Reusing Nonces in Schnorr Signatures
(and keeping it secure...)

Marc Beunardeau[1], Aisling Connolly[1], Houda Ferradi[2], Rémi Géraud[1], David Naccache[1], and Damien Vergnaud[1]

[1] Département d'informatique de l'ENS,
École normale supérieure, CNRS, PSL Research University
75005 Paris, France
given_name.family_name@ens.fr
[2] NTT Secure Platform Laboratories
3–9–11 Midori-cho, Musashino-shi, Tokyo 180–8585, Japan
ferradi.houda@lab.ntt.co.jp

Abstract. The provably secure Schnorr signature scheme is popular and efficient. However, each signature requires a fresh modular exponentiation, which is typically a costly operation. As the increased uptake in connected devices revives the interest in resource-constrained signature algorithms, we introduce a variant of Schnorr signatures that mutualises exponentiation efforts.
Combined with precomputation techniques (which would not yield as interesting results for the original Schnorr algorithm), we can amortise the cost of exponentiation over several signatures: these signatures share the same nonce. Sharing a nonce is a deadly blow to Schnorr signatures, but is not a security concern for our variant.
Our Scheme is provably secure, asymptotically-faster than Schnorr when combined with efficient precomputation techniques, and experimentally 2 to 6 times faster than Schnorr for the same number of signatures when using 1 MB of static storage.

1 Introduction

The increased popularity of lightweight implementations invigorates the interest in resource-preserving protocols. Interestingly, this line of research was popular in the late 1980's, when smart-cards started performing public-key cryptographic operations (e.g. [11]). Back then, cryptoprocessors were expensive and cumbersome, and the research community started looking for astute ways to identify and sign with scarce resources.

In this work we revisit a popular signature algorithm published by Schnorr in 1989 [24] and seek to lower its computational requirements assuming that the signer is permitted to maintain some read-only memory. This storage allows for time-memory trade-offs, which are usually not very profitable for typical Schnorr parameters.

We introduce a new signature scheme, which is provably secure in the random oracle model (ROM) under the assumption that the *partial discrete logarithm problem* (see below) is intractable. This scheme can benefit much more from precomputation techniques, which results in faster signatures.

Implementation results confirm the benefits of this approach when combining efficient precomputation techniques, when enough static memory is available (of the order of 250 couples of the form (x, g^x)). We provide comparisons with Schnorr for several parameters and pre-computation schemes.

1.1 Intuition and general outline of the idea

Schnorr's signature algorithm uses a large prime modulus p and a smaller prime modulus q dividing $p-1$. The security of the signature scheme relies on the discrete logarithm problem in a subgroup of order q of the multiplicative group of the finite field $\mathbb{Z}_p$ (with $q \mid p-1$). Usually the prime p is chosen to be large enough to resist index-calculus methods for solving the discrete-log problem (*e.g.* 3072 bits for a 128-bit security level), while q is large enough to resist the square-root algorithms [26] (e.g. 256 bits for 128-bit security level).

The intuition behind our construction is to consider a prime p such that $p-1$ has *several* different factors q_i large enough to resist these birthday attacks, i.e.

$$p = 1 + 2\prod_{i=1}^{\ell} q_i$$

then several "orthogonal" Schnorr signatures can share the same commitment component $r = g^k \bmod p$. This is not the case with standard Schnorr signatures where, if a k is reused then the secret signing key is revealed.

It remains to find how r can be computed quickly. In the original Schnorr protocol k is picked uniformly at random in $\mathbb{Z}_q$. However, to be secure, our construction requires that k is picked in the larger set $\mathbb{Z}_{p-1}$. This means that a much higher effort is required to compute r. Here we cut corners by generating an r with pre-computation techniques which allow an exponentiation to be sub-linear. The trick is that once the exponentiation is sub-linear, we are more effective in our setting than in the original Schnorr setting.

We start by reminding how the original Schnorr signature scheme works and explain how we extend it assuming that k is randomly drawn from $\mathbb{Z}_{p-1}$. We then present applications of our construction, by comparing several pre-processing schemes.

2 Preliminaries

We denote the security parameter by $\kappa \in \mathbb{N}$ which is given to all algorithms in the unary form 1^κ. Algorithms are randomized unless otherwise stated, and PPT stands for "probabilistic polynomial-time," in the security parameter. We denote random sampling from a finite set X according to the uniform distribution with

$x \twoheadleftarrow X$. We also use the symbol $\twoheadleftarrow$ for assignments from randomized algorithms, while we denote assignment from deterministic algorithms and calculations with the symbol $\leftarrow$. If n is an integer, we write $\mathbb{Z}_n$ for the ring $\mathbb{Z}/n\mathbb{Z}$. We let $\mathbb{Z}_n^*$ the invertible elements of $\mathbb{Z}_n$. As is usual, $f \in \text{negl}(\kappa)$ denotes a function that decreases faster than the inverse of any polynomial in κ; such functions are called negligible. The set of numbers $1, 2, \ldots, k$ is denoted $[k]$. Most of our security definitions and proofs use code-based games. A game G consists of an initializing procedure Init, one or more procedures to respond to oracle queries, and a finalizing procedure Fin.

2.1 Schnorr's Signature Scheme

Schnorr signatures [24] are an offspring ElGamal signatures [10] which are provably secure in the Random Oracle Model under the assumed hardness of solving generic instances of the Discrete Logarithm Problem (DLP) [22]. The Schnorr signature scheme is a tuple of algorithms defined as follows:

- $\mathsf{Setup}(1^\kappa)$: Large primes p, q are chosen, such that $q \geq 2^\kappa$ and $p-1 = 0 \bmod q$. A cyclic group $\mathbb{G} \subset \mathbb{Z}_p$ of prime order q is chosen, in which it is assumed that the DLP is hard, along with a generator $g \in \mathbb{G}$. A hash function $H : \{0,1\}^* \to \mathbb{G}$ is chosen. Public parameters are $\mathsf{pp} = (p, q, g, \mathbb{G}, H)$.
- $\mathsf{KeyGen}(\mathsf{pp})$: Pick an integer x uniformly at random from $[2, q-1]$ as the signing key sk, and publish $y \leftarrow g^x$ as the public key pk.
- $\mathsf{Sign}(\mathsf{pp}, \mathsf{sk}, m)$: Pick k uniformly at random in $\mathbb{Z}_q^*$, compute $r \leftarrow g^k \bmod q$, $e \leftarrow H(m, r)$, and $s \leftarrow k - ex \bmod p$. Output $\sigma \leftarrow \{r, s\}$ as a signature.
- $\mathsf{Verify}(\mathsf{pp}, \mathsf{pk}, m, \sigma)$: Let $(r, s) \leftarrow \sigma$, compute $e \leftarrow H(m, r)$ and return True if $g^s y^e = r$, and False otherwise.

2.2 Security model

We recall the strong[3] EUF-CMA security notion:

Definition 1 (Strong EUF-CMA Security). *A signature scheme Σ is* secure against existential forgeries in a chosen-message attack *(strongly* EUF-CMA*-secure) if the advantage of any PPT adversary $\mathcal{A}$ against the* EUF-CMA *game defined in Figure 1 is negligible:* $\mathsf{Adv}^{\mathsf{EUF}}_{\mathcal{A},\Sigma}(\kappa) = \Pr\left[\mathsf{EUF}^{\mathcal{A}}_{\Sigma}(\kappa) = 1\right] \in \text{negl}(\kappa)$.

3 Our Scheme: Using Multiple q's

Our construction relies on using a prime p of the form mentioned in the introduction. This is not a trivial change, and requires care as we discuss below.

[3] In contrast to the *weak* version, the adversary is allowed to forge for a message that they have queried before, provided that their forgery is *not* an oracle response.

$\underline{\mathsf{EUF}^{\mathcal{A}}_{\Sigma}(\kappa)}$:
$L \leftarrow \emptyset$
$(\mathsf{sk}, \mathsf{pk}) \twoheadleftarrow \Sigma.\mathsf{KeyGen}(1^\kappa)$
$(m^*, \sigma^*) \leftarrow \mathcal{A}^{\mathrm{Sign}(\cdot),\mathrm{Verify}(\cdot,\cdot),H(\cdot)}(1^\kappa)$
if $(m^*, \sigma^*) \notin L$
    return $\Sigma.\mathsf{Verify}(\mathsf{pk}, m^*)$
return 0

$\underline{\mathrm{Sign}(m)}$:
$\sigma \twoheadleftarrow \Sigma.\mathsf{Sign}(\mathsf{sk}, m)$
$L \leftarrow L \cup \{m, \sigma\}$
return σ

$\underline{\mathrm{Verify}(m, \sigma)}$:
return $\Sigma.\mathsf{Verify}(\mathsf{pk}, m, \sigma)$

Fig. 1: The strong EUF-CMA experiment for digital signature schemes.

Technically, our construction is a *stateful* signature scheme (see e.g. [15, Chapter 12]), in which we simultaneously sign only one message and keep a state corresponding to the values k, g^k and the index i for the current prime number. However, it is more compact and convenient to describe it as a signature for ℓ simultaneous messages.

3.1 Our Signature Scheme

Similar to the Schnorr signature scheme, our scheme is a tuple of algorithms (Setup, KeyGen, Sign, and Verify), which we define as follows:

- $\mathsf{Setup}(1^\kappa)$: Generate ℓ primes $q_1, \ldots, q_\ell$ of size $\geq 2^\kappa$ and ℓ groups $\mathbb{G}_1, \ldots, \mathbb{G}_\ell$ respectively of order $q_1, \ldots q_\ell$ such that the DLP is hard in the respective $\mathbb{G}_i$, and such that $p = 1 + 2\prod q_i$ is prime. This is easily achieved by selecting $(\ell - 1)$ safe primes q_i and varying the last one until p is prime.[4] Choose a cryptographic hash function $H : \{0,1\}^* \to \{0,1\}^{q_1}$. The hash function will be used to produce elements of $\mathbb{Z}_{q_i}$. For this we will denote by H_i the composition of H and a conversion function from $\{0,1\}^{q_1}$ to $\mathbb{Z}_{q_i}$[5]. Finally, choose g a generator of the group $\mathbb{Z}_p^*$ of order $p - 1$. The public parameters are therefore
$$\mathsf{pp} = \left(p, \{q_i\}_{i=1}^{\ell}, H, g, \{\mathbb{G}_i\}_{i=1}^{\ell}\right).$$
- $\mathsf{KeyGen}(\mathsf{pp})$: The signer chooses $x \twoheadleftarrow \mathbb{Z}_{p-1}^*$ and computes $y \leftarrow g^x \bmod p$. The key $\mathsf{sk} = x$ is kept private to the signer, while the verification key $\mathsf{pk} = y$ is made public.
- $\mathsf{Sign}(\mathsf{pp}, \mathsf{sk}, m_1, \ldots, m_\ell)$: The signer chooses $k \twoheadleftarrow \mathbb{Z}_p$, such that $k \neq 0 \bmod q_i$ for all i, and computes $r \leftarrow g^k \bmod p$.
  The signer can now sign the ℓ messages m_i as:
$$\rho_i \twoheadleftarrow \{0,1\}^\kappa, \quad e_i \leftarrow H_i(m_i, r, \rho_i), \quad \text{and} \quad s_i \leftarrow k - e_i x \bmod q_i$$

[4] See the full version of this paper for a discussion on some particularly interesting moduli.

[5] This conversion function can read the string as a binary number and reduce it $\bmod q_i$ for example.

outputting the ℓ signatures $\sigma_i = \{r, s_i, \rho_i\}$—or, in a more compact form[6],

$$\sigma = \{r, s_1, \ldots, s_\ell, \rho_1, \ldots, \rho_\ell\}.$$

– $\mathsf{Verify}(\mathsf{pp}, \mathsf{pk}, m_i, (r, s_i, \rho_i), i)$: Verifying a signature is achieved by slightly modifying the original Schnorr scheme: First check that $s_i \in \{0, \ldots q_i - 1\}$ and compute $e_i \leftarrow H_i(m_i, r, \rho_i)$, then observe that for a correct signature[7]:

$$(g^{s_i} y^{e_i})^{\frac{p-1}{q_i}} = r^{\frac{p-1}{q_i}} \bmod p.$$

The signature is valid if and only if this equality holds, otherwise the signature is invalid (see Lemma 1).

Remark 1. Note that unlike Schnorr, in the Sign algorithm we add a random ρ_i for a signature to make the argument of the hash function unpredictable. This will be useful for the proof of Theorem 1 in the ROM.

Remark 2. Note also that one almost recovers the original Schnorr construction for $\ell = 1$—the only differences being in the verification formula, where both sides are squared in our version, and the addition of a fresh random to hash.

Lemma 1 (Correctness). *Our signature scheme is correct.*

Proof. Let g, y, r, s_i, and ρ_i be as generated by the KeyGen and Sign algorithms for a given message m_i. We check that,

$$\left(\frac{g^{s_i} y^{e_i}}{r}\right)^{\frac{p-1}{q_i}} = 1 \bmod p.$$

By the definition of s_i, there exists $\lambda \in \mathbb{Z}$ such that $g^{s_i} = g^{k - e_i x + \lambda q_i}$, hence

$$g^{s_i} y^{e_i} g^{-k} = g^{\lambda q_i} \bmod p.$$

Raising this to the power of $\frac{p-1}{q_i}$ we get $g^{\lambda(p-1)} = 1$ since the order the multiplicative group $\mathbb{Z}_p^*$ is $p - 1$. □

3.2 Security

To aid in the proof of security, we introduce the following problem which we call the partial discrete logarithm problem (PDLP). Intuitively it corresponds to solving a discrete logarithm problem in the subgroup of our choice.

[6] The compact form allows not to send the nonce ℓ times, which gives an "amortized" size of the signature, and avoid an overhead in communication.

[7] One can note, $\frac{p-1}{q_i} = 2q_1 \cdots q_{i-1} q_{i+1} \cdots q_\ell$.

Definition 2 (PDLP). *Let $\ell \geq 2$ be an integer, $q_1, \ldots, q_\ell$ distinct prime numbers and $q = q_1 \ldots q_\ell$. Let $\mathbb{G}$ be a group of order q and g a generator of $\mathbb{G}$. Given $g, q, q_1, \ldots, q_l$, and $y = g^x$, the* partial discrete logarithm problem *(PDLP) consists in finding $i \in [\ell]$ and $x_i \in \mathbb{Z}_{q_i}$ such that $x_i = x \bmod q_i$.*

In our context, we are chiefly interested in a subgroup of order q of a multiplicative group of a finite field $\mathbb{Z}_p^*$, where q divides $p-1$—ideally, $q = (p-1)/2$. The best known algorithms to solve the PDLP are index-calculus based methods in $\mathbb{Z}_p^*$ and square-root algorithms in subgroups of prime order q_i for some $i \in [\ell]$. With p of bit-size 3072, $q = (p-1)/2$, $\ell = 12$ and $q_1, \ldots, q_\ell$ of bit-size 256, we conjecture that solving the PDLP requires about 2^{128} elementary operations. In the full version of this paper, we provide a security argument in the generic group model on the intractability of the PDLP for large enough prime numbers $q_1, \ldots, q_\ell$.

Theorem 1 (Existential unforgeability). *Our scheme is provably* EUF-CMA*-secure assuming the hardness of solving the PDLP, in the ROM.*

To prove this result, we will exhibit a reduction from an efficient EUF-CMA forger to an efficient PDLP solver. To that end we first show a sequence of indistinguishability results between the output distributions of

- Our signature algorithm $\mathsf{Sign} = \mathsf{Sign}_0$ on user inputs.
- A modified algorithm Sign_1 (see Figure 2), where the hash of user inputs is replaced by a random value. This situation is computationally indistinguishable from the previous one in the ROM.
- A modified algorithm Sign_2 (see Figure 2), that has no access to the signing key x. The output distribution of this algorithm is identical to the output of Sign_1 (Theorem 2).

Then we use the forking lemma [3, 23] to show that an efficient EUF-CMA-adversary against Sign_2 can be used to construct an efficient PDLP solver. Finally we leverage the above series of indistinguishably results to use an adversary against Sign_0. Let CRT (for Chinese Remainder Theorem) be the isomorphism that maps $\mathbb{Z}_{q_1} \times \cdots \times \mathbb{Z}_{q_\ell} \times \mathbb{Z}_2$ to $\mathbb{Z}_{p-1}$.

Theorem 2. *The output distributions of Sign_1 and Sign_2 are identical.*

Proof. This theorem builds on several intermediate results described in Lemmas 2 to 6. We denote δ the output distribution of Sign_1 and δ' the output distribution of Sign_2. The structure of the proof is the following :

- In Lemma 2 we show that the output of Sign_2 is a subset of the output of Sign_1.
- Lemma 3 shows that in Sign_1 there is a unique random tape per output.
- Lemma 4 shows that in Sign_2 there are exactly two random tapes per output.
- Lemma 6 shows that there are twice as many random tapes possible for Sign_2 than for Sign_1

$\underline{\mathsf{Sign}_1:}$
$\rho \twoheadleftarrow \{0,1\}^\kappa$
$k \twoheadleftarrow \mathbb{Z}_p \setminus \left(\bigcup_{i=1}^{\ell} \{q_i, 2q_i, \ldots, p-1\}\right)$
$r \leftarrow g^k \bmod p$
for $i = 1$ to ℓ
    $e_i \twoheadleftarrow \mathbb{Z}_{q_i}$
    $s_i \leftarrow k - e_i x \bmod q_i$
    $\rho_i \twoheadleftarrow \{0,1\}^\kappa$
end for
return $(r, e_1, \ldots, e_\ell, s_1, \ldots, s_\ell, \rho_1 \ldots, \rho_\ell)$

$\underline{\mathsf{Sign}_2:}$
for $i = 1$ to ℓ
    $e_i \twoheadleftarrow \mathbb{Z}_{q_i}$
    $s_i \twoheadleftarrow \mathbb{Z}_{q_i}$
    $\rho_i \twoheadleftarrow \{0,1\}^\kappa$
end for
$a \twoheadleftarrow \{0,1\}$
$b \twoheadleftarrow \{0,1\}$
$S \leftarrow \mathsf{CRT}(s_1, \ldots, s_\ell, a)$
$E \leftarrow \mathsf{CRT}(e_1, \ldots, e_\ell, b)$
$r \leftarrow g^S y^E$
for $i = 1$ to ℓ
    check that $r \neq 1 \bmod q_i$,
      otherwise abort
end for
return $(r, e_1, \ldots, e_\ell, s_1, \ldots, s_\ell, \rho_1 \ldots, \rho_\ell)$

Fig. 2: The algorithms used in Theorem 2, as part of the proof of Theorem 1.

This demonstrates that by uniformly choosing the random tape, the resulting distributions for Sign_1 and Sign_2 are identical, which is the uniform distribution on the set of valid signatures.

Lemma 2. *Every tuple of δ' is a valid signature tuple. Therefore $\delta' \subseteq \delta$.*

Proof (of Lemma 2). Let $(r, e_1, \ldots, e_\ell, s_1, \ldots, s_\ell, \rho_1, \ldots, \rho_\ell) \in \delta'$. Let $i \in [\ell]$. By the Chinese Remainder Theorem we have:

$$S = s_i \bmod q_i \quad \text{and} \quad E = e_i \bmod q_i.$$

So there exists $\lambda, \mu \in \mathbb{Z}$ such that

$$S = s_i + \lambda q_i \quad \text{and} \quad E = e_i + \mu q_i.$$

Hence:

$$\begin{aligned} r^{\frac{p-1}{q_i}} &= \left(g^S y^E\right)^{\frac{p-1}{q_i}} \\ &= \left(g^{s_i + \lambda q_i} y^{e_i + \mu q_i}\right)^{\frac{p-1}{q_i}} \\ &= \left(g^{s_i} y^{e_i}\right)^{\frac{p-1}{q_i}} g^{\lambda(p-1)} y^{\mu(p-1)} \\ &= \left(g^{s_i} y^{e_i}\right)^{\frac{p-1}{q_i}} \end{aligned}$$

The last equality holds since the order of the multiplicative group $\mathbb{Z}_p^*$ is $p-1$, and this concludes the proof with the fact that $r \neq 1 \bmod q_i$. □

Lemma 3. *There is exactly one random tape upon which Sign_1 can run to yield each particular tuple of δ.*

Proof (of Lemma 3). Let $k, e_1, \ldots, e_\ell, \rho_1, \ldots, \rho_\ell$ and $k', e'_1, \ldots, e'_\ell, \rho'_1, \ldots, \rho'_\ell$ be random choices of δ that both yield $(r, e_1, \ldots, e_\ell, s_1, \ldots, s_\ell, \rho_1, \ldots, \rho_\ell)$. It is immediate that $e_i = e'_i$ and $\rho_i = \rho'_i$ for all $i \in [\ell]$. Also since $g^k = g^{k'}$, g is of order $p-1$ and since k and k' are in $[p]$ then $k = k'$. □

Lemma 4. *There are exactly two random tapes over k, $\rho_1, \ldots, \rho_\ell$, $e_1, \ldots, e_\ell$ that output each tuple of δ'.*

Proof (of Lemma 4). Let $e_1, \ldots, e_\ell$, $s_1, \ldots, s_\ell$, a, b, $\rho_1, \ldots, \rho_\ell$ and $e'_1, \ldots, e'_\ell$, $s'_1, \ldots, s'_\ell$, a', b', $\rho'_1, \ldots, \rho'_\ell$ be random choices that both give $(r, e_1, \ldots, e_\ell, s_1, \ldots, s_\ell, \rho_1, \ldots, \rho_\ell)$. It is immediate that $e_i = e'_i$, $s_i = s'_i$, and $\rho_i = \rho'_i$ for all $i \in [\ell]$. Let S, S', E, and E' be the corresponding CRT images. We have $g^S y^E = g^{S'} y^{E'}$, which is $g^{S+xE} = g^{S'+xE'}$, and $S + xE = S' + xE' \bmod (p-1)$. Since x is odd (it is invertible mod $p-1$), it follows that $S+E$ and $S'+E'$ have the same parity. Therefore $a + b = a' + b' \bmod 2$ and we have two choices: $a = b$, or $a = 1 - b$, both of which are correct. □

Lemma 5. $\#\left(\mathbb{Z}_p \setminus \left(\bigcup_{i=1}^{\ell} \{q_i, 2q_i, \ldots, p-1\}\right)\right) = 2\prod_{i=1}^{\ell}(q_i - 1)$.

Proof (of Lemma 5). The number of invertible elements $\bmod p$ is $\prod_{i=1}^{\ell}(q_i - 1) \times (2-1)$ so the number of invertible mod q_i for all i (and not necessarily for 2) is $2\prod_{i=1}^{\ell}(q_i - 1)$. This is exactly the cardinality of the set

$$\left(\mathbb{Z}_p \setminus \left(\bigcup_{i=1}^{\ell} \{q_i, 2q_i, \ldots, p-1\}\right)\right).$$

□

Lemma 6. *There are twice as many possible random choices in δ' as in δ.*

Proof (of Lemma 6). For the number of random choices in δ we use Lemma 5 to count the number of k and then count the number of e_i and get $2\prod_{i=1}^{\ell}(q_i - 1) \times \prod_{i=1}^{\ell} q_i$. For δ', having $r \neq 1 \bmod q_i$ is equivalent to having $s_i \neq -e_i x$. Therefore it has the same number of random choices as a distribution picking the s_i from $\mathbb{Z}_{q_i} \setminus \{e_i x\}$ which is $\prod_{i=1}^{\ell} q_i \times \prod_{i=1}^{\ell}(q_i - 1) \times 2 \times 2$. □

It follows from the above results that the two distributions are the same, i.e. the uniform distribution over the set of valid signatures.
This concludes the proof of Theorem 2. □

Theorem 3 (Security under Chosen Message Attack). *An efficient attacker against Sign_2 can be turned into an efficient PDLP solver in the ROM.*

Proof. Let $\mathcal{A}$ be an attacker that wins the $\mathsf{EUF\text{-}CMA}$ game for our scheme, illustrated in Figure 3. We construct in Figures 4 and 5 an algorithm $\mathcal{R}$ that uses $\mathcal{A}$ to solve the PDLP. $\mathcal{A}'$ is equivalent to $\mathcal{A}$ (with the same random tape which we omit in the notation), the difference being that it interacts with different oracles.

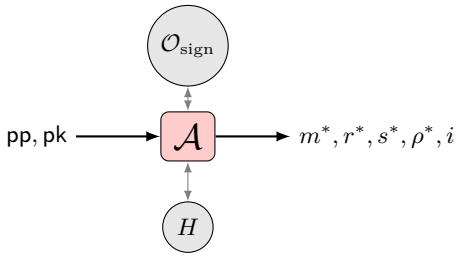

Fig. 3: An efficient EUF-CMA adversary $\mathcal{A}$ against our scheme, with random oracle H and a signing oracle $\mathcal{O}$.

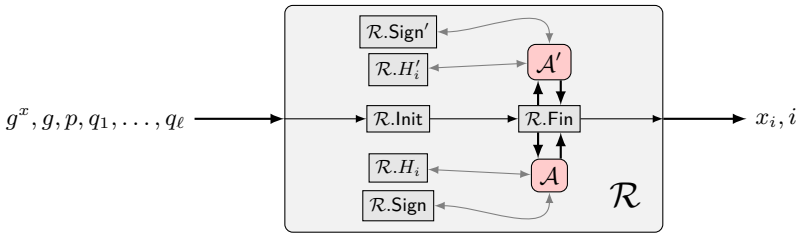

Fig. 4: An efficient solver $\mathcal{R}$ for the PDLP, using a polynomial number of queries to $\mathcal{A}$. $\mathcal{R}$ implements the random oracle as $\mathcal{R}.H$ and the signing oracle as $\mathcal{R}.\text{Sign}$. The rewinded adversary and oracles are indicated with a prime symbol.

Abusing notation we denote by $\mathcal{R}.H_i$ the composition of the hash function and the conversion function. If L is a list of pairs, we denote by $L^{-1}[e]$ the index of the element e in the list, and by $L[i]$ the i-th element of the list. If they cannot (i.e. if e is not in the list, or the list does not have an i-th element) they abort.

The algorithm $\mathcal{R}$ aborts in four possible ways during the simulation (denoted ($\star$), (†), (‡) and (§)) in Figures 4 and 5. We upper-bound the probability of these events in the following list:

- ($\star$) This occurs with negligible probability since the ρ is a fresh random which is unpredictable by the adversary.
- (†) This occurs with non overwhelming probability since the adversary is efficient.
- (‡) The element is in the list with non negligible probability because if the adversary forges on an unqueried hash in the ROM, it has a negligible chance to succeed.
- (§) This happens with non overwhelming probability due to the forking lemma [23].

If $\mathcal{R}$ does not abort, then $\left(g^{s^*} y^{e^*}\right)^{\frac{p-1}{q_{i^*}}} = (r^*)^{\frac{p-1}{q_{i^*}}} = \left(g^{\tilde{s}^*} y^{\tilde{e}^*}\right)^{\frac{p-1}{q_{i^*}}} \bmod p$. Then $s^* + e^* x = \tilde{s}^* + \tilde{e}^* \bmod q_{i^*}$. It follows that the value returned by $\mathcal{R}$ is equal to $x \bmod q_{i^*}$.
$\mathcal{R}$ succeeds with non negligible probability, as explained earlier. The probability of forking is polynomial in the number of queries to the random oracle, the

$\mathcal{R}.\mathsf{Init}(y = g^x, g, p, q_1, \ldots, q_\ell)$:
  $L \leftarrow \emptyset$
  $L' \leftarrow \emptyset$
  $\Sigma \leftarrow \emptyset$
  $j \leftarrow 1$
  $k \leftarrow 0$
  $l \leftarrow 0$
  $\mathsf{pk} \leftarrow y$
  $\mathsf{pp} \leftarrow \{p, \{q_i\}_{i=1}^{\ell}, g\}$
  return $(\mathsf{pk}, \mathsf{pp})$

$\mathcal{R}.\mathsf{Fin}(\mathsf{pk}, \mathsf{pp})$:
  $(m^*, r^*, s^*, \rho^*, i^*) \twoheadleftarrow \mathcal{A}(\mathsf{pp}, \mathsf{pk})$
  $e^* \leftarrow \mathcal{R}.H_{i^*}(m^*, r^* \bmod q_{i^*}, \rho^*)$
  $a \leftarrow L^{-1}[((m^*, r^* \bmod q_{i^*}, \rho^*), e^*)]$‡
  if not $\mathsf{Verify}_{\mathsf{pp},\mathsf{pk}}(m^*, r^*, s^*, i^*)$
    abort†
  $(m'^*, r'^*, s'^*, \rho'^*, i'^*) \twoheadleftarrow \mathcal{A}'(\mathsf{pp}, \mathsf{pk})$
  if $i^* \neq i'^*$ then abort§
  if $r^* \neq r'^*$ then abort§
  $e'^* \leftarrow \mathcal{R}.H_{i^*}(m'^*, r^* \bmod q_{i^*}, \rho'^*)$
  if $e^* = e'^*$ then abort§
  if not $\mathsf{Verify}_{\mathsf{pp},\mathsf{pk}}(m'^*, r^*, s'^*, i^*)$
    abort†
  $\Delta s \leftarrow s^* - s'^*$
  $\Delta e \leftarrow e'^* - e^*$
  return $(i^*, \Delta s/\Delta e)$

$\mathcal{R}.\mathsf{Sign}'(m)$:
  $l \leftarrow 0$
  return $\Sigma.[l]$
  $l \leftarrow l + 1$

$\mathcal{R}.H(\alpha)$:
  if $\exists(\alpha', h') \in L$ s.t. $\alpha' = \alpha$
    return h'
  else
    $h \twoheadleftarrow \mathbb{Z}_p$
    $L \leftarrow L \cup \{(\alpha, h)\}$
    return h

$\mathcal{R}.H'(\alpha)$:
  if $\exists(\alpha', h') \in L'$ s.t. $\alpha' = \alpha$
    return h'
  else
  if $k \leq a$
    $(\alpha', h') \leftarrow L.[k]$
    return h'
    $k \leftarrow k + 1$
    $L' \leftarrow L' \cup \{(\alpha, h)\}$
  else
    $h \twoheadleftarrow \mathbb{Z}_p$
    $L' \leftarrow L' \cup \{(\alpha, h)\}$
    return h

$\mathcal{R}.\mathsf{Sign}(m)$:
  if $j = 1$
    $(r, e_1, \ldots, e_\ell, s_1, \ldots, s_\ell, \rho_1, \ldots, \rho_\ell) \twoheadleftarrow \delta'$
    if $\exists h$ s.t. $((m, r \bmod q_1, \rho_1), h) \in L$
      abort⋆
    $L \leftarrow L \cup \{((m, r \bmod q_1, \rho_1), e_1)\}$
    $j \leftarrow j + 1 \bmod \ell$
    return $(s_1, r, \rho_1, 1)$
    $\Sigma \leftarrow \Sigma \cup \{(s_1, r, \rho_1, 1)\}$
  else
    if $\exists h$ s.t. $((m, r \bmod q_j, \rho_j), h) \in L$
      abort⋆
    $L \leftarrow L \cup \{(m, r \bmod q_j, \rho_j), e_j\}$
    $j \leftarrow j + 1 \bmod \ell$
    return (s_j, r, ρ_j, j)
    $\Sigma \leftarrow \Sigma \cup \{(s_j, r, \rho_j, j)\}$

Fig. 5: An efficient solver for the PDLP, constructed from an efficient EUF-CMA adversary against our scheme.

number of queries to the signature oracle, and ℓ. Note that the reduction is ℓ times looser than [23]. This concludes the proof of Theorem 3. □

Proof (of Theorem 1). Using Theorem 2, we can use Sign_0 instead of Sign_2 as a target for the attacker in Theorem 3. □

4 Provably Secure Pre-Computations

Often the bottleneck in implementations centers around modular exponentiation. In this section we briefly outline several proposed *pre-computation* techniques, as well as presenting in more detail two pre-computation schemes which were used in our implementation to compare timings between classical Schnorr and our scheme.

4.1 Brief Overview of Speed-up techniques

The problem of computing modular exponentiations is well-known to implementers of both DLP-based and RSA-based cryptosystems. In the specific case that we want to compute $g^x \bmod p$, the following strategies have been proposed but their security is often heuristic:

- Use signed expansions (only applicable to groups where inversion is efficient);
- Use Frobenius expansions or the GLV/GLS method (only applicable to certain elliptic curves);
- Batch exponentiations together, as suggested by M'Raïhi and Naccache [18].

The above approaches work for arbitrary values of x. Alternatively, one may choose a particular value of x with certain properties which make computation faster; however there is a possibility that doing so weakens the DLP:

- Choose x with low Hamming weight as proposed by Agnew et al. [1];
- Choose x to be a random Frobenius expansion of low Hamming weight, as discussed by Galbraith [12, Sec. 11.3];
- Choose x to be given by a random addition chain, as proposed by Schroeppel et al. [25];
- Choose x to be a product of low Hamming weight integers as suggested by Hoffstein and Silverman [13]—broken by Cheon and Kim [6];
- Choose x to be a small random element in GLV representation—broken by Aranha et al. [2];

Finally, a third branch of research uses large amounts of pre-computation to generate random pairs $(x, g^x \bmod p)$. The first effort in this direction was Schnorr's [24], quickly broken by de Rooij [9]. Other constructions are due to Brickell et al. [5], Lim and Lee [17], and de Rooij [8]. The first provably secure solution is due to Boyko et al. [4], henceforth BPV, which was extended and made more precise by [7, 19, 20]. This refined algorithm is called E-BPV (extended BPV).

4.2 The E-BPV Pre-computation Scheme

E-BPV[8] relies on pre-computing and storing a set of n pairs $(k_i, g^{k_i} \bmod p)$; then a "random" pair $(r, g^r \bmod p)$ is generated by choosing a subset S of size k the

[8] BPV is a special case of E-BPV where $h = 2$. As such they share the same precomputing step.

k_i, and for each i a random exponent x_i between 1 and h. Then a pair (r, R) is computed as $r \leftarrow \Sigma_{i \in S} x_i d_i \bmod \phi(p)$, $R \leftarrow g^r \bmod p$ with a non trivial speedup due to Brickell et al. [5] (BGMW). To guarantee an acceptable level of security, and resist lattice reduction attacks, the number n of precomputed pairs must be sufficiently large; and enough pairs with large enough exponents must be used to generate a new couple.

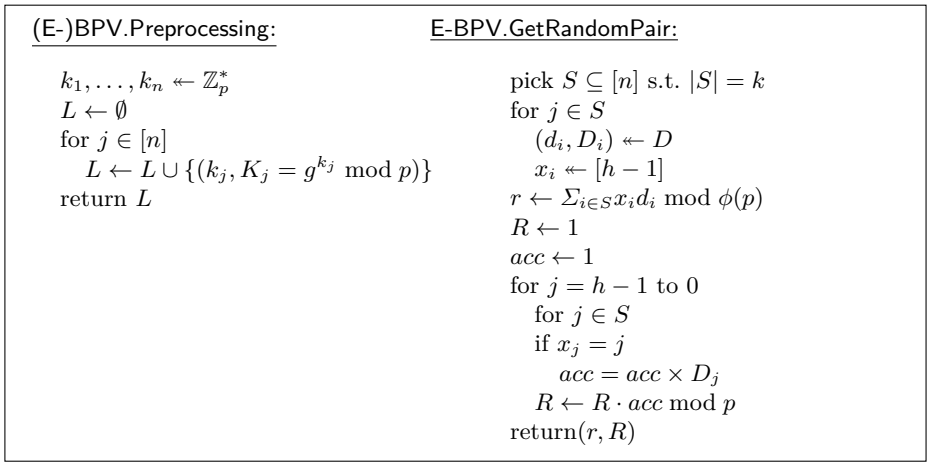

Fig. 6: The E-BPV algorithm for generating random pairs $(x, g^x \bmod p)$. The BPV algorithm is a special case of E-BPV for $h = 2$.

Nguyen et al. [19] showed that using E-BPV instead of standard exponentiation gives an adversary an advantage bounded by

$$m\sqrt{\frac{K}{\binom{n}{k}(h-1)^k}}$$

with m the number of signature queries by the adversary, (k, n, h) E-BPV parameters, and K the exponent's size.[9]

We fix conservatively $m = 2^{128}$. For our scheme, at 128-bit security, we have $K = P = 3072$. As suggested in [19] we set $n = k$, and constrain our memory:

$$h^k \geq 2^{3400}$$

Optimizing $2k + h$ under this constraint, we find $(h, k) = (176, 455)$. This corresponds to 1087 modular multiplications, i.e., an amortized cost of 90 multiplications per signature, for about 170 kB of storage.

Alternatively, we can satisfy the security constraints by setting $n = 2048$, $h = 100$, $k = 320$, which corresponds to about 770 kB of storage, giving an amortized cost of 62 modular multiplications per signature.

[9] For Schnorr, the exponent's size is Q; for our scheme, it is P.

In the implementation (Section 5), we solve the constrained optimisation problem to find the best coefficients (i.e., the least number of multiplications) for a given memory capacity.

Remark 3 (Halving storage cost). The following idea can halve the amount of storage required for the couples (x, g^x): instead of drawing the values x at random, we draw a master secret s once, and compute $x_{i+1} = \mathrm{OWF}(x, g^{x_i})$ with OWF being a one-way function. Only s, x_0, and the values g^{x_i} need to be stored; instead of all the couples (x_i, g^{x_i}). This remark applies to both BPV and E-BPV.

4.3 Lim and Lee Precomputation Scheme

We also consider a variation on Lim and Lee's fast exponentiation algorithm [17]. Their scheme originally computes g^r for r known in advance, but it is easily adapted to the setting where r is constructed on the fly. The speed-up is only linear, however, which ultimately means we cannot expect a sizable advantage over Schnorr. Nevertheless, Lim and Lee's algorithm is less resource-intensive and can be used in situations where no secure E-BPV parameters can be found (e.g., in ultra-low memory settings).

The Lim-Lee scheme (LL) has two parameters, h and v. In the original LL algorithm, the exponent is known in advance, but it is easily modified to generate an exponent on the fly. Intuitively, it consists in splitting the exponent in a "blocks" of size h, and dividing further each block in b sub-blocks of size v. The number of modular multiplications (in the worst case) is $a + b - 2$, and we have to store $(2^h - 1)v$ pairs. The algorithms are given in Figure 7.

For a given amount of memory M, it is easy to solve the constrained optimization problem, and we find

$$h_{\mathrm{opt}} = \frac{1}{\ln(2)} \left(1 + W\left(\frac{1+M}{e}\right)\right)$$

where W is the Lambert function. For a memory M of 750 kB, this gives $h \approx 8.6$. The optimal parameters for integers are $h = 9$ and $v = 4$.[10]

Remark 4. For LL, Remark 3 on halving storage requirements does not apply, as x need not be stored.

A summary of the properties for the pre-computations techniques E-PBV and LL can be found in Table 1.

5 Implementation Results

Our scheme, using the algorithms described in Sections 3 and 4, has been implemented in C using the GMP library. In the interest of timing comparison

[10] In practice, it turns out that $h = v = 8$ performs slightly better, due to various implementation speed-ups possible in this situation

LimLee.Preprocessing(h, v):

$g_0 \leftarrow g$
$L = \emptyset$
for $i = 0$ to $h - 1$
  $g_i \leftarrow g_{i-1}^{2^a}$
for $i = 0$ to $2^h - 1$
  let $i = e_{h-1} \dots e_1$ in binary
  $g_{0,i} = g_{h-1}^{e_{h-1}} \dots g_1^{e_1}$
for $i = 0$ to $2^h - 1$
  for $j = 0$ to $v - 1$
    $g_{j,i} \leftarrow g_{j-1,i}^{2^b}$
    $L \leftarrow L \cup \{g_{j,i}\}$
return L

LimLee.GetRandomPair:

$R \leftarrow 1$
$r \leftarrow 0$
for $i = b - 1$ to 0
  $R \leftarrow R^2$
  $r \leftarrow r + r$
for $j = v - 1$ to 0
  $r_{i,j} \twoheadleftarrow \{0, \dots, 2^h - 1\}$
  $R \leftarrow R \times g_{j,r_{i,j}}$
  $r \leftarrow r + r_{i,j}$
return (r, R)

Fig. 7: The LL algorithm for generating random pairs $(x, g^x \bmod p)$.

Table 1: Precomputation/online computation trade-offs.

Algorithm	Storage	Multiplications	Security
Square-and-multiply	0	$1.5 \log P$	Always
BPV [4]	nP	$k - 1$	$m\sqrt{\frac{P}{\binom{n}{k}}} < 2^{-\kappa}$
E-BPV [19]	nP	$2k + h - 3$	$m\sqrt{\frac{P}{\binom{n}{k}(h-1)^k}} < 2^{-\kappa}$
Lim and Lee [17]	$2^h \times v \times P$	$\frac{\log P}{h}(1 + \frac{1}{v}) - 3$	Always

we have also implemented the classical Schnorr scheme. The results for several scenarios are outlined in Table 2 (at 128-bit security) and Table 3 (at 192-bit security). Complete source code and timing framework are available upon request from the authors.

These experiments show that our scheme is faster than Schnorr when at least 250 pairs (i.e., 750 kB at 128-bit security) have been precomputed. This effect is even more markedly visible at higher security levels: our scheme benefits more, and more effectively, from the E-BPV+BGMW optimisation as compared to Schnorr.

Schnorr and our scheme achieve identical performance when using Lim and Lee's optimisation, confirming the theoretical analysis. When less than 1 MB of memory is allotted, this is the better choice.

6 Heuristic security

Several papers describe server-aided precomputation techniques (e.g., [16]), which perform exponentiations with the help of a (possibly untrusted) server, i.e., such

Table 2: Timing results for Schnorr and our scheme, at 128-bit security ($P = 3072$, $Q = 256$). Computation was performed on an ArchLinux single-core 32-bit virtual machine with 128 MB RAM. Averaged over 256 runs.

Scheme	Storage	Precomp.	Time (per sig.)	Verify
Schnorr	–	–	6.14 ms	73.9 ms
Schnorr + [19] + [5]	170 kB	33 s	2.80 ms	73.9 ms
Schnorr + [19] + [5]	750 kB	33 s	2.03 ms	73.9 ms
Schnorr + [19] + [5]	1 MB	34 s	2.00 ms	73.9 ms
Schnorr + [19] + [5]	2 MB	37 s	2.85 ms	73.9 ms
Schnorr + [17]	165 kB	3 s	949 μs	73.9 ms
Schnorr + [17]	750 kB	3 s	644 μs	73.9 ms
Schnorr + [17]	958 kB	3 s	630 μs	73.9 ms
Schnorr + [17]	1.91 MB	3 s	★ 472 μs	73.9 ms
Our Scheme	–	–	5.94 ms	2.4 s
Our Scheme + [19] + [5]	170 kB	33 s	1.23 ms	2.4 s
Our Scheme + [19] + [5]	750 kB	33 s	426 μs	2.4 s
Our Scheme + [19] + [5]	1 MB	34 s	371 μs	2.4 s
Our Scheme + [19] + [5]	2 MB	37 s	★ 327 μs	2.4 s
Our Scheme + [17]	165 kB	3 s	918 μs	2.4 s
Our Scheme + [17]	750 kB	3 s	709 μs	2.4 s
Our Scheme + [17]	958 kB	3 s	650 μs	2.4 s
Our Scheme + [17]	1.91 MB	3 s	757 μs	2.4 s

Table 3: Timing results for Schnorr and our scheme, at 192-bit security ($P = 7680$, $Q = 384$). Computation was performed on an ArchLinux single-core 32-bit virtual machine with 128 MB RAM. Averaged over 256 runs.

Scheme	Storage	Time (/sig.)
Schnorr	–	35.2 ms
Schnorr + [17]	715 kB	508 μs
Schnorr + [19] + [5]	750 kB	2.08 ms
Schnorr + [19] + [5]	1.87 MB	1.62 ms
Schnorr + [17]	1.87 MB	★ 476 μs
Our Scheme	–	33.0 ms
Our Scheme + [17]	715 kB	486 μs
Our Scheme + [17]	1.87 MB	467 μs
Our Scheme + [19] + [5]	1.87 MB	★ 263 μs

techniques allow to outsource the computation of $g^x \bmod n$, with public g and n, without revealing x to the server.

Interestingly, the most efficient algorithms in that scenario (which of course we could leverage) use parameters provided by Hohenberger and Lysyanskaya [14] for E-BPV. A series of papers took these parameters for granted (including [16]),

but we should point out that *these are not covered* by the security proof found in [19].

Despite this remark, it seems that no practical attack is known either; therefore if we are willing to relax our security expectations somewhat it is possible to compute the modular exponentiation faster. Namely, a Q-bit exponent can be computed in $O(\log Q^2)$ modular multiplications.

Our Scheme uses an exponent that is ℓ times bigger than Schnorr, which is amortized over ℓ signatures. Comparing our scheme to Schnorr, the ratio is $\frac{\ell \log(Q)^2}{(\log \ell Q)^2}$. With $Q = 256$ and $\ell = 12$ we get a ratio of approximately 5.7.

Note that as Q increases, so does ℓ, and therefore so does the advantage of our scheme over Schnorr in that regime.

7 Conclusion

We have introduced a new digital signature scheme variant of Schnorr signatures, that reuses the nonce component for several signatures. Doing so does not jeopardise the scheme's security; attempting to do the same with classical Schnorr signatures would immediately reveal the signing key. However the main appeal of our approach is that precomputation techniques, whose benefits can only be seen for large enough problems, become applicable and interesting. As a result, without loss of security, it becomes possible to sign messages using fewer modular multiplications. Our technique is general and can be applied to several signature schemes using several speed-up techniques.

References

1. Agnew, G.B., Mullin, R.C., Onyszchuk, I.M., Vanstone, S.A.: An implementation for a fast public-key cryptosystem. Journal of Cryptology 3(2), 63–79 (1991)
2. Aranha, D.F., Fouque, P.A., Gérard, B., Kammerer, J.G., Tibouchi, M., Zapalowicz, J.C.: GLV/GLS decomposition, power analysis, and attacks on ECDSA signatures with single-bit nonce bias. In: Sarkar, P., Iwata, T. (eds.) ASIACRYPT 2014, Part I. LNCS, vol. 8873, pp. 262–281. Springer, Heidelberg, Germany, Kaoshiung, Taiwan, R.O.C. (Dec 7–11, 2014)
3. Bellare, M., Neven, G.: Multi-signatures in the plain public-key model and a general forking lemma. In: Juels, A., Wright, R.N., Vimercati, S. (eds.) ACM CCS 06. pp. 390–399. ACM Press, Alexandria, Virginia, USA (Oct 30 – Nov 3, 2006)
4. Boyko, V., Peinado, M., Venkatesan, R.: Speeding up discrete log and factoring based schemes via precomputations. In: Nyberg, K. (ed.) EUROCRYPT'98. LNCS, vol. 1403, pp. 221–235. Springer, Heidelberg, Germany, Espoo, Finland (May 31 – Jun 4, 1998)
5. Brickell, E.F., Gordon, D.M., McCurley, K.S., Wilson, D.B.: Fast exponentiation with precomputation (extended abstract). In: Rueppel, R.A. (ed.) EUROCRYPT'92. LNCS, vol. 658, pp. 200–207. Springer, Heidelberg, Germany, Balatonfüred, Hungary (May 24–28, 1993)
6. Cheon, J.H., Kim, H.: Analysis of low hamming weight products. Discrete Applied Mathematics 156(12), 2264–2269 (2008), http://dx.doi.org/10.1016/j.dam.2007.09.018

7. Coron, J., M'Raïhi, D., Tymen, C.: Fast generation of pairs (k, [k]p) for koblitz elliptic curves. In: Vaudenay, S., Youssef, A.M. (eds.) Selected Areas in Cryptography, 8th Annual International Workshop, SAC 2001 Toronto, Ontario, Canada, August 16-17, 2001, Revised Papers. Lecture Notes in Computer Science, vol. 2259, pp. 151–164. Springer (2001), http://dx.doi.org/10.1007/3-540-45537-X_12
8. de Rooij, P.: Efficient exponentiation using procomputation and vector addition chains. In: Santis, A.D. (ed.) EUROCRYPT'94. LNCS, vol. 950, pp. 389–399. Springer, Heidelberg, Germany, Perugia, Italy (May 9–12, 1995)
9. de Rooij, P.: On Schnorr's preprocessing for digital signature schemes. Journal of Cryptology 10(1), 1–16 (1997)
10. ElGamal, T.: On computing logarithms over finite fields. In: Williams, H.C. (ed.) CRYPTO'85. LNCS, vol. 218, pp. 396–402. Springer, Heidelberg, Germany, Santa Barbara, CA, USA (Aug 18–22, 1986)
11. Fiat, A., Shamir, A.: How to prove yourself: Practical solutions to identification and signature problems. In: Odlyzko, A.M. (ed.) CRYPTO'86. LNCS, vol. 263, pp. 186–194. Springer, Heidelberg, Germany, Santa Barbara, CA, USA (Aug 1987)
12. Galbraith, S.D.: Mathematics of Public Key Cryptography. Cambridge University Press (2012), https://www.math.auckland.ac.nz/~sgal018/crypto-book/crypto-book.html
13. Hoffstein, J., Silverman, J.H.: Random small hamming weight products with applications to cryptography. Discrete Applied Mathematics 130(1), 37–49 (2003), http://dx.doi.org/10.1016/S0166-218X(02)00588-7
14. Hohenberger, S., Lysyanskaya, A.: How to securely outsource cryptographic computations. In: Kilian, J. (ed.) TCC 2005. LNCS, vol. 3378, pp. 264–282. Springer, Heidelberg, Germany, Cambridge, MA, USA (Feb 10–12, 2005)
15. Katz, J., Lindell, Y.: Introduction to Modern Cryptography. Chapman and Hall/CRC Press (2007)
16. Kiraz, M.S., Uzunkol, O.: Efficient and verifiable algorithms for secure outsourcing of cryptographic computations. Int. J. Inf. Sec. 15(5), 519–537 (2016), http://dx.doi.org/10.1007/s10207-015-0308-7
17. Lim, C.H., Lee, P.J.: More flexible exponentiation with precomputation. In: Desmedt, Y. (ed.) CRYPTO'94. LNCS, vol. 839, pp. 95–107. Springer, Heidelberg, Germany, Santa Barbara, CA, USA (Aug 21–25, 1994)
18. M'Raïhi, D., Naccache, D.: Batch exponentiation: A fast dlp-based signature generation strategy. In: Gong, L., Stearn, J. (eds.) CCS '96, Proceedings of the 3rd ACM Conference on Computer and Communications Security, New Delhi, India, March 14-16, 1996. pp. 58–61. ACM (1996), http://doi.acm.org/10.1145/238168.238187
19. Nguyen, P.Q., Shparlinski, I.E., Stern, J.: Distribution of modular sums and the security of the server aided exponentiation. In: Cryptography and Computational Number Theory, pp. 331–342. Springer (2001)
20. Nguyen, P.Q., Stern, J.: The hardness of the hidden subset sum problem and its cryptographic implications. In: Wiener, M.J. (ed.) CRYPTO'99. LNCS, vol. 1666, pp. 31–46. Springer, Heidelberg, Germany, Santa Barbara, CA, USA (Aug 15–19, 1999)
21. Pohlig, S.C., Hellman, M.E.: An improved algorithm for computing logarithms over gf(p) and its cryptographic significance (corresp.). IEEE Trans. Information Theory 24(1), 106–110 (1978), http://dx.doi.org/10.1109/TIT.1978.1055817
22. Pointcheval, D., Stern, J.: Security proofs for signature schemes. In: Maurer, U.M. (ed.) EUROCRYPT'96. LNCS, vol. 1070, pp. 387–398. Springer, Heidelberg, Germany, Saragossa, Spain (May 12–16, 1996)

23. Pointcheval, D., Stern, J.: Security arguments for digital signatures and blind signatures. Journal of Cryptology 13(3), 361–396 (2000)
24. Schnorr, C.P.: Efficient identification and signatures for smart cards. In: Brassard, G. (ed.) CRYPTO'89. LNCS, vol. 435, pp. 239–252. Springer, Heidelberg, Germany, Santa Barbara, CA, USA (Aug 20–24, 1990)
25. Schroeppel, R., Orman, H.K., O'Malley, S.W., Spatscheck, O.: Fast key exchange with elliptic curve systems. In: Coppersmith, D. (ed.) CRYPTO'95. LNCS, vol. 963, pp. 43–56. Springer, Heidelberg, Germany, Santa Barbara, CA, USA (Aug 27–31, 1995)
26. Shanks, D.: Class number, a theory of factorization and genera. Proceedings of Symposia in Pure Mathematics 20, 415–440 (1970)
27. Shoup, V.: Lower bounds for discrete logarithms and related problems. In: Fumy, W. (ed.) EUROCRYPT'97. LNCS, vol. 1233, pp. 256–266. Springer, Heidelberg, Germany, Konstanz, Germany (May 11–15, 1997)

A Generic Security of Partial Discrete Logarithm

In this section, we prove that the partial discrete logarithm problem introduced in Section 3.2 is intractable in the generic group model. This model was introduced by Shoup [27] for measuring the exact difficulty of solving classical discrete logarithm problems. Algorithms in generic groups do not exploit any properties of the encodings of group elements. They can access group elements only via a random encoding algorithm that encodes group elements as random bit-strings.

Proofs in the generic group model provide heuristic evidence of some problem hardness when an attacker does not take advantage of group elements' encoding. However, they do not necessarily say anything about the difficulty of specific problems in a concrete group.

Let ℓ be some non-negative integers, let $q_1, \ldots, q_\ell$ be some distinct prime numbers and let $q = q_1 \cdots q_\ell$. We consider a cyclic group $\mathbb{G}$ of (composite) order q generated by g. We assume without loss of generality that $q_1 = \max(q_1, \ldots, q_\ell)$. A classical method [21] to solve the partial discrete logarithm problem in $\mathbb{G}$ given $h = g^x \in \mathbb{G}$ is to compute $h^{q_2 \cdots q_\ell}$, an element of order dividing q_1 (that belongs to the subgroup generated by $g^{q_2 \cdots q_\ell}$) and to compute its discrete logarithm x_1 in base $g^{q_2 \cdots q_\ell}$ using a square root method such as Shanks "baby-step giant-step" algorithm [26]. It is easy to see that x_1 is equal to $x \bmod q_1$ and is obtained within time complexity $O(\sqrt{q_1} + \log(q_2 \cdots q_\ell))$ group operations.

Our goal is to prove that this time complexity is essentially optimal in the generic group model. Let $\mathcal{A}$ be a generic group adversary that solves the partial discrete logarithm problem in $\mathbb{G}$. As usual, the generic group model is implemented by choosing a random encoding $\sigma : \mathbb{G} \longrightarrow \{0,1\}^m$. Instead of working directly with group elements, $\mathcal{A}$ takes as input their image under σ. This way, all $\mathcal{A}$ can test is string equality. $\mathcal{A}$ is also given access to an oracle computing group multiplication and division: taking $\sigma(g_1)$ and $\sigma(g_2)$ and returning $\sigma(g_1 \cdot g_2)$ and $\sigma(g_1/g_2)$ respectively. Finally, we can assume that $\mathcal{A}$ submits to the oracle only encodings of elements it had previously received. This is because we can choose m large enough so that the probability of choosing a string that is also in the image of σ is negligible.

Theorem 4. *Let $\mathcal{A}$ be a generic algorithm that takes as input two encodings $\sigma(g)$ and $\sigma(h)$ (where g is a generator of $\mathbb{G}$ and $h = g^x \in \mathbb{G}$) and makes at most τ group oracle queries, then $\mathcal{A}$'s advantage in outputting a partial discrete logarithm (i, x_i) with $i \in \{1, \ldots, \ell\}$ and $x_i = x \bmod q_i$ is upper-bounded by $O(\tau^2/q_1)$.*

Proof. We consider an algorithm $\mathcal{B}$ playing the following game with $\mathcal{A}$. Algorithm $\mathcal{B}$ picks two bit strings σ_1, σ_2 uniformly at random in $\{0,1\}^m$. Internally, $\mathcal{B}$ keeps track of the encoded elements using elements in the ring $\mathbb{Z}_{q_1}[X_1] \times \cdots \times \mathbb{Z}_{q_\ell}[X_\ell]$. To maintain consistency with the bit strings given to $\mathcal{A}$, $\mathcal{B}$ creates a lists $\mathcal{L}$ of pairs (F, σ) where F is a polynomial vector in the ring $\mathbb{Z}_{q_1}[X_1] \times \cdots \times \mathbb{Z}_{q_\ell}[X_\ell]$ and $\sigma \in \{0,1\}^m$ is the encoding of a group element. The polynomial vector F represents the exponent of the encoded element in the group $\mathbb{Z}_{q_1} \times \cdots \times \mathbb{Z}_{q_\ell}$. Initially, $\mathcal{L}$ is set to

$$\{((1, 1, \ldots, 1), \sigma_1), ((X_1, \ldots, X_n), \sigma_2)\}$$

Algorithm $\mathcal{B}$ starts the game providing $\mathcal{A}$ with σ_1 and σ_2. The simulation of the group operations oracle goes as follows:

Group operation: Given two encodings σ_i and σ_j in $\mathcal{L}$, $\mathcal{B}$ recovers the corresponding vectors F_i and F_j and computes $F_i + F_j$ for multiplication (or $F_i - F_j$ for division) termwise. If $F_i + F_j$ (or $F_i - F_j$) is already in $\mathcal{L}$, $\mathcal{B}$ returns to $\mathcal{A}$ the corresponding bit string; otherwise it returns a uniform element $\sigma \xleftarrow{R} \{0,1\}^m$ and stores $(F_i + F_j, \sigma)$ (or $(F_i - F_j, \sigma)$) in $\mathcal{L}$.

After $\mathcal{A}$ queried the oracles, it outputs a pair $(i^*, x_i^*) \in \{1 \ldots, \ell\} \times \mathbb{Z}_{q_{i^*}}$ as a candidate for the partial discrete logarithm of h in base g. At this point, $\mathcal{B}$ chooses uniform random values $x_1, \ldots, x_n \in \mathbb{Z}_{q_1} \times \cdots \times \mathbb{Z}_{q_\ell}$. The algorithm $\mathcal{B}$ sets $X_i = x_i$ for $i \in \{1, \ldots, n\}$.

If the simulation provided by $\mathcal{B}$ is consistent, it reveals nothing about $(x_1, \ldots, x_\ell)$. This means that the probability of $\mathcal{A}$ guessing the correct value for $(i^*, x_i^*) \in \{1, \ldots, \ell\} \times \mathbb{Z}_{q_{i^*}}$ is $1/q_{i^*}$. The only way in which the simulation could be inconsistent is if, after we choose value for $x_1, \ldots, x_n$, two different polynomial vectors in $\mathcal{L}$ happen to produce the same value.

It remains to compute the probability of a collision happening due to a unlucky choice of values. In other words, we have to bound the probability that two distinct vectors F_i, F_j in $\mathcal{L}$ evaluate to the same value after the substitution, namely $F_i(x_1, \ldots, x_n) - F_j(x_1, \ldots, x_n) = 0$. This reduces to bound the probability of hitting a zero of $F_i - F_j$. By the simulation, this happens only if $F_i - F_j$ is a vector of polynomials where at least one coordinate — say the k-th — is a non-constant polynomial (and thus of degree one) denoted $(F_i - F_j)^{(k)}$.

Recall that the Schwartz-Zippel lemma says that, if F is a degree d polynomial in $\mathbb{Z}_{q_k}[X_k]$ and $S \subseteq \mathbb{Z}_{q_k}$ then

$$\Pr[F(x_k) = 0 \bmod q_k] \leq \frac{d}{|S|}$$

where x_k is chosen uniformly from S. Going back to our case, we obtain by applying the Schwartz-Zippel lemma :

$$\Pr[(F_i - F_j)^{(k)}(x_k) = 0 \in \mathbb{Z}_{q_k}] \leq 1/q_k \leq 1/q_1.$$

Therefore, the probability that the simulation provided by $\mathcal{B}$ is inconsistent is upper-bounded by $\tau(\tau - 1)/q_1$ (by the union bound) and the result follows. □

B Reduction-friendly moduli

As part of computing $g^k \bmod p$, a very costly operation is the reduction mod p. An interesting question is whether some particular moduli p can be found, for which reduction is particularly easy.

An example of such moduli are those that start with a `1` followed by many `0`.

Example 1. For $P = 3072$ and $Q = 256$, using (in hexadecimal notation)

$$\Delta_i = \{\texttt{12d}, \texttt{165}, \texttt{1e7}, \texttt{247}, \texttt{2f5}, \texttt{31b}, \texttt{327}, \texttt{34f}, \texttt{3a3}, \texttt{439}, \texttt{56b}, \texttt{4fe7}\}$$

and $q_i = 2^Q + \Delta_i$, we have that p equals:

```
2[60]e0e8[56]18058164[53]1479d1e16e8[51]aa09581f139be[48]3a9dc2e99b
080dd[47]dfe705c4e9b3a45678[43]25a378c4e6b62835f401[42]471d330fbde5
6ef2c80281e[39]5c5388621a308a5425f007648[37]4e506ba1a5b68dc5faca115
5e64[35]270051399124b193e6716e08b4408[34]8a07b85ed815e7eac1135861bd
67e3
```

where `[x]` denotes a sequence of `x` hexadecimal zeros.