A Pre-Computation Scheme for Speeding Up Public-Key Cryptosystems

by

Victor Boyko

B.A., Mathematics and Computer Science (1996)
New York University

Submitted to the Department of Electrical Engineering and Computer Science
in partial fulfillment of the requirements for the degree of

Master of Science in Electrical Engineering and Computer Science

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

June 1998

Author .....
Department of Electrical Engineering and Computer Science
May 22, 1998

Certified by .....
Shafi Goldwasser
Professor of Computer Science
Thesis Supervisor

Accepted by .....
Arthur C. Smith
Chair, Department Committee on Graduate Students

A Pre-Computation Scheme for Speeding Up Public-Key Cryptosystems

by

Victor Boyko

Submitted to the Department of Electrical Engineering and Computer Science
on May 22, 1998, in partial fulfillment of the
requirements for the degree of
Master of Science in Electrical Engineering and Computer Science

Abstract

This thesis presents 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. The schemes are most suited for server applications. The thesis also presents security analyses of the schemes using standard assumptions. The 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. The constructions use random walks on Cayley (expander) graphs over Abelian groups.

Thesis Supervisor: Shafi Goldwasser
Title: Professor of Computer Science

Acknowledgments

I would like to start by expressing my deepest gratitude to Shafi Goldwasser, my thesis advisor, for her continued advice and fruitful discussions. She has been very helpful and supporting throughout my time at MIT, and I look forward to further work with her.

The work for this thesis was performed in part at Bellcore under the direction of and in collaboration with Ramarathnam Venkatesan. I would like to thank Marcus Peinado for investigating the hidden subset sum problems related to our generators, and for collaborating on the conference version of our paper. I would also like to thank Ronald Rivest and Arjen Lenstra for their help, advice, and discussions.

Finally, I would like to thank my parents for making me possible, and for their constant love and encouragement.

Contents

1 Introduction **7**

1.1 Models of Analysis 9

1.2 RSA-based schemes 9

1.3 Lattice attacks on subset sum problems 10

1.4 Conventions and outline 11

2 The Generator for $(x, x^e \bmod N)$ **12**

2.1 The basic generator 12

2.2 Randomness Properties of the Basic Generator 13

2.2.1 Achieving Statistical Indistinguishability 14

2.3 The Full Generator: Introducing a Random Walk 16

2.4 Randomness Properties of the Full Generator 17

3 RSA-Based Schemes **19**

3.1 Random Oracle Model 20

3.2 $x^e, x \oplus M$ 23

3.3 Small decryption exponents 24

4 Discrete Log Based Schemes **25**

4.1 Generators 25

4.2 Speeding up Discrete-log-based Schemes 26

4.3 Signature Schemes 28

4.3.1 ElGamal signatures 28

4.3.2 DSS signatures 30
4.3.3 Schnorr signatures 31
4.4 Diffie-Hellman Key Exchange 32
4.5 ElGamal Encryption 33

5 Performance results **35**

List of Tables

5.1 A comparison of methods of generating pairs $(x, g^x \bmod p)$ and $(x, x^e \bmod N)$ for $|p| = 512$, $|\mathrm{ord}(g)| = 512$, $|N| = 512$, $|e| = 512$. 36

Chapter 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 [29, 11, 17] 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 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 [29, 11, 17] (depending on the parameter choices; cf. Chapter 5). This improvement is even more pronounced when compared to square-and-multiply. Note that the algorithms of [29, 11, 17] 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 [13], ElGamal and DSS signatures [13, 14], Schnorr's schemes for authentication and signatures [25, 26], and versions of the RSA [22] 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. 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 thesis 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 [29, 11, 17]. A scheme for fast generation of pairs of the form $(x, g^x \bmod p)$ was proposed by Schnorr [25] 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 [26]. This version was also broken by de Rooij [10]. 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.

1.1 Models of Analysis

Instead of using a restricted adversary model, such as the 'black box model' of Nechaev [20] or Shoup [27], 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.

1.2 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 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 [28] (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.

1.3 Lattice attacks on subset sum problems

Subset sum constructions have been so successfully attacked by lattice reduction [16] based methods [5, 15, 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 [24, 23]. 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 appears that attacks on the generator need to cope with the complications due to the hidden weights. For further discussion of the hidden subset sum problem, the reader is referred to [4].

1.4 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.

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

Chapter 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. Chapter 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.*

2.2 Randomness Properties of the Basic Generator

Claim 1. *The outputs of G with distinct subsets S are pairwise independent.*

Proof. For a set S, let $s(S) = \prod_{i \in S} \alpha_i$. Let S_1 and S_2 be any subsets of size κ, such that $S_1 \neq S_2$. Denote $x_j = \prod_{i \in S_j} \alpha_i$ for $j = 1, 2$. We show that x_1 is independent of x_2, i.e., for any y_1 and y_2, $\Pr[x_1 = y_1 \mid x_2 = y_2] = \Pr[x_1 = y_1]$, where the probability is over the choice of α_i. Since S_1 and S_2 are not equal, and they are of the same size, neither of them is the subset of the other. In particular, $S_1 - S_2 \neq \emptyset$. Let h be an element of $S_1 - S_2$.

$$\Pr[x_1 = y_1 \mid x_2 = y_2] = \Pr[\alpha_h = y_1 s(S_1 - \{h\})^{-1} \mid s(S_2) = y_2].$$

Since α_h is uniform and independent of all the variables appearing in the condition $s(S_2) = y_2$ and of all the variables in the right-hand side $y_1 s(S_1 - \{h\})^{-1}$, it follows that $\Pr[\alpha_h = y_1 s(S_1 - \{h\})^{-1} \mid s(S_2) = y_2] = 1/\phi(N)$. Also,

$$\Pr[x_1 = y_1] = \Pr[\alpha_h = y_1 s(S_1 - \{h\})^{-1}] = 1/\phi(N).$$

Thus $\Pr[x_1 = y_1 \mid x_2 = y_2] = \Pr[x_1 = y_1]$. □

Corollary 2. *$s : (\alpha_i)_{i=1}^n \times S \mapsto \prod_{i \in S} \alpha_i$ is a universal hash function.*

Claim 3. *Denote by $\mathcal{S}$ the set of all possible outputs of G for a particular choice of α_i, $1 \leq i \leq n$. Then (assuming that $\phi(N)$ is sufficiently large)*

$$E[|\mathcal{S}|] \geq \frac{\binom{n}{k}}{1 + O(\phi(N)^{-1}\binom{n}{k})},$$

where the expected value is over the choice of α_i, $1 \leq i \leq n$.

Proof. For a subset S, $|S| = \kappa$ let $n(S)$ denote the number of subsets $S' \neq S$, $|S'| = \kappa$, such that $s(S) = s(S')$. Let $n'(x)$ be the number of subsets S such that $s(S) = x$. Note that if $s(S) = x$, then $n'(x) = 1 + n(S)$. It is easy to see that

$$|\mathcal{S}| = \sum_{(x,x^e)\in\mathcal{S}} \frac{n'(x)}{n'(x)} = \sum_{S\subset[1,n],|S|=\kappa} \frac{1}{1+n(S)}.$$

For any subsets $S \neq S'$ of size κ, $s(S)$ and $s(S')$ are uniformly and independently distributed over $[1, \mathrm{ord}(g)]$. Therefore, $\Pr[s(S) = s(S')] = 1/\phi(N)$. It follows that for any such S, $E[n(S)] = (\binom{n}{k} - 1)\phi(N)^{-1}$. Assuming proper choice of parameters to ensure convergence, one gets $E[\frac{1}{1+n(S)}] \geq \frac{1}{1+O(E[n(S)])}$. We therefore have

$$E[|\mathcal{S}|] \geq \sum_{S\subset[1,n],|S|=\kappa} \frac{1}{1 + O(\phi(N)^{-1}\binom{n}{k})} = \frac{\binom{n}{k}}{1 + O(\phi(N)^{-1}\binom{n}{k})}.$$

□

We note that since the choice of S in each round of G is uniform and independent, even an information-theoretic adversary, given access to an unlimited number of outputs of G, will not be able to guess which element of $\mathcal{S}$ will be output next.

2.2.1 Achieving Statistical Indistinguishability

In this section we show that for sufficiently high values of parameters, the outputs of the generator are almost indistinguishable from random pairs (x, x^e).

We will need the following definitions from [18].

Definition 1 (Renyi entropy). *Let X and Y be independent and identically distributed random variables. Define the* Renyi entropy *of X as*

$$\mathrm{ent}_{\mathrm{Ren}}(X) = -\log(\Pr_{X,Y}[X = Y]).$$

It is easy to see that if $X \in_U \mathcal{S}$, i.e., X is uniformly distributed over a set $\mathcal{S}$, then $\mathrm{ent}_{\mathrm{Ren}} X = \log|\mathcal{S}|$.

Definition 2 (statistically distinguishable). *The* statistical distance *between distributions $\mathcal{D}_n$ and $\mathcal{E}_n$ over $\{0,1\}^n$ is*

$$\mathrm{dist}(\mathcal{D}_n, \mathcal{E}_n) = \frac{1}{2} \cdot \sum_{x \in \{0,1\}^n} |\Pr_X[X = z] - \Pr_Y[Y = z]|.$$

We say $\mathcal{D}_n$ and $\mathcal{E}_n$ are at most $\varepsilon(n)$-statistically distinguishable if $\mathrm{dist}(\mathcal{D}_n, \mathcal{E}_n) \leq \varepsilon(n)$.

Claim 4. *The outputs k_i of G are at most $2^{-(e+1)}$ statistically distinguishable from uniform, where $e = \frac{1}{2}(\log\binom{n}{\kappa} - m)$.*

Proof. Let $\mathcal{A}$ denote the table $(\alpha_i)_{i=1}^n$. $\mathcal{A}$ is chosen uniformly from among all possible tables (α_i). S is chosen uniformly from among all $\binom{n}{k}$ possible κ-subsets of $[1, n]$, so the Renyi entropy of S is $\log\binom{n}{k}$. It now follows from corollary 2 and the Smoothing Entropy Theorem [18] that $\langle s_{\mathcal{A}}(S), \mathcal{A}\rangle$ is at most 2^{-e+1} distinguishable from uniform. Therefore, $k_i = s_{\mathcal{A}}(S_i)$ are at most $2^{-(e+1)}$ statistically distinguishable from uniform. □

Thus the bigger $\binom{n}{k}$, the closer will the outputs of G be to uniform. Take, for instance, $m = 512$, $n = 1024$, and $\kappa = 167$ (in this case there is a factor of 4.5 speedup over square-and-multiply exponentiation). Then the output of G will be at most 2^{-70} statistically distinguishable from uniform, so no algorithm will be able to distinguish it from uniform in less than $\sim 2^{70}$ steps.

We emphasize that these high values of parameters need only be used if one wishes to have security that does not rely on computational assumptions.

2.3 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 [19]. 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].

In G the number of multiplications needed to generate a pair is κ. If κ is too small, the return time of G will be very short. In order to decrease κ while preserving randomness, we will make use of expanders:

Definition 3 (expander). *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.*

Definition 4 (Cayley graph). *The undirected Cayley graph $X(A, S)$ of a group A with respect to the set S of elements in A is the graph whose set of vertices is A and whose set of edges is the set of all unordered pairs $\{\{a, as\} : a \in A, s \in S\}$.*

The following result is shown in [2]:

Theorem 5. *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. For each $d_i \in S_e$, $1 \leq i \leq n_e$, 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}.

2.4 Randomness Properties of the Full Generator

Theorem 6. *(Resistance to Birthday attacks) The expected number of repetitions in a run of length ℓ is at most*

$$\frac{\binom{\ell}{2}}{\phi(N)} + \frac{\ell}{\binom{n}{\kappa}} \left(\frac{1}{1 - 2^{-c}} + \frac{1}{c} \kappa \log n \right) \tag{2.1}$$

for some constant c and sufficiently large N, n.

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

Theorem 7. *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

$$\begin{aligned} \mathrm{E}[C] &= \sum_{ij} \Pr(x_i = x_j) \leq \sum_{i<j; j-i<\delta} \frac{1}{\binom{n}{\kappa}} + \sum_{i<j; j-i \geq \delta} \left(\frac{1}{\phi(N)} + 2^{-c(j-i)} \right) \\ &< \frac{\binom{\ell}{2}}{\phi(N)} + \ell\delta \left(\frac{1}{\binom{n}{\kappa}} - \frac{1}{\phi(N)} \right) + \sum_{i<j; j-i \geq \delta} 2^{-c(j-i)} , \end{aligned} \tag{2.2}$$

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

By the definition of δ, we obtain $\delta \le \lceil -\log D/c \rceil$, where $D = \binom{n}{\kappa}^{-1} - \phi(N)^{-1}$. For sufficiently large $\binom{n}{\kappa}$, the second term of (2.2) is at most

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

because the function $x \log(1/x)$ is increasing for sufficiently small $x > 0$. Concerning the third term of (2.2), it is easily seen that

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

as $2^{-c\delta} < \binom{n}{\kappa}^{-1}$. The theorem follows by combining these bounds with (2.2). □

The first term of (2.1) 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.

Chapter 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 a 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.

- *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 $E(M) = E(M, x) = (x^e, f(x) \oplus M)$, where (x, x^e) is an output of G or G_{exp}.

- *Decryption:* Given a pair a, b, output $D(a, b) = f(a^d) \oplus b$.

We note that this scheme is actually quite close to the way that RSA encryption is used in practice. In most applications, RSA encryption is applied not to the message itself, but rather to a random session key. The session key is then used in conjunction with a symmetric encryption scheme to encrypt the message. In our construction x is the session key and $M \mapsto f(x) \oplus M$ plays the role of a symmetric encryption function. It is interesting to note that for a stream cipher, such as the commonly

used RC4 algorithm, the symmetric encryption transformation has exactly the form $M \mapsto f(x) \oplus M$, where f is a pseudorandom number generator. This can be modeled by considering f as a random oracle.

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, 21] 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 seems 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. We will assume that there are no repetitions in the output sequence of the generator. This is, $x_i \neq x_j$ for all $i \neq j$. This can be ensured with high probability by choosing the parameters of the generator appropriately. In particular, for the basic generator (without the random walk component), the table entries have to be reset sufficiently often. For the full generator (including the random walk component), we refer to Theorem 6.

Theorem 8. *Let $x_1, \ldots, x_k$ ($k \geq 1$) be successive outputs of G such that $x_k \neq x_i$ for all $1 \leq i < k$. Let $M_1, \ldots, M_k$ be a sequence of messages of the adversary's choice. Then distinguishing $E(M_k, x_k)$ from an encryption of a random message, given $(M_1, E(M_1, x_1)), \ldots, (M_{k-1}, E(M_{k-1}, x_{k-1})), M_k$ is as hard as inverting $x \mapsto x^e$ (i.e. RSA) on random inputs.*

Proof. Suppose the statement of the theorem is not true, i.e., there exists a pair of probabilistic poly-time algorithms A_1 and A_2 such that A_1 can generate a sequence $M_1, \ldots, M_k$, and A_2, given

$$I = [(M_1, E(M_1, x_1)), \ldots, (M_{k-1}, E(M_{k-1}, x_{k-1})), M_k],$$

can distinguish $E(M_k, x_k)$ from the encryption of a random message. More formally,

$$\Pr_R[A_2(I, \{E(M_k, x_k), E(R, x_k)\}) = E(M_k, x_k) \mid (M_1, \ldots, M_k) \leftarrow A_1] > \epsilon$$

for some nonnegligible ϵ. The probability is taken over R and the coins of A_1, A_2, and G, with the constraint that $x_k \neq x_i$ for all $1 \leq i < k$.

Let's now construct an algorithm B that would invert $z \mapsto z^e$ with nonnegligible probability. Suppose we are given z^e. Start by running A_1 to obtain $(M_1, \ldots, M_k)$ (if A_1 makes any queries to the oracle, return uniformly distributed independent random answers for new queries, or stored answers for repeated queries, and store the queries and answers in a table). Generate random κ-sized subsets S_j, for $1 \leq j \leq k$. Let h be a random element of S_k. Let r be a random number. Set β_h to $r^e z^e$ 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 in $\mathbb{Z}_N^*$, and set $\beta_i = \alpha_i^e$. Let $y_j = \prod_{i \in S_j} \beta_i$ for $1 \leq j \leq k$.

For j from 1 to k, set o_j as follows:

1. If A_1 has asked the random oracle a query x such that $x^e = y_j$, then set o_j to be the answer given to that query.

2. If $1 < j < k$ and $y_j = y_i$ for some $1 \leq i < j$, then set $o_j = o_i$.

3. Otherwise, set o_j to be a uniformly distributed independent random number in $\mathbb{Z}_N^*$.

Compute

$$I = [(M_1, (y_1, o_1 \oplus M_1)), \ldots, (M_{k-1}, (y_{k-1}, o_{k-1} \oplus M_{k-1})), M_k].$$

Now run $A_2(I, \{(y_k, o_k \oplus M_k), (y_k, o_k \oplus R)\}$ for a random R.

If A_2 queries the random oracle for a value of $f(x)$, $x \in \mathbb{Z}_N^*$, the following procedure is followed:

1. If $x^e = y_j$ for some $j \in [1, k]$, then return o_j.
2. If x has been queried before, return the original answer to that query.
3. Otherwise, generate and return a new uniformly distributed independent random answer.

It is easy to see that the distribution of the inputs given to A_2 is the same as the original distribution of I, $x_1, \ldots, x_k$, and the outputs of f. Therefore, by assumption, A_2 will distinguish between $o_k \oplus M_k$ and $o_k \oplus R$ with a nonnegligible probability. However, o_k is uniformly distributed random and independent of I, y_k, and R, and the only way it could be obtained by A_2 is by sending a query x to the random oracle such that $x^e = y_k$. Therefore, A_2 must send such a query x to the oracle with nonnegligible probability. We can monitor the queries that A_2 makes to get the value of x (we can efficiently check whether a query is equal to x using the equality $x^e = y_k$).

We have

$$\left(\frac{x}{r \prod_{i \in S_k - \{h\}} \alpha_i}\right)^e = \frac{y_k}{r^e \prod_{i \in S_k - \{h\}} \beta_i} = \frac{z^e \prod_{i \in S_k} \beta_i}{\prod_{i \in S_k} \beta_i} = z^e.$$

Thus, given x, we can compute z as

$$\frac{x}{r \prod_{i \in S_k - \{h\}} \alpha_i}.$$

It is easy to see that the algorithm B shown above computes z from z^e with nonnegligible probability. □

3.2 $x^e, x \oplus M$

In practice, we cannot assume that a random oracle is available. However, even in the simple case $f(x) = x$, our scheme can be shown to have a certain level of security.

Theorem 9. *Let $x_1, \ldots, x_k$ ($k \geq 1$) be successive outputs of G. Then computing M_k, given $E(M_1), \ldots, E(M_{k-1}), E(M_k)$ for uniformly distributed independent random M_1, $\ldots$, M_k, is as hard as inverting $x \mapsto x^e$ (i.e. RSA) on random inputs.*

Proof. Suppose the statement of the theorem is not true, i.e., there exists a probabilistic poly-time algorithm A which, given

$$I = [E(M_1, x_1), \ldots, E(M_k, x_k)],$$

can compute M_k. More formally,

$$\Pr[A(I) = M_k \mid M_1, \ldots, M_k \stackrel{R}{\leftarrow} \mathbb{Z}_N^*] > \epsilon$$

for some nonnegligible ϵ. The probability is taken over the choice of $M_1, \ldots, M_k$ and over the coins of A and G.

Let's now construct an algorithm B that would invert $z \mapsto z^e$ with nonnegligible probability. Suppose we are given z^e. Generate random κ-sized subsets S_j, for $1 \leq j \leq k$. Let h be a random element of S_k. Let r be a random number. Set β_h to $r^e z^e$ and set α_h to undefined. Now, for $i \in [1, n] \setminus \{h\}$, set α_i to be uniformly distributed independent random numbers in $\mathbb{Z}_N^*$, and set $\beta_i = \alpha_i^e$. Let $y_j = \prod_{i \in S_j} \beta_i$ for $1 \leq j \leq k$.

Generate uniformly distributed independent random $o_1, \ldots, o_k$. Compute

$$I = [(y_1, o_1), \ldots, (y_k, o_k)].$$

Run $A(I)$. It is easy to see that the distribution of the input given to A is the same as the original distribution of I. Therefore, A will with nonnegligible probability return M_k such that (y_k, o_k) is an encryption of M_k. Then $x = M_k \oplus o_k$ has the property

$x^e = y_k$. We now have

$$\left(\frac{x}{r\prod_{i\in S_k-\{h\}}\alpha_i}\right)^e = \frac{y_k}{r^e\prod_{i\in S_k-\{h\}}\beta_i} = \frac{z^e\prod_{i\in S_k}\beta_i}{\prod_{i\in S_k}\beta_i} = z^e.$$

Thus we can compute z as

$$\frac{x}{r\prod_{i\in S_k-\{h\}}\alpha_i}.$$

It is easy to see that the algorithm B shown above computes z from z^e with nonnegligible probability. □

3.3 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 [28]. 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.

Chapter 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 Chapter 2 can be translated into the discrete logarithm framework. Results of sections 2.2 and 2.4 (in particular, Theorem 6) can be easily adapted to the discrete log versions of the generators.

Let p be a prime of length m, and let $g \in Z_p^*$ with order $\mathrm{ord}(g)$ (we note that the exact order of g does not need to be known, and it is sufficient for $\mathrm{ord}(g)$ to be a multiple of the order). The task is to generate a random k and compute $g^k \bmod p$— as required by many protocols. In the remainder of this chapter, 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 $k = \sum_{i \in S} \alpha_i \bmod \text{ord}(g)$. If $k = 0$, stop and start again. Let $K = \prod_{i \in S} \beta_i$ and return (k, K) as the result of G'.

Generation algorithm G'_{exp}: Let $n_e = c \log \text{ord}(g)$ for some constant c (e.g. $c = 0.5$ or $c = 1$).

- **Preprocessing Step:** Generate n random integers $\alpha_i \in \mathbb{Z}_{\text{ord}(g)}$. Compute $\beta_i = g^{\alpha_i}$ for each i and store the α_i's and β_i's in a table.

  Generate a random subset $S_e \subset \mathbb{Z}_{\text{ord}(g)}$ of size n_e. For each $d_i \in S_e$, $1 \leq i \leq n_e$, set $D_i = g^{d_i}$ and store (d_i, D_i) in a table. Set r to a random element of $\mathbb{Z}_{\text{ord}(g)}$ and R to g^r.

- **Whenever a pair (x, g^x) is needed:** Randomly generate $S \subset [1, n]$ such that $|S| = \kappa$. Select a random $j \in [1, n_e]$. Set $r := (r + d_j) \bmod \text{ord}(g)$ and $R := R \cdot D_j$. Let $k = (r + \sum_{i \in S} \alpha_i) \bmod \text{ord}(g)$. Let $K = R \cdot \prod_{i \in S} \beta_i$ and return (k, K) as the result of G'_{exp}.

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. Below we denote by μ the distribution of the outputs of G'. Note that all the results shown here for G' can be easily extended to G'_{exp}.

Theorem 10. *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 $O(1/\varepsilon)$.*

Proof. 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. □

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 $(y^{\alpha_i})_{i \leq n}$ 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.

4.3.1 ElGamal signatures

ElGamal signatures [13] are defined as follows. x is the secret key, $y = g^x$ is the public key. $\sigma(M, k) = (r, s)$, where $r = g^k \bmod p$, $s = (M - xr)k^{-1} \bmod (p-1)$. To verify the signature, one checks whether $r^s = g^M y^{-r}$.

For ElGamal signatures the condition $k = 0$ in G' is replaced by $\gcd(k, p-1) \neq 1$. Let us show that ElGamal signatures with k and r generated by G' are as secure against known-message attacks as if k were uniformly distributed independent random numbers.

Lemma 11. *(Security against known message attacks) Assume ElGamal Signatures*

with intermediate random numbers k generated by G' is insecure against known message attacks. Then one can construct B^A such that for all $\bar{M}$, M, k, and some $\bar{k}$, $B^A(y, \bar{M}, (M, \sigma(M,k))) = \sigma(\bar{M}, \bar{k})$.

That is, given access to A and a *single* message-signature pair, B^A can forge the signature of a new message. This would be a serious known message attack of size one on ElGamal signatures. To use A, the algorithm B^A has to fake the generation of I above with k_i distributed identically as the outputs of G'. The next claim which is a known attack on ElGamal signatures, addresses this. The messages whose signatures are being faked this way, would be distributed almost randomly.

Claim 12. *Given y, $(M, \sigma(M,k)) = (r,s)$ for any M and k such that $\gcd(r, p-1) = 1$, and any $c \in \mathbb{Z}^*_{p-1}$, it is possible to compute $\sigma(M', ck)$ for some $M' \neq M$ without knowing x or k.*

Proof. Let $\sigma(M,k) = (r,s)$. All the inverses below are modulo $p-1$. We have $\sigma(M', ck) = (r', s')$ for

$$\begin{aligned} r' &= r^c \bmod p, \\ s' &= c^{-1} r' r^{-1} s \bmod (p-1), \\ M' &= r' r^{-1} M \bmod (p-1), \end{aligned}$$

Then, $r'^{s'} = g^{kcc^{-1}r'r^{-1}(M-xr)k^{-1}} = g^{M'-xr'} \bmod p$. □

Proof of the Lemma. Let us construct B^A to work as follows: Let $(r,s) = \sigma(M,k)$. If $\gcd(r, p-1) \neq 1$ (which happens with constant probability less than 1), B^A fails. Otherwise, it would generate random α_j and k_i as in G' (without computing β_j or K_j). B^A can then compute $\sigma(M'_i, k_i k)$, $1 \leq i \leq \ell$, using the above method. $A(y, \bar{M}, \{\sigma(M'_i, k_i k)\}_{i=1}^{\ell}) = \sigma(\bar{M}, \bar{k})$ is returned.

The sequence given by B^A to A looks as though it was produced by G' with the table $\alpha'_j = \alpha_j k$. Since α_j are random, α'_j are random. Therefore, we have:

Claim: the input to A from B^A has the distribution σ_μ (i.e., the distribution on outputs of σ with k distributed by μ).

It follows that B^A will have the same success rate as A. □

4.3.2 DSS signatures

DSS signatures are defined as follows. q is a large prime divisor of $p-1$, and g has order q in $\mathbb{Z}_p^*$. x is the private key and $y = g^x$ is the secret key. $\sigma(M,k) = (r,s)$, where $r = (g^k \bmod p) \bmod q$ and $s = (M + xr)k^{-1} \bmod q$. Verifying a signature is done by checking if $((g^M y^r)^{s^{-1}} \bmod p) \bmod q = r$.

Lemma 13. *If DSS signatures with k generated by G' are insecure against known-message attacks, then an arbitrary DSS signature can be forged using only the public key.*

Let us first prove the following:

Claim 14. *For any c and d in $\mathbb{Z}_q^*$ such that $y^c g^d \neq 1$, it is possible to compute $\sigma(M', (cx+d) \bmod q)$ for some M' knowing x*

Proof. Let

$$r' = (y^c g^d \bmod p) \bmod q,$$
$$s' = r'c^{-1} \bmod q,$$
$$M' = r'c^{-1}d \bmod q.$$

It is easy to see that $(r', s') = \sigma(M', (cx+d) \bmod q)$. □

Proof of the lemma. Suppose there exists A such that $A(y, M, I) = \sigma(M, k)$ for some k, where $I = \{(M_i, \sigma(M_i, k_i))\}_{i=1}^{\ell}$, with k_i and g^{k_i} generated by G'. Let us construct B^A such that $B^A(y, M) = \sigma(M, k)$ for some k.

B^A would work as follows. It would fix some c_0 and d_0 in $\mathbb{Z}_q^*$ such that $y^{c_0} g^{d_0} \neq 1$. Then it would generate random α_j and k_i as in G' (without computing β_j or K_i). B^A can then compute $I = \{\sigma(M_i', (c_0 k_i x + d_0 k_i) \bmod q)\}_{i=1}^{\ell}$ using the above method. $A(y, M, I) = \sigma(M, k)$ is returned.

It is easy to see that the sequence given by B^A to A looks as though it was produced by G' with the table $\alpha'_j = \alpha_j(c_0 x + d_0)$. Since α_j are random, α'_j are random, and so we have:

Claim: The input of A in B^A has distribution σ_μ.

It follows that B^A will have the same success rate as A. □

4.3.3 Schnorr signatures

Schnorr signatures [26] are defined as follows.[1] q is a large prime divisor of $p-1$, and g has order q in $\mathbb{Z}_p^*$. s is the private key and $v = g^{-s}$ is the public key. $\sigma(M,k) = (r,y)$, where $r = h(g^k, M) \in [0, 2^t - 1]$, h is a hash function (not necessarily one-way), and $y = (k+sr) \bmod q$. Verification is done by computing $\bar{x} = g^y v^r$ and checking whether $r = h(\bar{x}, M)$.

We will prove that using G' with Schnorr signatures is secure for $t = |q|$. We will consider h defined by $h(x, M) = (x + M) \bmod q$. This h satisfies the requirements given in [26], as it is uniform in x, depends on at least $|q|$ bits of x, and, if M is the output of a one-way hash function on the signed message, one-way with respect to the original message (in any case, the last requirement is not needed if one is not concerned about chosen-message attacks).

Lemma 15. *For t and h specified above, Schnorr signatures with G' used to generate k are as secure against known-message attacks as Schnorr signatures with independent k.*

Proof. Suppose there exists A such that for any M and I, and for some k, $A(v, M, I) = \sigma(M,k)$, where $I = \{(M_i, \sigma(M_i, k_i))\}_{i=1}^{\ell}$, with k_i and g^{k_i} generated by G'. Let us construct B^A such that for any M and for some k, $B^A(v, M) = \sigma(M,k)$. Thus, B^A could forge any signature after seeing just the public key.

[1] The definition given here is similar to DSS and ElGamal signatures in that it does not perform one-way hashing on M. Just as in DSS and ElGamal, chosen message attacks become possible, but they do not present any real threat since the messages signed would be random. In practice, M would be the one-way hash of the actual message. Note that in [26] one-way hashing of the message is part of the definition.

B^A would work as follows. Fix some r_0 and y_0. Generate random α_j and k_i as in G'. Then compute

$$r_i = k_i r_0 \bmod q,$$
$$y_i = k_i y_0 \bmod q,$$
$$x_i = g^{y_i} v^{r_i},$$
$$M_i = (r_i - x_i) \bmod q.$$

The result of $A(v, M, \{(M_i, (r_i, y_i))\}_{i=1}^{\ell})$ is returned.

We have $x_i = g^{k_i(y_0 - sr_0)}$. It is easy to see that the sequence given by B^A to A looks as though it was produced by G' with the table $\alpha'_j = \alpha_j(y_0 - sr_0)$. Since α_j are random, α'_j are random, and so we have

Claim: The input of A in B^A has distribution σ_μ.

It follows that B^A will have the same success rate as A. □

4.4 Diffie-Hellman Key Exchange

Diffie-Hellman key exchange is defined as follows. Alice generates a random $a \in \mathbb{Z}_{\text{ord}(g)}$ and sends g^a to Bob. Bob generates a random $b \in \mathbb{Z}_{\text{ord}(g)}$ 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.

Lemma 16. *Diffie-Hellman key exchange with G' used to generate (a, g^a) is as secure as Diffie-Hellman key exchange with independent a's.*

Proof. Suppose that Diffie-Hellman key exchange using G' is not secure, i.e., an adversary can guess the secret key of a particular session after seeing ℓ key exchanges. In other words, there exists an algorithm A such that for any I and any a, $A(I, g^a) = g^{ak_\ell}$, where $I = \{g^{k_i}\}_{i=1}^{\ell}$ is generated by G' (messages g^{b_i} sent by Bob are not included in I since they are not correlated with g^{k_i}). Let us construct an algorithm B^A such that for any a and b, $B^A(g^a, g^b) = g^{ab}$, which would contradict the security of the standard Diffie-Hellman scheme.

B^A would work as follows. Generate α_i randomly and independently and let $\beta_i = g^{\alpha_i}$. Generate random S_i, for $1 \leq i \leq \ell$. Let h be a random element of S_ℓ. Let r be a random number independent of α_i's. Set β_h to $g^r g^b$ and set α_h to undefined. Let $K_i = \prod_{j \in S_i} \beta_j$. Let $z = A(\{K_i\}_{i=1}^{\ell}, g^a)$. Now compute $X = z/(g^a)^{r + \sum_{i \in S_\ell - \{h\}} \alpha_i}$. Return X.

Claim: The sequence $\{K_i\}$ generated above has the distribution μ.

Therefore, A would have a non-negligible success rate on such input. Suppose that A is successful. By assumption on A we have

$$z = g^{ak_\ell} = (K_\ell)^a = \prod_{i \in S_\ell} (\beta_i)^a = (\beta_h)^a \prod_{i \in S - \{h\}} (\beta_i)^a.$$

Therefore,

$$X = \frac{z}{(g^a)^{r + \sum_{i \in S_\ell - \{h\}} \alpha_i}} = (\beta_h)^a \cdot \frac{\prod_{i \in S - \{h\}} (\beta_i)^a}{g^{ar} \prod_{i \in S_\ell - \{h\}} g^{a\alpha_i}} = g^{ar} g^{ab} \cdot \frac{\prod_{i \in S - \{h\}} (\beta_i)^a}{g^{ar} \prod_{i \in S - \{h\}} (\beta_i)^a} = g^{ab}.$$

□

4.5 ElGamal Encryption

ElGamal encryption [13] 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 17. *ElGamal encryption with G' used to generate (k, g^k) is secure against ciphertext-only attacks if standard Diffie-Hellman is secure.*

Proof. Let I denote a sequence $e(M_i, k_i)$, $1 \leq i \leq \ell$, with k_i and g^{k_i} computed by G'. Suppose there exists A such that for any I, $A(y, I) = M_\ell$. Let us construct A'^A such that for any $I' = \{g^{k_i}\}_{i=1}^{\ell}$ generated by G', and for any a, $A'(I', g^a) = g^{ak_\ell}$. It would then follow from lemma 16 that there exists $B^{A'}$ such that for any a and b,

$B^{A'}(g^a, g^b) = g^{ab}$, which would contradict the security of the standard Diffie-Hellman scheme.

A'^A would set y to g^a and generate uniform and independent random numbers r_i. Let $M_\ell = A(y, \{(g^{k_i}, r_i)\}_{i=1}^{\ell})$. A' returns r_ℓ / M_ℓ as the result.

Since $e(M_\ell, k_\ell) = (g^{k_\ell}, r_\ell)$, $M_\ell = r_\ell / y^{k_\ell}$. Therefore, $r_\ell / M_\ell = y^{k_\ell} = g^{ak_\ell}$. □

Chapter 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 (i.e. for schemes or modes of attack for which security was not proved to be preserved), 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, or any other scheme for which a security proof has been given above), 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 5.1 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 5.1 displays the resource requirements for these parameter choices as well as those for the algorithms of [29,

Table 5.1: A comparison of methods of generating pairs $(x, g^x \bmod p)$ and $(x, x^e \bmod N)$ for $|p| = 512$, $|\mathrm{ord}(g)| = 512$, $|N| = 512$, $|e| = 512$. 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. [29]	512	100	not applicable	
Brickell et al. [29]	10880	64	not applicable	
Lim and Lee [17]	317	100	not applicable	
Lim and Lee [17]	13305	52	not applicable	
de Rooij [11]	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

11, 17] and square-and-multiply. For the algorithms of [29, 17], 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 $G_{\mathrm{exp}}, G'_{\mathrm{exp}}$.

Bibliography

[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] Noga Alon and Yuval Roichman. Random Cayley graphs and expanders, 1996.

[3] D. Boneh and R. Venkatesan. Breaking RSA may not be equivalent to factoring. In *EUROCRYPT '98*, 1998. To appear.

[4] Victor Boyko, Ramarathnam Venkatesan, and Marcus Peinado. Speeding up discrete log and factoring based schemes via precomputations. In *EUROCRYPT '98*, 1998. To appear.

[5] E. Brickell. Solving low density knapsacks. In *Proceedings of CRYPTO '83*, pages 25–37, New York, 1984. Plenum Press.

[6] R. Canetti. Towards realizing random oracles: Hash functions that hide all partial information. In B. Kaliski, editor, *Advances in Cryptology: CRYPTO '97*, number 1294 in 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. On Schnorr's preprocessing for digital signature schemes. In T. Helleseth, editor, *Advances in Cryptology: EUROCRYPT '93*, volume 765 of *Lecture Notes in Computer Science*, pages 435–439, Berlin, 1993. Springer-Verlag.

[11] P. J. N. de Rooij. Efficient exponentiation using precomputation and vector addition chains. In Alfredo De Santis, editor, *Advances in Cryptology: EUROCRYPT '94*, volume 950 of *Lecture Notes in Computer Science*, pages 389–399, Berlin, 1995. Springer-Verlag.

[12] W. Diffie and M. E. Hellman. New directions in cryptography. *IEEE Trans. Inform. Theory*, IT-22:644–654, November 1976.

[13] T. ElGamal. A public key cryptosystem and a signature scheme based on discrete logarithms. *IEEE Trans. Inform. Theory*, 31:469–472, 1985.

[14] National Institute for Standards and Technology. Digital Signature Standard (DSS). *Federal Register*, 56(169), August 30 1991.

[15] J. Lagarias and A. Odlyzko. Solving low density subset sum problems. *Journal of the ACM*, 32:1985, 1985.

[16] A. Lenstra, H. Lenstra, and L. Lovász. Factoring polynomials with rational coefficients. *Mathematische Annalen*, 261:513–548, 1982.

[17] C. H. Lim and P. J. Lee. More flexible exponentiation with precomputation. In Y. Desmedt, editor, *Advances in Cryptology: CRYPTO '94*, number 839 in Lecture Notes in Computer Science, 1994.

[18] Michael Luby. *Pseudorandomness and Cryptographic Applications.* Princeton University Press, Princeton, NJ, 1996.

[19] Rajeev Motwani and Prabhakar Raghavan. *Randomized Algorithms.* Cambridge University Press, 1995.

[20] V.I. Nechaev. Complexity of determinate algorithm for the discrete logarithm. *Mathematical Notes*, 55(2):165–172, 1994.

[21] D. Pointcheval and J. Stern. Security proofs for signature schemes. In Ueli Maurer, editor, *Advances in Cryptology: EUROCRYPT '96*, number 1070 in Lecture Notes in Computer Science, 1996.

[22] R. L. Rivest, A. Shamir, and L. Adleman. A method for obtaining digital signatures and public-key cryptosystems. *CACM*, 21(2):120–126, 1978.

[23] C. Schnorr and M. Euchner. Lattice basis reduction: Improved practical algorithms for solving subset sum problems. *Mathematical Programming*, 66:181–199, 1994.

[24] C. 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*, number 921 in Lecture Notes in Computer Science, 1995.

[25] 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.

[26] C. P. Schnorr. Efficient signature generation by smart cards. *J. Cryptology*, 4:161–174, 1991.

[27] V. Shoup. Lower bounds on discrete logarithms and related problems. In W. Fumy, editor, *Advances in Cryptology: EUROCRYPT '97*, volume 1233 of *Lecture Notes in Computer Science*, 1997.

[28] M.J. Wiener. Cryptanalysis of short RSA secret exponents. *IEEE Transactions on Information Theory*, 36(3):553–558, 1990.

[29] David B. Wilson, Kevin S. McCurley, Daniel M. Gordon, and Ernest F. Brickell. 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, 1992. Springer-Verlag.