak{g}, \mathsf{H}(\mathsf{pk}_{i*} \| m_{i*}))$,
(3) Compute $\mathfrak{s}_{i*} = \mathtt{Pow}(\mathfrak{g}, s_{i*})$,
(4) Compute the hash output $e_{i*} = \mathsf{H}(h_{i*} \| \mathfrak{y}_{i*} \| \mathfrak{a}_{i*} \| \mathfrak{b}_{i*})$, and
(5) Compute $A_{i*} = \mathtt{Mult}(\mathfrak{s}_{i*}, \mathtt{Inv}(\mathtt{Pow}(\mathsf{pk}_{i*}, e_{i*}))) = \tau(s_{i*} - x_{i*}e_{i*})$.

This ensures that the following tuples appear in $\mathcal{L}$:

- $(\mathfrak{s}_{i*}, \vec{0}, s_{i*})$,
- $(h_{i*}, \vec{0}, \mathsf{H}(\mathsf{pk}_{i*} \| m_{i*}))$,
- $(\mathtt{Pow}(\mathsf{pk}_{i*}, e_{i*}), \hat{u}_{i*}e_{i*}, 0)$,
- $(\mathtt{Pow}(\mathsf{pk}_{i*}, -e_{i*}), -\hat{u}_{i*}e_{i*}, 0)$, and
- $(A_{i*}, -e_{i*}\hat{u}_{i*}, s_{i^*})$.

We can now define the event $\mathsf{FailtoFind}(A_{i*})$ to be the event that A_{i*} was not previously recorded in our list $\mathcal{L}$ before we computed A_{i*} in the last step. Similarly, we let $\mathsf{BadQuery}$ denote the event that the signature is valid but for the only prior tuple $(A_{i*}, \vec{a}, b) \in \mathcal{L}$ we have $\vec{a} = -e_{i*}\hat{u}_{i*}$. If the signature is valid and neither event $\mathsf{FailtoFind}(A_{i*})$ nor $\mathsf{BadQuery}$ occurs then the bridge event BRIDGE^N must have occurred and we immediately win the game since $(A_{i*}, \vec{a}, b) \in \mathcal{L}, (A_{i*}, -e_{i*}\hat{u}_{i*}, s_{i*}) \in \mathcal{L}$ and $\vec{a} \neq -e_{i*}\hat{u}_{i*}$.

Our analysis of the failure events $\mathsf{FailtoSign}$, $\mathsf{FailtoFind}(A_{i*})$, and $\mathsf{BadQuery}$ is similar to the analysis in Theorem 6 with a few minor difference. First, we consider the hash output $\mathsf{H}(h_{i*} \| \mathfrak{y}_{i*} \| A_{i*} \| \mathfrak{b}_{i*})$ instead of $\mathsf{H}(I_{i*} \| m_{i*})$ since we are analyzing Chaum-Pedersen-FDH instead of short Schnorr Signatures. Additionally, since we are analyzing FDH when we analyze $\mathsf{BadQuery}$ we have to consider the possibility of a random oracle collisions. The attacker makes at most q_{H} random oracle queries and each query to a signing oracle generates at most 2 additional random oracle queries. Thus, the probability the attacker outputs a collision can be upper bounded as $\widetilde{q}^2/2^{2k}$ since the domain of our random oracle where $\widetilde{q} = q_{\mathsf{H}} + q_{\mathsf{G}} + 2q_{\mathsf{S}}$. Similar to Theorem 6 we can show that $\Pr[\mathsf{FailtoSign}] \leq q_{\mathsf{S}}(q_{\mathsf{H}} + q_{\mathsf{S}})/p$, $\Pr[\mathsf{FailtoFind}(A_{i*})] \leq \frac{q_{\mathsf{H}}+q_{\mathsf{S}}}{p-|\mathcal{L}|} + \frac{1}{2^{2k}}$, and $\Pr[\mathsf{BadQuery}] \leq q_{\mathsf{H}}/2^{2k} + \widetilde{q}^2/2^{2k}$, with a minor difference that the hash output is $2k$ bits instead of k bits. Since we have $|\mathcal{L}| \leq N + 3q_{\mathsf{G}} + 1$ we conclude that

$\Pr[\mathsf{SigForge}^{\tau,N}_{\mathcal{A}_{\mathsf{sig}},\Pi}(k) = 1]$

$$\begin{aligned}
&\leq \Pr[\mathsf{BridgeChal}^{\tau,N}_{\mathcal{A}}(k)=1] + \Pr[\mathsf{FailtoSign}] + \Pr[\mathsf{FailtoFind}(A_{i*})] + \Pr[\mathsf{BadQuery}] \\
&\leq \frac{q_{\mathtt{G}}N + 3q_{\mathtt{G}}(q_{\mathtt{G}}+1)/2}{p-(N+3q_{\mathtt{G}}+1)^2-N} + \frac{q_{\mathsf{S}}(q_{\mathsf{H}}+q_{\mathsf{S}})}{p} + \frac{q_{\mathsf{H}}+q_{\mathsf{S}}}{p-(N+3q_{\mathtt{G}}+1)} + \frac{q_{\mathsf{H}}+\widetilde{q}^2+1}{2^{2k}} \\
&= \mathcal{O}\left(\frac{q+N}{2^k}\right),
\end{aligned}$$

where the second inequality follows by applying Theorem 5 to upper bound $\Pr[\mathsf{BridgeChal}^{\tau,N}_{\mathcal{A}}(k)=1]$. □

Reminder of Theorem 10. *Let $\Pi = (\mathsf{Kg}, \mathsf{Sign}, \mathsf{Vfy})$ be a key-prefixed Chaum-Pedersen-FDH signature scheme and $p > 2^{2k}$ be a prime number. Let $N \in \mathbb{N}$ be a parameter and $(\mathcal{A}^{\mathsf{pre}}_{\mathsf{sig}}, \mathcal{A}^{\mathsf{on}}_{\mathsf{sig}})$ be a pair of generic algorithms with an encoding map $\tau : \mathbb{Z}_p \to \mathbb{G}$ such that $\mathcal{A}^{\mathsf{pre}}_{\mathsf{sig}}$ makes at most $q^{\mathsf{pre}}_{\mathsf{H}} < 2^{3k}$ queries to the random oracle $\mathsf{H} : \{0,1\}^* \to \{0,1\}^{2k}$ and outputs an S-bit hint $\mathsf{str}_{\tau,\mathsf{H}}$, and $\mathcal{A}^{\mathsf{on}}_{\mathsf{sig}}$ makes at most $q^{\mathsf{on}}_{\mathtt{G}} := q^{\mathsf{on}}_{\mathtt{G}}(k)$ queries to the generic group oracles and at most $q^{\mathsf{on}}_{\mathsf{H}}$ queries to the random oracle. Then $\Pr\left[\mathsf{SigForge}^{\tau,N}_{\mathcal{A}^{\mathsf{on}}_{\mathsf{sig}},\mathsf{str}_{\tau,\mathsf{H}},\Pi}(k)=1\right] \leq \varepsilon$, with*

$$\varepsilon = \widetilde{\mathcal{O}}\left(\frac{SN(q^{\mathsf{on}}_{\mathtt{G}}+N)(q^{\mathsf{on}}_{\mathtt{G}}+2N)}{p}\right) + \frac{Nq^{\mathsf{pre}}_{\mathsf{H}}q^{\mathsf{on}}_{\mathsf{S}}}{p^2} + \frac{q^{\mathsf{on}}_{\mathsf{S}}(q^{\mathsf{on}}_{\mathsf{S}}+q^{\mathsf{on}}_{\mathsf{H}})}{p} + \frac{4(q^{\mathsf{on}}_{\mathsf{H}}+\widetilde{q}^2_{\mathsf{on}}+1)}{2^{2k}} + \frac{3N^2(S+2k)}{2p},$$

where $q^{\mathsf{on}}_{\mathsf{S}}$ denotes the number of queries to the signing oracle, $\widetilde{q}_{\mathsf{on}} = q^{\mathsf{on}}_{\mathsf{H}} + 2q^{\mathsf{on}}_{\mathsf{S}}$, and the randomness is taken over the selection of τ and the random coins of $\mathcal{A}^{\mathsf{on}}_{\mathsf{sig}}$ (the hint $\mathsf{str}_{\tau,\mathsf{H}} = \mathcal{A}^{\mathsf{pre},\mathtt{GO}}_{\mathsf{sig}}(\mathfrak{g})$ is selected independently of the random coins used by the challenger).

Proof Sketch of Theorem 10: We follow a similar security reduction as in Theorem 8, except for the differences as follows:

- When the attacker tries to forge a signature on the message m_i without knowledge of the secret key x_j, s/he follow Figure 3 (top). Note that the corresponding public key is $\mathsf{pk}_j = \tau(x_j)$ for the secret key x_j.
- Whenever the attacker submits a random oracle query of the form $\mathsf{H}(\mathsf{pk}_j \| h \| \mathfrak{y} \| \mathfrak{a} \| \mathfrak{b})$ we use the oracle $\mathtt{DLog}(\cdot)$ to ensure that $\mathfrak{y}, \mathfrak{a}$ and $\mathfrak{b}$ appear in our list $\mathcal{L}$ if they do not already. Letting $e = \mathsf{H}(\mathsf{pk}_j \| h \| \mathfrak{y} \| \mathfrak{a} \| \mathfrak{b})$, we then compute $\mathtt{Pow}(\mathsf{pk}_j, e)$ and $\mathtt{Inv}(\mathtt{Pow}(\mathsf{pk}_j, e))$ to ensure that both the tuples $(\mathtt{Pow}(\mathsf{pk}_j, e), e\hat{u}_j, 0)$ and $(\mathtt{Inv}(\mathtt{Pow}(\mathsf{pk}_j, e)), -e\hat{u}_j, 0)$ appear in $\mathcal{L}$.
- After $\mathcal{A}_{\mathsf{sig}}$ outputs $\sigma_{i*} = (\mathfrak{y}_{i*}, \mathfrak{a}_{i*}, \mathfrak{b}_{i*}, s_{i*})$ and m_{i*} for some $i* \in [N]$ we perform the following computations in hope of causing the bridge event BRIDGE^N (if it has not already):
  (1) Query the oracle $\mathtt{DLog}(\cdot)$ on any fresh inputs $\mathfrak{y}_{i*}, \mathfrak{a}_{i*}, \mathfrak{b}_{i*}$ (if applicable),
  (2) Compute $h_{i*} = \mathsf{H}'(\mathsf{pk}_{i*} \| m_{i*}) = \mathtt{Pow}(\mathfrak{g}, \mathsf{H}(\mathsf{pk}_{i*} \| m_{i*}))$,
  (3) Compute $\mathfrak{s}_{i*} = \mathtt{Pow}(\mathfrak{g}, s_{i*})$,
  (4) Compute the hash output $e_{i*} = \mathsf{H}(\mathsf{pk}_{i*} \| h_{i*} \| \mathfrak{y}_{i*} \| \mathfrak{a}_{i*} \| \mathfrak{b}_{i*})$, and
  (5) Compute $A_{i*} = \mathtt{Mult}(\mathfrak{s}_{i*}, \mathtt{Inv}(\mathtt{Pow}(\mathsf{pk}_{i*}, e_{i*}))) = \tau(s_{i*} - x_{i*}e_{i*})$.
  This ensures that the following tuples appear in $\mathcal{L}$:
  - $(\mathfrak{s}_{i*}, \vec{0}, s_{i*})$,

- $(h_{i*}, \vec{0}, \mathsf{H}(\mathsf{pk}_{i*} \| m_{i*}))$,
- $(\mathtt{Pow}(\mathsf{pk}_{i*}, e_{i*}), \hat{u}_{i*} e_{i*}, 0)$,
- $(\mathtt{Pow}(\mathsf{pk}_{i*}, -e_{i*}), -\hat{u}_{i*} e_{i*}, 0)$, and
- $(A_{i*}, -e_{i*} \hat{u}_{i*}, s_{i^*})$.

To analyze the probability that the bridge event occurs we first analyze the following possibilities for the reduction to terminate early with the output $\perp$. Everything goes the same with the analysis in Theorem 8 except for the following:

(1) Consider the ith query to the signing oracle and suppose we ask for a signature for m_i under public key pk_j. Let $B_{i,j}$ denote the bad event that the preprocessing attacker previously submitted a query of the form $\mathsf{H}(\mathsf{pk}_j \| \mathsf{H}'(\mathsf{pk}_j \| m_i) \| \mathfrak{y}_i \| \mathfrak{a}_i \| \mathfrak{b}_i)$. Since we pick pk_j randomly, and as well as $\mathfrak{a}_i$ (observe that $\mathfrak{y}_i, \mathfrak{a}_i$, and $\mathfrak{b}_i$ might be correlated), we have $\Pr[B_{i,j}] \le q_{\mathsf{H}}^{\mathsf{pre}}/p^2$. We now define the event $\mathsf{DetectProgramming} = \bigcup_{j \le N, i \le q_{\mathsf{S}}} B_{i,j}$. Applying union bounds, we have

$$\Pr[\mathsf{DetectProgramming}] = \Pr[\cup_{i,j} B_{i,j}] \le \sum_{i,j} \Pr[B_{i,j}] \le \frac{N q_{\mathsf{H}}^{\mathsf{pre}} q_{\mathsf{S}}^{\mathsf{on}}}{p^2}.$$

Assuming that the event $\mathsf{DetectProgramming}$ does not occur, the attacker will not be able to distinguish between a programmed response and the original response.

(2) One way to output failure is during the signing oracle if $\mathsf{H}(\mathsf{pk}_j \| \mathsf{H}'(\mathsf{pk}_j \| m_i) \| \mathfrak{y}_i \| \mathfrak{a}_i \| \mathfrak{b}_i)$ has been previously queried during the *online* phase. We define this event as $\mathsf{FailtoSign}$. As before we can argue that $\Pr[\mathsf{FailtoSign}] \le \frac{q_{\mathsf{S}}^{\mathsf{on}}(q_{\mathsf{S}}^{\mathsf{on}} + q_{\mathsf{H}}^{\mathsf{on}})}{p}$ i.e., see Theorem 9.

(3) We also consider a bad event $\mathsf{Collision}_{\mathsf{H}'}$ which denotes the event that the attacker finds a collision on $\mathsf{H}'(\mathsf{pk}_{i*} \| \cdot)$ so that the attacker can forge a signature without the bridge event BRIDGE^N occurring. Claim 6 upper bounds the probability of this event using a similar compression argument as in Claim 5. The proof of Claim 6 is given below.

$$\Pr[\mathsf{FailtoFind} \cup \mathsf{BadQuery}] \le \frac{4(q_{\mathsf{H}}^{\mathsf{on}} + 1)}{2^{2k}} + \frac{N^2(S + 2k)}{p}.$$

Claim 6. $\Pr[\mathsf{Collision}_{\mathsf{H}'}] \le \frac{4\widetilde{q}_{\mathsf{on}}^2}{2^{2k}} + \frac{N^2(S+2k)}{2p}$, *where* $\widetilde{q}_{\mathsf{on}} = q_{\mathsf{H}}^{\mathsf{on}} + 2q_{\mathsf{S}}^{\mathsf{on}}$.

Applying Theorem 7 and from the similar argument in Theorem 8, we have

$$\begin{aligned}
&\Pr\left[\mathsf{SigForge}^{\tau,N}_{\mathcal{A}^{\mathsf{on}}_{\mathsf{sig},\mathsf{str}_{\tau,\mathsf{H}}},\Pi}(k) = 1\right] \\
&\quad \le \Pr\left[\mathsf{BridgeChal}^{\tau,N}_{\mathcal{A}^{\mathsf{on}}_{\mathsf{sig},\mathsf{str}_{\tau,\mathsf{H}}}}(k) = 1\right] + \Pr[\mathsf{DetectProgramming}] + \Pr[\mathsf{FailtoSign}] \\
&\qquad + \Pr[\mathsf{FailtoFind}] + \Pr[\mathsf{BadQuery}] + \Pr[\mathsf{Collision}_{\mathsf{H}'}] \\
&\quad \le \widetilde{\mathcal{O}}\left(\frac{SN(q_{\mathsf{G}}^{\mathsf{on}} + N)(q_{\mathsf{G}}^{\mathsf{on}} + 2N)}{p}\right) + \frac{N q_{\mathsf{H}}^{\mathsf{pre}} q_{\mathsf{S}}^{\mathsf{on}}}{p^2} + \frac{q_{\mathsf{S}}^{\mathsf{on}}(q_{\mathsf{S}}^{\mathsf{on}} + q_{\mathsf{H}}^{\mathsf{on}})}{p}
\end{aligned}$$

$$+\frac{4(q_{\mathsf{H}}^{\mathsf{on}}+\widetilde{q}_{\mathsf{on}}^2+1)}{2^{2k}}+\frac{3N^2(S+2k)}{2p},$$

which completes the proof. □

Remark 6. Similar to Theorem 8, if $q_{\mathsf{G}}^{\mathsf{on}} \geq 10N(1+2\log p)$ and $S \geq 10\log(8p)$, then we can argue that

$$\Pr\left[\mathsf{SigForge}^{\tau,N}_{\mathcal{A}^{\mathsf{on}}_{\mathsf{sig},\mathsf{str}_{\tau,\mathsf{H}}},\Pi}(k)=1\right]$$
$$\leq \frac{12SN(q_{\mathsf{G}}^{\mathsf{on}})^2\log p}{p}+\frac{Nq_{\mathsf{H}}^{\mathsf{pre}}q_{\mathsf{S}}^{\mathsf{on}}}{p^3}+\frac{q_{\mathsf{S}}^{\mathsf{on}}(q_{\mathsf{S}}^{\mathsf{on}}+q_{\mathsf{H}}^{\mathsf{on}})}{p^2}+\frac{4(q_{\mathsf{H}}^{\mathsf{on}}+\widetilde{q}_{\mathsf{on}}^2+1)}{2^{2k}}+\frac{3N^2(S+2k)}{2p},$$

for a key-prefixed Chaum-Pedersen signature scheme Π. ◁

Reminder of Claim 6. $\Pr[\mathsf{Collision}_{\mathsf{H}'}] \leq \frac{4\widetilde{q}_{\mathsf{on}}^2}{2^{2k}}+\frac{N^2(S+2k)}{2p}$, *where* $\widetilde{q}_{\mathsf{on}} = q_{\mathsf{H}}^{\mathsf{on}} + 2q_{\mathsf{S}}^{\mathsf{on}}$.

Proof of Claim 6: Fixing τ and H, and for some parameter t (which we will determine later), we define $\varepsilon_{F,\tau,\mathsf{H}}$ to be $\varepsilon_{F,\tau,\mathsf{H}} := \min_{Y:|Y|\leq Nt}\Pr[\mathsf{Collision}_{\mathsf{H}'}]$, where the probability is taken over the random selection of $x_1,\ldots,x_N \in \mathbb{Z}_p \setminus Y$ and the random coins of $\mathcal{A}^{\mathsf{on}}_{\mathsf{sig},\mathsf{str}_{\tau,\mathsf{H}}}$. Similar to Claim 5, we have that $\varepsilon'_{F,\tau,\mathsf{H}} \leq \varepsilon_{F,\tau,\mathsf{H}} + N^2t/p$ where $\varepsilon'_{F,\tau,\mathsf{H}}$ is the same probability when $x_1,\ldots,x_N \in \mathbb{Z}_p$ are chosen without any restrictions. We also sample t uniformly random strings $R_1,\ldots,R_t$ by querying the random oracle at $\mathcal{O}(t)$ fixed points and repeat the following experiment t times to extract t input/output predictions by using R_i as its random coin for each $i \in [t]$.

Let $z_1,\ldots,z_{\widetilde{q}_{\mathsf{on}}}$ be the queries to the random oracles either directly or indirectly through the signing oracle, and consider the following experiment:

(1) For $i \in [t]$, sample $\vec{x}^i = (x_1^i,\ldots,x_N^i)$ subject to the restriction that $x_j^i \neq x_{j'}^{i'}$ for any pair $(i,j) \neq (i',j')$.
(2) Pick a pair $(q_1^i, q_2^i) \in [\widetilde{q}_{\mathsf{on}}] \times [\widetilde{q}_{\mathsf{on}}]$ uniformly at random subject to the restriction that $q_1^i < q_2^i$.
(3) Simulate $\mathcal{A}^{\mathsf{on}}_{\mathsf{sig},\mathsf{str}_{\tau,\mathsf{H}}}$ until the query q_1^i to the random oracle and observe the output $W_{q_1^i} := \mathsf{H}(z_{q_1^i})$. Record the input/output pair $(z_{q_1^i}, W_{q_1^i})$ of this query.
(4) For the query q_2^i, we predict that the output of query $z_{q_2^i}$ is equal to $W_{q_1^i}$ *without actually* querying the random oracle, i.e., output the input/output pair prediction $(z_{q_2^i}, W_{q_1^i})$.

During each iteration i, we output a correct input/output pair with probability at least $2\varepsilon_{F,\tau,\mathsf{H}}/\widetilde{q}_{\mathsf{on}}^2$. Thus, the procedure above correctly outputs t input/output pairs with probability at least $(2\varepsilon_{F,\tau,\mathsf{H}}/\widetilde{q}_{\mathsf{on}}^2)^t$. Similar to Claim 5, we define $\mathsf{SucceedExp}$ as an event that the experiment above correctly outputs t input/output pairs (without querying the random oracle at these t points) when τ and H are picked randomly. Now let $\varepsilon_F := \mathbb{E}_{\tau,\mathsf{H}}[\varepsilon_{F,\tau,\mathsf{H}}]$ and to derive a contradiction, suppose that $\varepsilon_F > 4\widetilde{q}_{\mathsf{on}}^2/2^{2k}$. Then Markov inequality tells us that

$$\Pr_{\tau,\mathsf{H}}\left[\varepsilon_{F,\tau,\mathsf{H}} > \frac{2\widetilde{q}_{\mathsf{on}}^2}{2^{2k}}\right] > \frac{\varepsilon_F}{2}.$$

Thus, if we pick τ and H randomly and run the above procedure, we will succeed with probability at least

$$\Pr[\mathsf{SucceedExp}] \geq \frac{\varepsilon_F}{2}\left(\frac{2\varepsilon_{F,\tau,\mathsf{H}}}{\widetilde{q}_{\mathsf{on}}^2}\right)^t > 2\widetilde{q}_{\mathsf{on}}^2 2^{-2k} 2^{(2-2k)t}.$$

Similarly, since we have $\Pr[\mathsf{SucceedExp}] \leq 2^{-2kt+S}$ from Lemma 4, picking $t = (S+2k)/2$ we have a contradiction as we have

$$2\widetilde{q}_{\mathsf{on}}^2 2^{-kS-2k^2+S} < \Pr[\mathsf{SucceedExp}] \leq 2^{-kS-2k^2+S}.$$

Hence, we finally have

$$\begin{aligned}
\varepsilon'_F &:= \Pr[\mathsf{Collision}_{\mathsf{H}'}] \\
&= \mathbb{E}_{\tau,\mathsf{H}}[\varepsilon'_{F,\tau,\mathsf{H}}] \\
&\leq \mathbb{E}_{\tau,\mathsf{H}}\left[\varepsilon_{F,\tau,\mathsf{H}} + \frac{N^2(S+2k)}{2p}\right] \\
&= \varepsilon_F + \frac{N^2(S+2k)}{2p} \leq \frac{4\widetilde{q}_{\mathsf{on}}^2}{2^{2k}} + \frac{N^2(S+2k)}{2p}.
\end{aligned}$$

□

Reminder of Theorem 11. *The (short) Katz-Wang signature scheme is* $\left(N, q_{\mathsf{H}}, q_{\mathfrak{G}}, q_{\mathsf{S}}, \mathcal{O}\left(\frac{q+N}{2^k}\right)\right)$-MU-UF-CMA *secure under the generic group model of prime order* $p \approx 2^{2k}$ *and the programmable random oracle model, where* q *denotes the total number of queries made by an adversary.*

Proof Sketch of Theorem 11: Let Π be the Katz-Wang signature scheme. We follow essentially the same security reduction as in Theorem 6 using the signing oracle in Figure 3 (bottom). One minor difference here is that after $\mathcal{A}_{\mathsf{sig}}$ outputs $\sigma_{i*} = (s_{i*}, e_{i*})$ and m_{i*} for some $i* \in [N]$, we first compute $\mathfrak{a}_{j,i*} = \mathtt{Mult}(\mathtt{Pow}(\tau(p_j), s_{i*}), \mathtt{Pow}(\mathtt{Inv}(\mathfrak{h}_j), e_{i*}))$ for $j = 1, 2$, and check to see if we previously had any tuple of the form $(\mathfrak{a}_{j,i*}, \vec{a}_j, b_j) \in \mathcal{L}$ for any of $j = 1, 2$. If no such tuple exists, then we return $\perp$, and otherwise, we let $\vec{a}, b$ be given such that $\mathfrak{a}_{j,i*} = \tau(\vec{a}_j \cdot \vec{x} + b_j)$ for some j (or both). If $\vec{a}_j + e_{i*}\hat{u}_{i*} = \vec{0}$, then we return $\perp$, and otherwise, we have a BRIDGE^N instance $(\tau(s_{i*} - b_j), \vec{a}_j + e_{i*}\hat{u}_{i*}, 0) \in \mathcal{L}$ and $(\tau(s_{i*} - b_j), \vec{0}, s_{i*} - b_j) \in \mathcal{L}$. Then we have the following possibilities to output $\perp$ for failure correspondingly:

(1) One way to output failure is during the signing oracle, if $\mathsf{H}(\mathsf{pk}_j \| \mathfrak{a}_{1,i} \| \mathfrak{a}_{2,i} \| m_i)$ has been previously queried during the online phase. We define this event as $\mathsf{FailtoSign}$. Since pk_j is fixed before attacker runs, we cannot view pk_j as random. Note that every time the attacker queries the signing oracle, we would generate a new query to the random oracle H. Thus, we would have at most $q_{\mathsf{H}} + q_{\mathsf{S}}$ input/output pairs recorded for the random oracle. Since there are p possible choices for $\mathfrak{a}_{1,i}$, the probability that $(\mathsf{pk}_j \| \mathfrak{a}_{1,i} \| \mathfrak{a}_{2,i} \| m_i)$ is one of the inputs is at most $(q_{\mathsf{H}} + q_{\mathsf{S}})/p$. Applying union bound over q_{S} queries to the signing oracle, we conclude that $\Pr[\mathsf{FailtoSign}] \leq q_{\mathsf{S}}(q_{\mathsf{H}} + q_{\mathsf{S}})/p$.

(2) The second way to output failure is if *both* the value $\mathfrak{a}_{1,i*}$ and $\mathfrak{a}_{2,i*}$ does not previously appear in the list $\mathcal{L}$. We remark that if either value appears in the list, then we can still generate the BRIDGE^N event. We define this event by $\mathsf{FailtoFind}(\mathfrak{a}_{1,i*}, \mathfrak{a}_{2,i*})$. Since we already know that for a single instance, $\Pr[\mathsf{FailtoFind}(\mathfrak{a}_{j,i*})] \leq \frac{q_\mathsf{H}+q_\mathsf{S}}{p-|\mathcal{L}|} + 2^{-k}$ holds for each $j = 1, 2$ (i.e., see Claim 2), we have that $\Pr[\mathsf{FailtoFind}(\mathfrak{a}_{1,i*}, \mathfrak{a}_{2,i*})] \leq \frac{q_\mathsf{H}+q_\mathsf{S}}{p-|\mathcal{L}|} + \frac{1}{2^k}$, because of the observation that $\Pr[A \cap B] \leq \Pr[A]$.

(3) Finally, we could output failure if $\vec{a}_j = -e_{i*}\hat{u}_{i*}$ for both $j = 1$ and $j = 2$. We call this event $\mathsf{BadQuery}$. From Claim 3, we know that the probability to have a bad query for a single instance $\mathfrak{a}_{j,i*}$ is upper bounded by $q_\mathsf{H}/2^k$. Hence, from the same observation from (2), the probability to have bad queries for both instances is also upper bounded by $q_\mathsf{H}/2^k$. Thus, $\Pr[\mathsf{BadQuery}] \leq q_\mathsf{H}/2^k$.

Since we have $|\mathcal{L}| \leq N + 3q_\mathtt{G} + 1$, from Theorem 5 we conclude that

$$\begin{aligned}
&\Pr[\mathsf{SigForge}^{\tau,N}_{\mathcal{A}_{\mathsf{sig}},\Pi}(k) = 1] \\
&\leq \frac{q_\mathtt{G}N + 3q_\mathtt{G}(q_\mathtt{G}+1)/2}{p-(N+3q_\mathtt{G}+1)^2-N} + \frac{Nq_\mathsf{S}(q_\mathsf{H}+q_\mathsf{S})}{p^2} + \frac{q_\mathsf{H}+q_\mathsf{S}}{p-(N+3q_\mathtt{G}+1)} + \frac{q_\mathsf{H}+1}{2^k} \\
&= \mathcal{O}\left(\frac{q+N}{2^k}\right).
\end{aligned}$$

□

Reminder of Theorem 12. *Let $\Pi = (\mathsf{Kg}, \mathsf{Sign}, \mathsf{Vfy})$ be a Katz-Wang signature scheme and $p > 2^{2k}$ be a prime number. Let $N \in \mathbb{N}$ be a parameter and $(\mathcal{A}^{\mathsf{pre}}_{\mathsf{sig}}, \mathcal{A}^{\mathsf{on}}_{\mathsf{sig}})$ be a pair of generic algorithms with an encoding map $\tau : \mathbb{Z}_p \to \mathbb{G}$ such that $\mathcal{A}^{\mathsf{pre}}_{\mathsf{sig}}$ makes at most $q^{\mathsf{pre}}_\mathsf{H} < 2^{3k}$ queries to the random oracle at most $q^{\mathsf{pre}}_\mathsf{H} < 2^{3k}$ queries to the random oracle $\mathsf{H} : \{0,1\}^* \to \{0,1\}^{k_1}$ and outputs an S-bit hint $\mathsf{str}_{\tau,\mathsf{H}}$, and $\mathcal{A}^{\mathsf{on}}_{\mathsf{sig}}$ makes at most $q^{\mathsf{on}}_\mathtt{G} := q^{\mathsf{on}}_\mathtt{G}(k)$ queries to the generic group oracles and at most $q^{\mathsf{on}}_\mathsf{H}$ queries to the random oracle. Then* $\Pr\left[\mathsf{SigForge}^{\tau,N}_{\mathcal{A}^{\mathsf{on}}_{\mathsf{sig}},\mathsf{str}_{\tau,\mathsf{H}},\Pi}(k) = 1\right] \leq \varepsilon$, *with*

$$\varepsilon = \widetilde{\mathcal{O}}\left(\frac{SN(q^{\mathsf{on}}_\mathtt{G}+N)(q^{\mathsf{on}}_\mathtt{G}+2N)}{p}\right) + \frac{Nq^{\mathsf{pre}}_\mathsf{H}q^{\mathsf{on}}_\mathsf{S}}{p^2} + \frac{q^{\mathsf{on}}_\mathsf{S}(q^{\mathsf{on}}_\mathsf{S}+q^{\mathsf{on}}_\mathsf{H})}{p} + \frac{4(q^{\mathsf{on}}_\mathsf{H}+1)}{2^{k_1}} + \frac{N^2(S+k_1)}{p},$$

where $q^{\mathsf{on}}_\mathsf{S}$ denotes the number of queries to the signing oracle and the randomness is taken over the selection of τ and the random coins of $\mathcal{A}^{\mathsf{on}}_{\mathsf{sig}}$ (the hint $\mathsf{str}_{\tau,\mathsf{H}} = \mathcal{A}^{\mathsf{pre},\mathtt{GD}}_{\mathsf{sig}}(\mathfrak{g})$ is selected independently of the random coins used by the challenger).

Proof Sketch of Theorem 12: We follow essentially the same security reduction as in Theorem 8, except for the differences as follows:

- When the attacker tries to forge a signature on the message m_i without knowledge of the secret key x_j, s/he follow Figure 3 (bottom).
- After $\mathcal{A}_{\mathsf{sig}}$ outputs $\sigma_{i*} = (s_{i*}, e_{i*})$ and m_{i*} for some $i* \in [N]$, we first compute $\mathfrak{a}_{j,i*} = \mathtt{Mult}(\mathtt{Pow}(\tau(p_j), s_{i*}), \mathtt{Pow}(\mathtt{Inv}(\mathfrak{h}_j), e_{i*}))$ for $j = 1, 2$, and check

to see if we previously had any tuple of the form $(\mathfrak{a}_{j,i*}, \vec{a}_j, b_j) \in \mathcal{L}$ for any of $j = 1, 2$. If no such tuple exists, then we return $\perp$, and otherwise, we let $\vec{a}, b$ be given such that $\mathfrak{a}_{j,i*} = \tau(\vec{a}_j \cdot \vec{x} + b_j)$ for some j (or both). If $\vec{a}_j + e_{i*}\hat{u}_{i*} = \vec{0}$, then we return $\perp$, and otherwise, we have a BRIDGE^N instance $(\tau(s_{i*} - b_j), \vec{a}_j + e_{i*}\hat{u}_{i*}, 0) \in \mathcal{L}$ and $(\tau(s_{i*} - b_j), \vec{0}, s_{i*} - b_j) \in \mathcal{L}$.

Then we have the following possibilities to output $\perp$ for failure correspondingly. Everything goes the same with the analysis in Theorem 8 except for the following:

(1) Let $B_{i,j}$ denote the bad event that the preprocessing attacker previously submitted a query of the form $\mathsf{H}(\mathsf{pk}_j \| \mathfrak{a}_{1,i} \| \mathfrak{a}_{2,i} \| m_i)$. Since we pick pk_j randomly, and as well as $\mathfrak{a}_{1,i}$, we have $\Pr[B_{i,j}] \leq q_{\mathsf{H}}^{\mathsf{pre}}/p^2$. We now define the event $\mathsf{DetectProgramming} = \bigcup_{j \leq N, i \leq q_{\mathsf{S}}} B_{i,j}$. Applying union bounds, we have

$$\Pr[\mathsf{DetectProgramming}] = \Pr[\cup_{i,j} B_{i,j}] \leq \sum_{i,j} \Pr[B_{i,j}] \leq \frac{N q_{\mathsf{H}}^{\mathsf{pre}} q_{\mathsf{S}}^{\mathsf{on}}}{p^2}.$$

Assuming that the event $\mathsf{DetectProgramming}$ does not occur, the attacker will not be able to distinguish between a programmed response and the original response.

(2) One way to output failure is during the signing oracle if $\mathsf{H}(\mathsf{pk}_j \| \mathfrak{a}_{1,i} \| \mathfrak{a}_{2,i} \| m_i)$ has been previously queried during the online phase. We define this event as $\mathsf{FailtoSign}$. As before, we can argue that $\Pr[\mathsf{FailtoSign}] \leq \frac{q_{\mathsf{S}}^{\mathsf{on}}(q_{\mathsf{S}}^{\mathsf{on}} + q_{\mathsf{H}}^{\mathsf{on}})}{p}$, i.e., see Theorem 11.

(3) Another way to output $\perp$ is if the event $\mathsf{DetectProgramming}$ did not occur and both the value $\mathfrak{a}_{1,i*}$ and $\mathfrak{a}_{2,i*}$ does not previously appear in the list $\mathcal{L}$. We call this event $\mathsf{FailtoFind}(\mathfrak{a}_{1,i*}, \mathfrak{a}_{2,i*})$. The final case to output failure is if both $\vec{a}_j = -e_{i*}\hat{u}_{i*}$ holds for $j = 1$ and $j = 2$, and we call this event $\mathsf{BadQuery}$. Similar to the argument from Theorem 11, each probability is bounded by the event with the single instance. Hence, we can directly adapt the same argument from Theorem 8 and say that

$$\Pr[\mathsf{FailtoFind}(\mathfrak{a}_{1,i*}, \mathfrak{a}_{2,i*}) \cup \mathsf{BadQuery}] \leq \frac{4(q_{\mathsf{H}}^{\mathsf{on}} + 1)}{2^{k_1}} + \frac{N^2(S + k_1)}{p}.$$

Applying Theorem 7 and from the similar argument in Theorem 8, we have

$$\begin{aligned}\Pr\left[\mathsf{SigForge}^{\tau,N}_{\mathcal{A}^{\mathsf{on}}_{\mathsf{sig},\mathsf{str}_{\tau,\mathsf{H}}},\Pi}(k) = 1\right] &\leq \widetilde{\mathcal{O}}\left(\frac{SN(q_{\mathsf{G}}^{\mathsf{on}} + N)(q_{\mathsf{G}}^{\mathsf{on}} + 2N)}{p}\right) + \frac{N q_{\mathsf{H}}^{\mathsf{pre}} q_{\mathsf{S}}^{\mathsf{on}}}{p^2} + \frac{q_{\mathsf{S}}^{\mathsf{on}}(q_{\mathsf{S}}^{\mathsf{on}} + q_{\mathsf{H}}^{\mathsf{on}})}{p} \\ &\quad + \frac{4(q_{\mathsf{H}}^{\mathsf{on}} + 1)}{2^{k_1}} + \frac{N^2(S + k_1)}{p},\end{aligned}$$

which completes the proof. □

Remark 7. Similar to Theorem 8, if $q_{\mathsf{G}}^{\mathsf{on}} \geq 10N(1+2\log p)$ and $S \geq 10\log(8p)$, then we can argue that

$$\Pr\left[\mathsf{SigForge}^{\tau,N}_{\mathcal{A}^{\mathsf{on}}_{\mathsf{sig},\mathsf{str}_{\tau,\mathsf{H}}},\Pi}(k)=1\right] \leq \frac{12SN(q_{\mathsf{G}}^{\mathsf{on}})^2\log p}{p} + \frac{Nq_{\mathsf{H}}^{\mathsf{pre}}q_{\mathsf{S}}^{\mathsf{on}}}{p^2} + \frac{q_{\mathsf{S}}^{\mathsf{on}}(q_{\mathsf{S}}^{\mathsf{on}}+q_{\mathsf{H}}^{\mathsf{on}})}{p} + \frac{4(q_{\mathsf{H}}^{\mathsf{on}}+1)}{2^{k_1}} + \frac{N^2(S+k_1)}{p},$$

for the Katz-Wang signature scheme Π. ◁