Speeding up Discrete Log and Factoring Based Schemes via Precomputations

Victor Boyko*[1], Marcus Peinado**[2], and Ramarathnam Venkatesan*[3]

[1] Massachusetts Institute of Technology, LCS, USA boyko@theory.lcs.mit.edu
[2] GMD Research Center, SCAI, 53754 St. Augustin, Germany, peinado@gmd.de
[3] Microsoft Research, Redmond, WA 98052, USA, venkie@microsoft.com

Abstract. We present fast and practical methods for generating randomly distributed pairs of the form $(x, g^x \bmod p)$ or $(x, x^e \bmod N)$, using precomputation. These generation schemes are of wide applicability for speeding-up public key systems that depend on exponentiation and offer a smooth memory-speed trade-off. The steps involving exponentiation in these systems can be reduced significantly in many cases. Our schemes are most suited for server applications. We present security analyses of our schemes using standard assumptions, including analyses for fully adaptive attacks. Our methods are novel in the sense that they identify and thoroughly exploit the randomness issues related to the instances generated in these public-key schemes. Our constructions use random walks on Cayley (expander) graphs over Abelian groups. Our analysis involves non-linear versions of lattice problems. It appears that any realistic attack on our schemes would need to solve such problems.

1 Introduction

Modular exponentiation is a basic operation widely used in cryptography and constitutes a computational bottleneck in many protocols. This work presents a method to significantly speed up modular exponentiation.

In practice, two versions of modular exponentiation are most frequent: In factoring based schemes, $x^e \bmod N$ must often be computed for fixed $N = pq$ (p, q primes), fixed e and many randomly chosen x. Many discrete log based schemes compute terms of the form $g^x \bmod p$, for a fixed $g \in \mathbb{Z}_p^*$ and many random x.

The well known square-and-multiply algorithm for modular exponentiation requires, on average, $1.5\,n$ modular multiplications for an n-bit exponent. In the case of 512-bit integers, the algorithm performs, on average, 766 modular multiplications of 512 bit numbers. Several authors [5, 10, 19] describe alternative algorithms for the discrete log case which reduce the number of multiplications per exponentiation by means of precomputation and table lookup. These algorithms allow a time-memory trade-off. For 512-bit numbers, the number of

* Part of the work done while at Bellcore.
** Part of the work was done while visiting ICSI, Berkeley, CA.

multiplications can be reduced to about 100, using a modest amount of memory and precomputation time.

We present new methods to reduce the cost of exponentiation even further. In the discrete log case, our scheme may need significantly fewer multiplications than even the improved algorithms of [5, 10, 19] (depending on the parameter choices; cf. Section 5). This improvement is even more pronounced when compared to square-and-multiply. Note that the algorithms of [5, 10, 19] apply only to the discrete log case. To the best of our knowledge, our scheme is the only available method to speed up exponentiation in the factoring case, and research into variants may be of interest.

The key to these performance improvements lies in abandoning the basic input-output relation the known algorithms adhere to: unlike these algorithms, our scheme does not receive x as its input but generates a random x together with $g^x \bmod p$, or $x^e \bmod N$, respectively. While this joint generation makes it possible to reduce the number of multiplications noticeably, it also limits the direct applicability of our scheme to protocols in which a party generates a random x and computes $g^x \bmod p$ or $x^e \bmod N$. Still, there are many cryptographic protocols which involve exactly these two steps and which, in some cases, are speeded up significantly by our generation scheme. We present some examples: Diffie-Hellman key exchange [12], ElGamal encryption [14], ElGamal and DSS signatures [14, 13], Schnorr's schemes for authentication [23, 24], 2^m-th root identification scheme [27], and versions of the RSA cryptosystem which we define here.

The simplest version of our generator is based on a subset sum or subset product construction. The set of possible outputs x is determined by the set of possible subsets which can be generated given the parameters of the generator. A second version of the generator combines the basic version with a random walk on an expander which is a Cayley graph of an Abelian group. The random walk component expands the set of possible outputs incrementally and spans the entire underlying group. Due to space limitations, we do not present the analysis of this scheme in this version. We remark that our methods depend only on the group structure of the underlying domain and are thus also applicable to elliptic curve based schemes.

The main part of the paper is devoted to analyzing the security of protocols under the distribution of outputs produced by our generators. This is necessary since correlations in the generator's outputs can introduce potential weaknesses which do not arise in [5, 10, 19]. A scheme for fast generation of pairs of the form $(x, g^x \bmod p)$ was proposed by Schnorr [23] for use in his authentication and signature scheme. It was broken by de Rooij for the small parameters suggested by Schnorr [9], and fixed by Schnorr [24]. This version was also broken by de Rooij [11]. De Rooij's attack easily extends to any discrete-log based signature scheme for which an equation linear in the random parameter can be written (e.g. ElGamal, DSS, and Brickell-McCurley). De Rooij's attack is based on linear relations between the consecutive outputs and the tables of Schnorr's generator. We note that an attack of this sort cannot be applied to our generator.

Models of Analysis: Instead of using a restricted adversary model, such as the 'black box model' of Nechaev [21] or Shoup [28], our analysis considers general adversaries. It proceeds either without additional assumptions or invokes standard complexity assumptions (e.g. hardness of factoring). We believe that this approach, by treating more general adversaries, can yield better optimizations. In some cases (e.g. Diffie-Hellman key exchange, ElGamal encryption, ElGamal and DSS signatures, and Schnorr's authentication and signature schemes) we present an analysis which proves security against certain kinds of non-adaptive attacks without using additional assumptions.

To analyze fully adaptive attacks, we fall back on the known pseudo-randomness properties of subset sum problems: Impagliazzo and Naor [16] have shown that, given random weights a_i $(i \leq n)$ and a modulus M, a random subset sum b modulo M is indistinguishable from a random string of the same length. Using this and further observations, we show that several protocols which use our generator are immune even against fully adaptive attacks, provided the original protocols are secure and the adversary cannot break the underlying subset-sum-type problem. Consideration of the most efficient attacks on the subset sum problem [26] and our own experiments indicate that solving this problem (especially the hidden-weight version described below) is hard for the parameters used in practice (see below).

RSA-based schemes: We present some RSA-like systems for applications with large encryption exponent. Commonly, the RSA public key exponents are chosen to be small in order to reduce encryption times. Consequently decryption takes far longer than encryption. In RSA signature schemes, the situation is reversed. It may be desirable to decrease the asymmetry of loads on the two ends and to have roughly similar costs for encryption and decryption. For example, a server which is networked with many small clients that form frequent short-lived sessions may be (paradoxically) overloaded. Formally, our speedup scheme can only be applied to the encryption of messages. Decryption times can be reduced by using small decryption exponents d, which should be chosen according to Wiener's recommendations [29] (also discussed later). We also note that certain attacks on RSA exploit low exponents, and some future applications appear to require large exponents [3, 7]. We analyze the generation schemes using standard assumptions.

Lattice attacks on subset sum problems: Subset sum constructions have been so successfully attacked by lattice reduction [18] based methods [4, 17, 8] that it is often considered risky to base cryptographic constructions on them. Our experiments show that the L^3 algorithm can be expected to solve subset sum problems up to about $n = 40$, where n is the size of the set from which subset sums are formed. Let ℓ be the length of the integers in this set. As n becomes larger than 40, L^3 finds the shortest vector only if ℓ/n (or n/ℓ) exceeds some threshold t_0. The value t_0 itself grows rapidly with n.

At present, the most successful attack on subset sum is described in [26, 25]. It combines the L^3 algorithm with a branch-and-bound search for the shortest vector and search pruning heuristics. Algorithms of this kind can be expected to solve subset sum problems for all values of ℓ up to about $n = 100$. Based on our own experiments, we observe that as n is increased far beyond $100 - 200$ the behavior of the algorithms becomes qualitatively similar to that of L^3 for $n > 40$: The shortest vectors are found only if ℓ is sufficiently larger (or smaller) than n. In practice, all known attacks break down at n around 200 for the more difficult SUBSET SUM problems (ℓ not much larger than n). Furthermore, the attacks do not appear to profit significantly from the fact that only κ-subset sums (as opposed to arbitrary subset sums) have to be solved, unless κ is extremely small. Typical applications of our generators correspond to $n \approx \ell > 500$ (the length of discrete log or factoring moduli) and $\kappa = 64$. The known methods appear to require excessive amounts of time to solve subset sum problems of this size. Furthermore, it is important to note that the problem arising in connection with our generators is not even a standard subset sum problem. A key property of our generators is the fact that the adversary sees only the subset sums (which are generated internally). The subset sum weights are secret. There is no reason to reveal them. It can be shown that the task of recovering the weights given enough subset sums is a well defined one:

HIDDEN SUBSET SUM PROBLEM: Given integers $M, b_1, \ldots, b_m \in \{0,1\}^\ell$, find integers $\alpha_1, \ldots, \alpha_n$ ($\alpha_j \in \{0,1\}^\ell$) such that each b_i $(1, \ldots, m)$ is some subset sum modulo M of $\alpha_1, \ldots, \alpha_n$.

It appears that attacks on the generator need to cope with the complications due to the hidden weights. We defer a detailed discussion to the full version of the paper. We can show that this problem is at least as hard as the standard subset sum problem and thus, by the results of Ajtai, as hard as worst case lattice approximation problems. Our experiments and discussions with other researchers lead us to conjecture that it is potentially harder. Our results have applicability to general lattice based problems (like Ajtai's), random affine codes (or syndrome decoding) based systems [15, 1], and polynomials.

Conventions and outline: Given an integer x, let $|x|$ denote its length in bits. We use $\phi(N)$ to denote the size of the multiplicative group $\mathbb{Z}_N^*$. We use $[a, b]$ to denote the set $\{a, \ldots, b\}$, where a, b are integers.

Section 2 describes our generation scheme for pairs of the form $(x, x^e \bmod N)$. Section 3 describes and analyzes applications of this scheme, in particular RSA-like public-key systems and Shoup's 2^m-th root identification scheme. Section 4 describes the generator for pairs of the form $(x, g^x \bmod p)$ and its application in several protocols, including signature schemes. Section 5 discusses parameter choices and presents some performance results.

Due to space restrictions, most proofs and parts of the discussion have been omitted in this version. A full version of the paper will be available shortly.

2 The Generator for $(x, x^e \bmod N)$

2.1 The Basic Generator

The generators in this section are targeted towards speeding up protocols in which a party must generate a random $x \in \mathbb{Z}_N^*$ and $x^e \bmod N$ (below all computations, if not specified otherwise, are done modulo N), for a given $N = pq$, where p, q are primes, and $e \in \mathbb{Z}^+$ of length m. The generator has two parameters n, κ. Its outputs correspond to κ-subsets of a set of n random numbers α_i. We choose the parameters n, κ such that $\binom{n}{\kappa}$ (the number of possible κ-subsets) is sufficiently large to make the corresponding subset product problem intractable, and to make birthday attacks infeasible (cf. Section 5).

Generation algorithm G:
Preprocessing Step: Generate n random integers $\alpha_i \in \mathbb{Z}_N^*$. Compute $\beta_i = (\alpha_i)^e \bmod N$ for each i and store both α_i's and β_i's in a table.
Whenever a pair (x, x^e) is needed: Randomly generate $S \subset [1, n]$ such that $|S| = \kappa$. Let $k = \prod_{i \in S} \alpha_i \bmod N$. Let $K = \prod_{i \in S} \beta_i \bmod N$. Return (k, K) as the result of G.

Obviously, $K = k^e$. The preprocessing takes $O(mn)$ multiplications. Subsequently, each output (x, x^e) is computed with only 2κ multiplications. Similar ideas have been used previously in [1]. G can be used in many schemes and analyzed without further assumptions. We now present computationally simple modifications of the generator that improve its performance. We will state the security proofs for the simple generator. They can be easily adapted for the full generator.

Remark 1. Note that the table is internal to the user, and no external updates or synchronizations for the schemes we discuss are needed.

Remark 2. We assume throughout the paper that $\binom{n}{\kappa}$ is large enough so that the first ℓ outputs of the generator are distinct with high probability. This simplifies matters and avoids repetitive statements.

2.2 The Full Generator: Introducing a Random Walk

Our full generator combines a random walk on expanders based on Cayley Graphs on Abelian groups with the outputs of G. For standard references on expanders, rapid mixing and their set hitting properties see [20]. Notable are "Chernoff Bounds" for random walk sequences that allow remarkable statements about passing general statistical (e.g. arbitrary moment) tests on the output numbers. For our applications, we need expanders on *specific domains* over which discrete log and factoring are defined. Fortunately these graphs exist: It is sufficient to select a small set of generators at random. The resulting Cayley graph is an expander with high probability [2]. A graph H is called a c-expander if for every set of vertices S, $|\Gamma(S)| > c|S|(1-|S|/|H|)$, where $\Gamma(S)$ is the set of all neighbors of S. The Cayley graph $X(A,S)$ of a group A with respect to the set S of elements in the group is the graph whose set of vertices is A and whose set of edges is the set of all unordered pairs $\{\{g,gs\} : g \in A, s \in S\}$. It is shown in [2] that for every $1 > \varepsilon > 0$ there exists a $c(\varepsilon) > 0$ such that the following holds. Let A be a group of order N, and let S be a random set of $c(\varepsilon)\log N$ elements of A. Then the Cayley graph $X(A,S)$ is an ε-expander almost surely.

Generation algorithm G_{exp}: Let N, e, n, κ be as in G. There is an additional parameter $n_e = c\log\phi(N)$ for some constant c (e.g. $c = 0.5$ or $c = 1$).
Preprocessing Step: Generate n random integers $\alpha_i \in \mathbb{Z}_N^*$. Compute $\beta_i = \alpha_i^e$ for each i and store the α_i's and β_i's in a table.

Generate a random subset $S_e \subset \mathbb{Z}_N^*$ of size n_e (by the result of [2] $X(\mathbb{Z}_N^*, S_e)$ is an expander almost surely). For each $d_i \in S_e$, $1 \le i \le n_e$, set $D_i = d_i^e$ and store (d_i, D_i) in a table. Set r to a random element of $\mathbb{Z}_N^*$ and R to r^e.
Whenever a pair (x, x^e) is needed: Randomly generate $S \subset [1,n]$ such that $|S| = \kappa$. Select a random $j \in [1, n_e]$. Set $r := r \cdot d_j$ and $R := R \cdot D_j$. Let $k = r \cdot \prod_{i\in S} \alpha_i$. Let $K = R \cdot \prod_{i\in S} \beta_i$ and return (k, K) as the result of G_{exp}.

2.3 Randomness Properties of the Full Generator

Theorem 1. *(Resistance to Birthday attacks) The expected number of repetitions in a run of length ℓ is at most*

$$\frac{\binom{\ell}{2}}{\phi(N)} + \frac{\ell}{\binom{n}{\kappa}}\left(\frac{1}{1-2^{-c}} + \frac{1}{c}\kappa\log n\right) \tag{1}$$

for some constant c and sufficiently large N, n.

The proof is outlined in the appendix. The first term of (1) is the expected number of repetitions in an ideal sequence whose elements are independent random elements of $\mathbb{Z}_N^*$ and is negligible for feasible runs of the generator. The point to note is that the second term – which represents the additional collisions due to our generator – contains ℓ only as a linear factor. In contrast, the goal of a birthday attack is to increase the expected number of collisions proportional to ℓ^2. The constant c depends on the parameters of the expander, which can easily be chosen such that $c \approx 1$.

Achieving similar security against birthday attacks without the expander would require κ to be almost doubled reducing the speed by a factor of 2. At the expense of the additional storage for the (d_i, D_i) table, the expander component requires only two additional multiplications per output. In addition, it improves the randomness properties of the output numbers substantially (see [1]). Here it makes the outputs look like subset sums of size approximately 2κ. which can heuristically be seen by dividing two successive outputs.

3 RSA-Based Schemes

Our generators do not speed up RSA-based schemes in a general way, but open some new possibilities. The generator cannot be applied directly to RSA since it requires exponentiation of a given message rather than random one. We analyze versions of the following scheme, in which f is an appropriately chosen function. The schemes defined in this section will use either $f(x) = x$ or consider f as a random oracle. Attacks on low exponent RSA due to Håstad and Coppersmith suggest (see [7]) that the RSA function should be applied to a suitably randomized version of the input (plaintext), rather than the input itself.
Key generation: The public and private keys (e, d), as well as the modulus N are generated as in RSA. That is, a party generates two large random primes p, q, sets $N = pq$, computes e, d such that $ed \equiv 1 \bmod \phi(N)$, and publishes e and N.
Encryption: A message M is encrypted as $E(M) = (x^e, f(x) \oplus M)$, where (x, x^e) is an output of G.
Decryption: Given a pair a, b, output $D(a, b) = f(a^d) \oplus b$.

Our generator speeds up encryption, when the encryption exponent is large. We also discuss how the decryption times may be reduced. In comparison with ordinary RSA, the length of the ciphertext is doubled. We can prove that the scheme is secure by relating it to RSA and by analyzing it in the random oracle model.

3.1 Random Oracle Model

The random oracle model (e.g. [6, 22] and references therein) provides an idealized view of cryptographic hash functions. Protocols are allowed to use a random oracle, i.e. a publicly available function f whose values $f(x)$ are determined independently at random for each input x. In the absence of better analysis, one often extrapolates the security results from random oracles to existing hash functions by using a heuristic assumption that some secure hash function behaves like a random oracle. We view this as a strong assumption warranting caution, but such analysis seem to yield better results than without it.

In the random oracle model, we choose $f(x) = h(x)$, where $h(x)$ is a random function. Let the parameters of the generator be such that repetitions in the output sequence are unlikely (e.g. see Thm. 1).

Theorem 2. *Let all elements in the sequence $x_1, \ldots, x_k$ ($k \geq 1$) be different. For $k > 0$, distinguishing $E(M_k)$ from a random string, given $(M_1, E(M_1)), \ldots,$ $(M_{k-1}, E(M_{k-1})), M_k$ is as hard as inverting $x \mapsto x^e$ (i.e. RSA) on random inputs.*

3.2 $x^e, x \oplus M$

This subsection considers the simplest case, i.e. $f(x) = x$. We describe some practical implications of our scheme. We omit our security analysis due to space restrictions.

Small decryption exponents: Use of G speeds up encryption for any exponent. Decryption still requires an exponentiation with the decryption exponent d. Decryption can be speeded up by choosing d and e such that d is small. The most efficient attack against RSA with small decryption exponent is the Diophantine approximation method of Wiener [29]. The attack breaks down if $d \geq N^{1/4+\delta}$ ($\delta > 0$), or if e is replaced by $e' = e + r\phi(N)$ such that $|e'| \geq 1.5\,|N|$, where r is a random number.

Quadratic Residues: $e = 2$ It is known that given enough bits of the plaintext, one can uniquely pick one of the square roots for decryption. The proof of the following theorem is outlined in the appendix.

Theorem 3. *Fix some ℓ, let $e = 2$, and let $I \leftarrow G'(.)$ be a run of ℓ outputs from the generator. Assume that there exists an algorithm that, given I, computes the square root (modulo N) of the next output of G' with a noticeable success rate ε . Then there is an algorithm to compute square roots of arbitrary quadratic residues modulo N in expected time $1/\varepsilon$. Furthermore, there is an algorithm to factor N.*

3.3 2^m-th Root Identification Scheme

In this subsection, we show that using G in the 2^m-th root identification scheme [27] preserves its full security. We recall the scheme first: N is the product of two randomly selected primes of equal length, both congruent to 3 mod 4. $q = 2^m$ is the exponent, for some sufficiently large m. $a \in \mathbb{Z}_N^*$ is the private key and $b = a^q \bmod N$ is the public key. If Alice (Prover) wants to prove her identity to Bob (Verifier), she chooses $k \in \mathbb{Z}_N^*$ at random, computes $x = k^q$, and sends x to Bob. Bob checks that $x \neq 0$, chooses $r \in [0, q-1]$ at random and sends r to Alice. Alice computes $y = ka^r$ and sends y to Bob. Bob accepts if $y^q = xb^r$.

Note that an authentication scheme can be converted into a signature scheme by replacing the verifier's challenge by a hash of the message [24]. It can be seen that, if an authentication scheme is secure when used with G, then so is the corresponding signature scheme.

Theorem 4. *Let $n \geq c_1 \log N$ and $n \gg \kappa \geq c_2$ for some constants c_1 and c_2. If factoring is intractable then the 2^m-th root scheme is secure against active attacks when the prover uses G to generate its first-round messages.*

The omitted proof can be adapted to show security against active attacks for the generalized Ong-Schnorr authentication scheme when used with G ([27] shows the security for the case of fully independent random numbers). We defer the details to the final version of this paper.

4 Discrete Log Based Schemes

In this section, we present a modification of our generation scheme which makes it suitable for speeding up protocols based on the discrete logarithm problem.

These include ElGamal, DSS, and Schnorr signatures, Diffie-Hellman key exchange, and ElGamal encryption.

4.1 Generators

All versions of the generator presented in Section 2 can be translated into the discrete logarithm framework. Due to space limitations, we present only the basic version G'. The corresponding full version including an expander G'_{exp} is analogous to G_{exp}. Thm. 1 is easily adapted to G'_{exp}.

Let p be a prime of length m, and let $g \in Z_p^*$. The task is to generate a random k and compute $g^k \bmod p$ – as required by many protocols. In the remainder of this section, all operations are done modulo p. Again, the purpose of the generator is to speed up the modular exponentiation.
Generation algorithm G':
Preprocessing Step: Generate n random integers $\alpha_i \in \mathbb{Z}_{\mathrm{ord}(g)}$. Compute $\beta_i = g^{\alpha_i}$ for each i and store both α_i's and β_i's in a table.
Then, whenever a pair (x, g^x) is needed: Randomly generate $S \subset [1, n]$ such that $|S| = \kappa$. Let $k = \sum_{i \in S} \alpha_i \bmod \mathrm{ord}(g)$. If $k = 0$, stop and start again. Let $K = \prod_{i \in S} \beta_i$ and return (k, K) as the result of G.

4.2 Speeding up Discrete-log-based Schemes

Our first theorem outlines a main aspect of our generator, which stems from the fact that the precomputation tables are chosen by the generator and kept secret.

Theorem 5. *Fix some ℓ and let $I := (g^{k_i})_i \leftarrow G(.)$ be a run of ℓ outputs from the generator. Assume that there exists an algorithm that, given I, computes the discrete log of the next output of G with success rate ε . Then there is an algorithm to compute discrete log on arbitrary inputs in expected time $O(1/\varepsilon)$.*

The proof is outlined in the appendix. Despite the small number of multiplications used in G, for all but a negligible fraction of the choices of the initial precomputation tables, computing the discrete log of any new output of the generator is as hard as solving the full discrete log problem, namely given *arbitrary* $y = g^x$ compute x. Note that the attack algorithm never sees the discrete log of any element from the list of its outputs. In practice this means that it suffices to ensure that in any run of practical interest its outputs do not repeat. More complicated issues will arise when the discrete logs are used to generate some outputs. This is the case in many signature schemes.

4.3 Signature Schemes

Our generators can be used to speed up several signature schemes. The signature schemes we consider use pairs (k, g^k) in two contexts. For example, in the ElGamal scheme, a signer generates one pair $(x, y = g^x)$, publishes y and keeps x secret. This pair is generated *only once* and corresponds to the generation of a private and a public key. We do *not* use our generator to speed up the generation of this pair. Our generator is only used to speed up the generation of the random pairs (k, g^k) which are needed every time a message is to be signed. However, given y, a third table containing $(y^{\alpha_i})_{i \le n}$ can be added to our generator. Thus, the computation of y^k can also be speeded up. This does not raise further security issues.

Let $\sigma(M, k)$ be some discrete-log based signature of message M using a random number k. Suppose there exists an attack algorithm A such that $A(y, \bar{M}, I) = \sigma(\bar{M}, \bar{k})$ for some $\bar{k}$, where $I = \{(M_i, \sigma(M_i, k_i))\}_{i=1}^{\ell}$, with k_i generated by G. Note that A does not query the signing algorithm. It is simply given a sequence of signatures and messages. The messages given to A can be arbitrary. This corresponds to a *known message* attack.

Theorem 6. *The following signature schemes are secure against known-message attacks when used with G': ElGamal, DSS and Schnorr.*

Security Against Adaptive attacks: Theorem 6 does not does not cover adversaries who can choose their messages adaptively, depending on previous messages and their signatures.

Our first approach to making the scheme resistant against adaptively chosen message attacks is to make the outputs of G very close to uniform and independent. This can be achieved by choosing the parameters of the generator appropriately. It can be shown that the new schemes that use G for generation are secure if and only if the original schemes are. Details omitted.

Our second approach is to use cryptographic pseudo-randomness in the sense of Blum-Micali and Yao. As a motivation, consider the ElGamal signature scheme. Unlike in Thm. 5, an attacker sees not only numbers of the form $r = g^k$ but additional information that depends on k. Namely, the triple (r, M, g^x) and $k^{-1}(M - xr) \bmod p - 1$.

Theorem 7. *If the sequence of k is cryptographically pseudo-random, then the speeded-up versions of the following schemes are secure against polynomial time adaptive attacks: ElGamal Signatures, DSS, Schnorr authentication and signatures.*

The attacker does not see the subset sums directly. For example, in the case of ElGamal signatures, he sees only $k^{-1}(M - xr)$, a multiple of the inverse of a hidden number k. It is not clear how to accommodate this in lattice attacks. It is worth noting that de Rooij's attack succeeds in recovering the hidden number (the signer's secret key) using tight correlations among consecutive outputs. But in our case the numbers are chosen from a large set every time, and this set itself moves over the entire group with a mixing rate which is logarithmic in the group size (since expander random walks mix rapidly).

4.4 Diffie-Hellman Key Exchange and ElGamal Encryption

Diffie-Hellman key exchange is defined as follows. Alice generates a random $a \in \mathbb{Z}_{\mathrm{ord}(g)}$ and sends g^a to Bob. Bob generates a random $b \in \mathbb{Z}_{\mathrm{ord}(g)}$ and sends g^b to Alice. Now they share a secret $g^{ab} = (g^b)^a = (g^a)^b$. Alice and Bob can use G to generate (a, g^a) and (b, g^b), respectively.

ElGamal encryption [14] is defined as follows. x is the secret key, $y = g^x$ is the public key. A message M is encrypted as $E(M, k) = (g^k, My^k)$. We speed up the scheme by using G to generate k and g^k for each encryption. G is not used to compute x and y.

Lemma 1. *(a) Diffie-Hellman key exchange with G used to generate (a, g^a) and (b, g^b) is as secure as Diffie-Hellman key exchange with independent a's and b's.*

(b) ElGamal encryption with G used to generate (k, g^k) is secure against ciphertext-only attacks if standard ElGamal encryption is secure.

The proof is similar to the proof of Thm. 5. One can cope with adaptive attacks by achieving output distributions that are statistically close to being uniform. Details are omitted due to space constraints.

5 Performance Results

The time and storage requirements as well as the security of our generators depend on the choices of the parameters n, κ, n_e. For the purpose of making direct performance comparisons with existing algorithms and based on our analysis, we consider concrete parameter choices for two broad classes of applications:

If the security of the protocol using our generator depends on the hardness of the hidden subset sum problem (e.g. adaptive attacks against signature schemes), the parameters should be chosen such that solving the hidden subset sum problem is infeasible. If the security of the protocol using our generator does not depend on the hardness of the hidden subset sum problem (e.g. Diffie-Hellman key exchange), it is only necessary to choose the parameters large enough to avoid birthday attacks. In this case, the number of multiplications per exponentiation can be made extremely small.

Table 1 gives the storage requirements and average number of multiplications using various methods to generate random pairs $(x, g^x \bmod p)$ and $(x, x^e \bmod N)$ for 512-bit numbers. For protocols of the first kind (hardness of subset sum is important), it appears that $n = n_e = 512$ and $\kappa = 64$ (or $\kappa = 32$ for the expander version) should provide sufficient security. For certain protocols of the second kind, it appears that κ can be chosen to be as small as 6 or 16 and $n = 256$. Table 1 displays the resource requirements for these parameter choices as well as those for the algorithms of [5, 10, 19] and square-and-multiply. For the algorithms of [5, 19], we display examples with small and large storage requirements. Using comparable amounts of memory, our generators need fewer multiplications than the other algorithms, especially in the case of $G_{\text{exp}}, G'_{\text{exp}}$.

6 Conclusions

We have suggested methods for speeding up public key schemes which are based on discrete log and factoring. Our methods focus on the generation of distributions of pairs of the form (k, g^k) $(g^k \in Z_p^*)$ or $(x, x^e) \in (Z_n^*)^2$. We have analyzed the security of their use in several example schemes.

In the process of this analysis, we had to consider versions of a hidden lattice problem which seems to be of interest in its own right and whose hardness should be studied further. An interesting question is if the apparent non-linearity of this problem is an inherent property. An extension of this problem to other (i.e. non-lattice) domains will be presented in a future work.

Acknowledgments: We thank Arjen Lenstra for his generous discussions. We also thank D. Boneh, C.P. Schnorr, D. Coppersmith, D. Bienstock and R. Kannan for discussions on methods for attacking hidden subset sum problems.

Table 1. A comparison of methods of generating pairs $(x, g^x \bmod p)$ and $(x, x^e \bmod N)$ for $|p| = 512$, $\text{ord}(g) = 512$, $|N| = 512$, $|e| = 512$. Storage requirements are in 512-bit numbers. Times are in multiplications per exponentiation.

	$(x, g^x \bmod p)$		$(x, x^e \bmod N)$	
	Storage	Time	Storage	Time
Square-and-multiply	0	766	0	766
Brickell et al. [5]	512	100	not applicable	
Brickell et al. [5]	10880	64	not applicable	
Lim and Lee [19]	317	100	not applicable	
Lim and Lee [19]	13305	52	not applicable	
de Rooij [10]	64	128	not applicable	
G $(n = 512, \kappa = 64)$	1024	63	1024	126
G_{exp} $(n = n_e = 512, \kappa = 32)$	2048	33	2048	66
G $(n = 256, \kappa = 16)$	512	15	512	30
G_{exp} $(n = n_e = 256, \kappa = 6)$	1024	7	1024	14

References

1. W. Aiello, S. Rajagopalan, and R. Venkatesan. Design of practical and provably good random number generators. In *Proceedings of the 6th Annual Symposium on Discrete Algorithms*, pages 1–9, San Francisco, January 1995. ACM Press. (also to appear in Journal of Algorithms).
2. N. Alon and Y. Roichman. Random Cayley graphs and expanders. *Random Structures and Algorithms*, 5, 1994.
3. D. Boneh and R. Venkatesan. Breaking RSA may not be equivalent to factoring. Eurocrypt '98, this proceedings.
4. E. Brickell. Solving low density knapsacks. In *Proceedings of Crypto'83*, pages 25–37, New York, 1984. Plenum Press.
5. E. F. Brickell, D. M. Gordon, K. S. McCurley, and D. B. Wilson. Fast exponentiation with precomputation. In R. A. Rueppel, editor, *Advances in Cryptology:*

EUROCRYPT '92, volume 658 of *Lecture Notes in Computer Science*, pages 200–207, Berlin, 1993. Springer-Verlag.

6. R. Canetti. Towards realizing random oracles: Hash functions that hide all partial information. In *Advances in Cryptology - Crypto'97*, Lecture Notes in Computer Science, 1997.
7. D. Coppersmith. Small solutions to polynomial equations and low exponent RSA vulnerabilities. *Journal of Cryptology*, 10(4), 1997.
8. M. J. Coster, A. Joux, B. A. LaMacchia, A. M. Odlyzko, C. P. Schnorr, and J. Stern. Improved low-density subset sum algorithms. In *Computational Complexity 2*, pages 111–128. Birkhäuser-Verlag, Basel, 1992.
9. P. J. N. de Rooij. On the security of the Schnorr scheme using preprocessing. In D. W. Davies, editor, *Advances in Cryptology: EUROCRYPT '91*, volume 547 of *Lecture Notes in Computer Science*, pages 71–80, Berlin, 1991. Springer-Verlag.
10. P. J. N. de Rooij. Efficient exponentiation using precomputation and vector addition chains. In A. De Santis, editor, *Advances in Cryptology: EUROCRYPT '94*, volume 950 of *Lecture Notes in Computer Science*, pages 389–399, Berlin, 1994. Springer-Verlag.
11. P. J. N. de Rooij. On Schnorr's preprocessing for digital signature schemes. *Journal of Cryptology*, 10(1):1–16, 1997.
12. W. Diffie and M. Hellman. New directions in cryptography. *IEEE Transactions on Information Theory*, 22:644–654, 1976.
13. *Digital Signature Standard.* National Bureau of Standards FIPS Publication 186, 1994.
14. T. ElGamal. A public key cryptosystem and a signature scheme based on discrete logarithms. *IEEE Trans. Inform. Theory*, 31:469–472, 1985.
15. J. Fischer and J. Stern. An efficient pseudo-random generator provably as secure as syndrome decoding. In U. Maurer, editor, *Advances in Cryptology: EUROCRYPT '96*, volume 1070 of *Lecture Notes in Computer Science*, pages 245–255, Berlin, 1996. Springer-Verlag.
16. R. Impagliazzo and M. Naor. Efficient cryptographic schemes provably as secure as subset sum. *Journal of Cryptology*, 9(4):199–216, 1996.
17. J. Lagarias and A. Odlyzko. Solving low density subset sum problems. *Journal of the ACM*, 32, 1985.
18. A. Lenstra, H. Lenstra, and L. Lovász. Factoring polynomials with rational coefficients. *Mathematische Annalen*, 261:513–548, 1982.
19. C. H. Lim and P. J. Lee. More flexible exponentiation with precomputation. In *Advances in Cryptology - Crypto'94*, Lecture Notes in Computer Science, 1994.
20. R. Motwani and P. Raghavan. *Randomized Algorithms.* Cambridge University Press, 1995.
21. V. I. Nechaev. Complexity of determinate algorithm for the discrete logarithm. *Mathematical Notes*, 55(2):165–172, 1994.
22. D. Pointcheval and J. Stern. Security proofs for signature schemes. In U. Maurer, editor, *Advances in Cryptology - Eurocrypt'96*, Lecture Notes in Computer Science, 1996.
23. C. P. Schnorr. Efficient identification and signatures for smart cards. In G. Brassard, editor, *Advances in Cryptology: CRYPTO '89*, volume 435 of *Lecture Notes in Computer Science*, pages 239–252, Berlin, 1990. Springer-Verlag.
24. C. P. Schnorr. Efficient signature generation by smart cards. *J. Cryptology*, 4:161–174, 1991.

25. C. P. Schnorr and M. Euchner. Lattice basis reduction: Improved practical algorithms for solving subset sum problems. *Mathematical Programming*, 66:181–199, 1994.
26. C. P. Schnorr and H. Hörner. Attacking the Chor-Rivest cryptosystem by improved lattice reduction. In L. Guillou and J. Quisquater, editors, *Advances in Cryptology - Eurocrypt'95*, Lecture Notes in Computer Science, 1995.
27. V. Shoup. On the security of a practical identification scheme. In Ueli Maurer, editor, *Advances in Cryptology: EUROCRYPT '96*, volume 1070 of *Lecture Notes in Computer Science*, pages 344–353, Berlin, 1996. Springer-Verlag.
28. V. Shoup. Lower bounds on discrete logarithms and related problems. In *Advances in Cryptology - Eurocrypt'97*, Lecture Notes in Computer Science, 1997.
29. M. J. Wiener. Cryptanalysis of short RSA secret exponents. *IEEE Transactions on Information Theory*, 36(3):553–558, 1990.

A Appendix

A.1 Proof of Theorem 1

The following theorem is a version of results obtained in [1].

Theorem 8. *The probability that any particular number output by the full generator repeats after exactly m steps is at most*

$$\min\left\{\frac{1}{\binom{n}{\kappa}}, \frac{1}{\phi(N)} + 2^{-cm}\right\}$$

(for some constant $c > 0$).

If there exists an integer $m < \ell$ such that $1/\phi(N) + 2^{-cm} \leq 1/\binom{n}{\kappa}$ then let δ be the smallest such integer. Otherwise, let $\delta = \ell$. Let the random variable C denote the number of collisions. Then

$$\begin{aligned} \mathrm{E}C = \sum_{ij} \Pr(x_i = x_j) &\leq \sum_{i<j; j-i<\delta} \frac{1}{\binom{n}{\kappa}} + \sum_{i<j; j-i\geq\delta} \left(\frac{1}{\phi(N)} + 2^{-c(j-i)}\right) \\ &< \frac{\binom{\ell}{2}}{\phi(N)} + \ell\delta\left(\frac{1}{\binom{n}{\kappa}} - \frac{1}{\phi(N)}\right) + \sum_{i<j; j-i\geq\delta} 2^{-c(j-i)} , \end{aligned} \tag{2}$$

where x_i is the i-th element in the output sequence and the sums go over all ordered pairs (i, j) such that $1 \leq i < j \leq \ell$ and either $j - i < \delta$ or $j - i \geq \delta$.

By the definition of δ, we obtain $\delta \leq \lceil -\log D/c\rceil$, where $D = \binom{n}{\kappa}^{-1} - \phi(N)^{-1}$. For sufficiently large $\binom{n}{\kappa}$, the second term of (2) is at most

$$\ell\delta D \leq \ell D \lceil \log(1/D)/c \rceil < \frac{\ell}{c}\frac{1}{\binom{n}{\kappa}} \log\binom{n}{\kappa} ,$$

because the function $x \log(1/x)$ is increasing for sufficiently small $x > 0$. Concerning the third term of (2), it is easily seen that

$$\sum_{i<j; j-i \geq \delta} 2^{-c(j-i)} < \ell \frac{2^{-c\delta}}{1-2^{-c}} < \frac{\ell}{\binom{n}{\kappa}} \frac{1}{1-2^{-c}} \; ,$$

as $2^{-c\delta} < \binom{n}{\kappa}^{-1}$. The theorem follows by combining these bounds with (2).

A.2 Proof of Theorem 5

Suppose it is possible to compute the discrete log of an output of G after seeing a sequence of ℓ outputs. In other words, suppose there exists $i \in [1, \ell]$ and an algorithm A such that for $I = \{g^{k_j}\}_{j=0}^{\ell}$ generated by G, $A(I) = k_i$. Without loss of generality we can assume $i = \ell$. Let A's success rate be ε.

We construct an algorithm B^A such that, given *any* $y = g^x$, $B^A(y) = x$ with success rate ε. B^A would work as follows. Generate random κ-sized subsets S_j, for $1 \leq j \leq \ell$. Let h be a random element of S_ℓ. Let r be a random number. Set β_h to $g^r g^x$ and set α_h to undefined. Now, for $i \in [1, n] \setminus \{h\}$, set α_i to be uniformly distributed independent (both of each other and of r) random numbers, and set $\beta_i = g^{\alpha_i}$. Let $K_j = \prod_{i \in S_j} \beta_i$. Let $z = A(\{K_j\}_{j=1}^{\ell})$. Compute $X = z - r - \sum_{i \in S_\ell \setminus \{h\}} \alpha_i$. Return X.

Next, we show that the K_j's produced by B^A have the correct distribution. Since r is uniformly distributed and independent of β_i for $i \in [1, n] \setminus \{h\}$, and since the β_i's (for $i \in [1, n] \setminus \{h\}$) are uniformly distributed and independent of each other, β_i for all $i \in [1, n]$ are uniformly distributed and independent. The S_j's for $1 \leq j \leq \ell$ are also random and independent. Since the distribution of the outputs of G depends only on the distributions of the β's and S's, the sequence $\{K_j\}$ generated by B has the same distribution as the output of G with completely random tables. Hence A has success rate ε on such input. Suppose that A is successful. By assumption on A we have $g^z = K_\ell = \prod_{i \in S_\ell} \beta_i = \beta_h \prod_{i \in S_\ell \setminus \{h\}} \beta_i$,

$$g^X = g^{z - r - \sum_{i \in S_\ell \setminus \{h\}} \alpha_i} = \frac{g^z}{g^r \prod_{i \in S_\ell \setminus \{h\}} g^{\alpha_i}} = \beta_h \frac{\prod_{i \in S_\ell \setminus \{h\}} \beta_i}{g^r \prod_{i \in S_\ell \setminus \{h\}} \beta_i} = \beta_h / g^r = g^x.$$

It follows that $X = x$, and that A would find the discrete log in expected $1/\varepsilon$ steps.

A.3 Proof of Theorem 3

The proof of the first part is similar to the proof of Thm. 5. Indeed both theorems are special cases of a more general fact about groups. The statement about factoring follows from the well known reduction from factoring to quadratic residues.