Design and analysis of provably secure pseudorandom generators

Design and analysis of provably secure pseudorandom generators

PROEFSCHRIFT

ter verkrijging van de graad van doctor aan de
Technische Universiteit Eindhoven, op gezag van de
Rector Magnificus, prof.dr.ir. C.J. van Duijn, voor een
commissie aangewezen door het College voor
Promoties in het openbaar te verdedigen
op maandag 29 oktober 2007 om 16.00 uur

door

Andrey Sidorenko

geboren te Moskou, Rusland

Dit proefschrift is goedgekeurd door de promotor:

prof.dr.ir. H.C.A. van Tilborg

Copromotor:
dr.ir. L.A.M. Schoenmakers

CIP-DATA LIBRARY TECHNISCHE UNIVERSITEIT EINDHOVEN

Sidorenko, Andrey

Design and analysis of provably secure pseudorandom generators /
door Andrey Sidorenko. –
Eindhoven : Technische Universiteit Eindhoven, 2007.
Proefschrift. – ISBN 978-90-386-1130-3
NUR 919
Subject headings : cryptology
2000 Mathematics Subject Classification : 11T71, 65C10, 68Q17

Printed by Printservice Technische Universiteit Eindhoven
Cover by Jorrit van Rijt, Oranje Vormgevers

Моей дорогой бабушке
To my dear grandmother

Contents

Chapter 1

Introduction

Random numbers should not be generated with a method chosen at random. Some theory should be used.

Donald Ervin Knuth

A cryptographically secure pseudorandom generator is a mechanism for producing random bits that can be used for cryptographic purposes. Since randomness is essential for most cryptographic systems, developing robust sources of random data is a very important problem.

Quite often it is just assumed in cryptographic literature that a source of random data is available, paying no attention to the fact that actually randomness is quite difficult to generate. In this thesis, we provide new ideas about how to generate random data. We analyze several existing pseudorandom generators and we also propose some new constructions.

Pseudorandom generators are an alternative for hardware random number generators. Although hardware random number generators are sometimes useful, they have certain drawbacks and they fail to fulfill requirements of some applications.

We like to emphasize that we focus on provably secure constructions. Provable security means that the bits produced by the generator are indistinguishable from independent uniformly random bits unless a certain intractable problem can be solved. We analyze pseudorandom generators in the setting of concrete security. For each construction, we determine concrete parameters (e.g., the seed length) such that a desired level of security is reached under the corresponding intractability assumption.

The thesis starts with an introduction into the field of randomness. Applications of randomness as well as different approaches for generating random data are discussed.

1.1 Applications of randomness

One of the most successful lotteries ever organized in the world is “Sportloto”, which was founded by the government of the Soviet Union in 1970 to support the sports organizations. The rules were quite simple. First, a participant chose 6 different numbers between 1 and 49. Then, at the day of the lottery, 6 balls out of 49 were drawn from a well-stirred urn and the participant could win a lot of money if he had guessed correctly the numbers written on the balls. One can compute that the chance to win in this lottery was quite small, i.e., 1/13983816. However, the participation price was reasonably low so every time about 30 million people (more than 10% of the population of the country) pushed their luck. The week budget of “Sportloto” was about 9 million rubles. The lottery was stopped only when the Soviet Union collapsed. Similar lotteries still exist in quite a few other countries.

The key issue of any lottery is *randomness*. To choose a winner among many participants in a fair way, a set of random numbers has to be generated.

Of course, lotteries are not the only consumers of random data. There are many other (perhaps, more important) applications of randomness. Random numbers are used by a wide spectrum of algorithms. For instance, a well-known technique to approximate π is to pick many random points in a one-by-one square and to measure how many points are at a distance at most 1 from a particular apex. Random numbers are also used to simulate natural phenomena, say, in the area of hydrodynamics or nuclear physics. Another application of random numbers is sampling. Imagine, for instance, an election campaign. To estimate a rank of a candidate, it is possible to interview relatively few random voters.

In this thesis, we focus on cryptographic applications of random numbers. The central idea of any cryptographic scheme is that there is a secret which is known only to the authorized users. The best way to make the secret unpredictable for the adversaries is to choose it uniformly at random (in this case the entropy of the secret is maximal).

The main cryptographic applications of randomness are the following:

- Session keys for symmetric ciphers.
- Private keys for asymmetric cryptosystems and digital signature schemes.
- Random data for probabilistic encryption and signing.
- Initialization vectors for block ciphers used in chaining modes.
- Random challenges and nonces for cryptographic protocols.
- Salts to be added to passwords and passphrases.

In some cases random data is transmitted and stored in clear (e.g., initialization vectors and challenges) while in the other cases it is kept secret (e.g., session keys). In both situations, security of the system relies on the availability of random numbers with certain properties. Since any system is as secure as its weakest link, it is important to have a robust source of random numbers.

Unless specified otherwise, by a random number we mean an integer uniformly distributed in a certain interval.

1.2 Hardware random number generators

The problem is that random numbers are often difficult to generate. A possible solution is to use a *hardware random number generator* based on a physical phenomenon, such as thermal noise or radioactive decay. For instance, the Intel 80802 Firmware Hub chip included a hardware random number generator using two free running oscillators, one fast and one slow. A thermal noise source from two diodes was used to modulate the frequency of the slow oscillator, which then triggered a measurement of the fast oscillator. The output rate of this device was somewhat less than 100,000 bits per second. The device is not included in modern PCs.

Figure 1.1: One of the popular hardware random number generators is a *lava lamp* sealed within a transparent liquid-filled cylinder. The illuminated colored globs inside the lamp slowly rise and fall in such a way that their shapes and paths change unpredictably. Therefore, a digital camera filming the lamp can produce a sequence of random numbers.

Although in many cases hardware random number generators are useful, they have several disadvantages. First of all, they are not always available. Some of them are quite slow, others are too expensive, many of them have malfunctions that are extremely difficult to detect. Marsaglia [Mar95] points out that many seemingly robust hardware random number generators in fact produce highly correlated data. Sometimes the malfunctions can be initiated by the adversary. Finally, it is impossible to reproduce the sequences of numbers output by hardware random number generators, which is quite unfortunate for many applications.

1.3 Pseudorandom generators

Is there any way to generate random numbers on deterministic computers?

Say, is it possible to use the binary expansion of π as a sequence of random bits? By now, the bits of π have been calculated to billions of places and, in fact, an arbitrary bit of π can be determined without needing to compute any of the preceding bits. The bits of π are proved to "behave randomly" in a certain statistical sense under a plausible conjecture in the field of chaotic dynamics (for more details see, e.g., [Pre01]). Unfortunately, the bits of π cannot be used for cryptographic applications simply because they are publicly known. For example, if you are the organizer of "Sportloto" or any other similar lottery you do not want someone to compute the winning numbers in advance. The moral is that even if a sequence of random numbers is produced deterministically it has to be a function of an unpredictable random seed.

1.3.1 Computational indistinguishability

Loosely speaking, a *pseudorandom generator* is a deterministic algorithm that converts a relatively short random seed into a longer sequence of numbers that "behaves randomly". The output sequence is called the pseudorandom sequence.

A formal definition of pseudorandom generator is discussed in Chapter 2. At this point, it is important to specify what we actually mean by saying that a sequence of numbers "behaves randomly". A lot of *statistical tests* have been designed to check if a sequence of numbers is sufficiently random, that is, it does not deviate much from a sequence of independent uniformly distributed random numbers. For instance, given a decimal sequence one can check if all single digits occur about one tenth of the time, all two-digit combinations occur about one one-hundredth of the time, and so on. In fact, this simple test is a variant of the first Golumb's randomness postulate (the Golumb's randomness postulates are described, for instance, in [MvOV00]). Other tests include monobit, frequency, serial, poker, runs/long-runs, and autocorrelation tests. Some of these tests are included in FIPS 140-1 standard. Even more of them are a part of the well-known Diehard battery developed by Marsaglia [Mar95].

For cryptographic applications, it is important that a sequence produced by a pseudorandom generator passes all efficient statistical tests, even those that have not been discovered yet. At first glance, it sounds like utopia. How can we guarantee that such a strong requirement is satisfied? It turns out, that sometimes we can. Actually, there exist pseudorandom generators such that no *efficient distinguisher* can tell the difference between the sequences produced by these generators and uniformly random sequences. The latter pseudorandom generators are referred to as *cryptographically secure* pseudorandom generators.

Except for a few situations, all pseudorandom generators considered in this thesis are cryptographically secure. Whenever it is clear from the context that a pseudorandom generator is cryptographically secure, we simply call it a pseudorandom generator.

It can be shown that sequences produced by pseudorandom generators are *unpredictable* in the sense that knowledge of many consecutive elements of the sequence gives no information about the next element. This important result is often attributed to Yao [Yao82] but actually it did not appear in [Yao82]. In fact, Yao presented this result and the proof idea in his talk at the Symposium on Foundations of Computer Science in 1982. A good exposition of the proof is in [Gol01, Section 3.3.5].

In some sources, pseudorandom generators are called pseudorandom number generators [Yao82; KSF99], pseudorandom bit generators [MvOV00; Kal88], or deterministic random bit generators [BK05; Cam06]. The term pseudorandom generator [Gol95; Lub94] is more common for number-theoretic constructions, on which we focus in this thesis.

Note that a pseudorandom generator that outputs bits can be transformed into a pseudorandom generator that outputs numbers in any interval. We discuss this issue in detail in Chapter 7. For the rest of the thesis we assume that pseudorandom generators output bits, without loss of generality.

1.3.2 Kolmogorov complexity

Actually, computational indistinguishability is not the only way to determine the degree of randomness of a sequence. An alternative way is to use the approach proposed by Solomonov and Kolmogorov in the early 1960s (for a recent treatment see, e.g., [Gol95]). Loosely speaking, a sequence of numbers is Kolmogorov-random if its length is the same as the length of the shortest algorithm producing this sequence. In other words, a sequence is Kolmogorov-random if it cannot be compressed. The reason why the approach of Solomonov and Kolmogorov is not suitable in our context is that Kolmogorov complexity is a function that in general cannot be efficiently computed. On the contrary, we consider only computationally bounded distinguishers.

1.3.3 Generating the seed

Generation of an unpredictable random seed is an important issue. The seed is typically obtained from operating system events, for instance, mouse and keyboard activity, disk I/O operations, and specific interrupts. These events are referred to as sources of entropy.

Unfortunately, in many cases the seed is generated improperly, that is, there are either too few sources of entropy or the sources of entropy are observable by the adversary. For instance, the Linux distribution for wireless routers OpenWRT does not provide enough entropy for its pseudorandom generator [GPR06] and thus the seed can be computed by exhaustive search. For a similar reason, the pseudorandom generator used by the Secure Sockets Layer protocol of the early Netscape browser is insecure [GW96]. The latter problem is especially dangerous because the SSL protocol is developed to provide secure Internet transactions.

In this thesis, we do not consider the problem of collecting entropy. We analyse security of pseudorandom generators assuming that the seed is uniformly random and unknown for unauthorized users.

1.3.4 Repeatability

For a fixed seed, a pseudorandom generator always outputs the same pseudorandom sequence. This important property called repeatability gives pseudorandom generators extra applications that hardware random number generators do not have.

For instance, pseudorandom generators are used as keystream generators for synchronous stream ciphers. To encrypt a message with a secret key, the sender feeds a pseudorandom generator with the key and XORs the message with the resulting pseudorandom sequence. The recipient that knows the key can also compute the pseudorandom sequence and reconstruct the message.

Furthermore, pseudorandom generators are used as building blocks for commitment schemes [Nao89] and pseudorandom functions [NR04].

1.4 Provably secure pseudorandom generators

In this thesis, we analyze pseudorandom generators in the setting of *provable security*. A cryptographic scheme is said to be provably secure if breaking the scheme can be shown to be essentially as difficult as solving a well-known and supposedly difficult (typically, number-theoretic) problem. In the case of pseudorandom generators, to break an algorithm means to distinguish its output from uniformly random with some non-negligible advantage. Thus, provable security means security under a certain computational assumption.

The first cryptographically secure pseudorandom generator is proposed by Blum and Micali [BM82]. The Blum-Micali generator outputs one bit per iteration, which costs one exponentiation modulo a large prime p. Security of the Blum-Micali generator is based on intractability of the discrete logarithm problem in $\mathbb{Z}_p^*$ (for any integer $N > 1$, we denote by $\mathbb{Z}_N^*$ the multiplicative group of integers modulo N that are relatively prime to N). The Blum-Micali generator is improved by Long and Wigderson [LW83] and Håstad and Näslund [HN99] who show that the generator remains secure if one outputs $O(\log \log p)$ bits per iteration. Kaliski [Kal88] proposes an elliptic curve version of the Blum-Micali generator. Patel and Sundaram [PS98] and Gennaro [Gen05] suggest a further improvement for the Blum-Micali generator. They propose a pseudorandom generator that outputs $O(\log p)$ bits per modular exponentiation. However, security of the new generator is based on a strong and not so well-studied "discrete logarithm with short exponents" assumption.

Another line of pseudorandom generators is based on factoring-like assumptions. The well-known RSA generator proposed by Goldwasser et al. [GMT82] iterates the RSA encryption function $x \mapsto x^e \bmod N$, where N is an RSA modulus, e is a public exponent, and outputs one bit per iteration. Security of the RSA generator is based on the RSA problem. Related to the RSA generator is the Blum-Blum-Shub generator that iterates the Rabin function $x \mapsto x^2 \bmod N$, where N is a Blum integer, and outputs one bit per iteration [BBS86]. Security of the Blum-Blum-Shub generator is based on the hardness of factoring. The RSA generator as well as the Blum-Blum-Shub generator are shown to be secure even if not only one but $O(\log\log N)$ bits are output per iteration [BOCS83; VV84; ACGS88; HN99; SS05]. The recent result by Steinfeld et al. [SPW06] implies that $O(\log N)$ bits can be output per iteration if one assumes that a non-standard variant of the RSA problem (which is potentially easier to solve than the original RSA problem) is intractable.

Many other intractable number-theoretic problems have been used to design pseudorandom generators. Impagliazzo and Naor [IN89] propose a pseudorandom generator based on the intractability of the subset sum problem. A promising pseudorandom generator based on the hardness of solving a random system of multivariate quadratic equations over a finite field is presented by Berbarin et al. [BGP06] (however, as pointed out by Yang et al. [YCBC07], this pseudorandom generator is secure only if the finite field is small). Boneh et al. [BHHG01] introduce the modular inversion hidden number problem and present a construction based on the intractability of this problem.

The seminal paper by Håstad et al. [HILL99] implies that a provably secure pseudorandom generator can be constructed from any one-way function. Whether or not one-way functions exist is an open conjecture. Nevertheless, certain functions (e.g., exponentiation in a finite field and multiplication of two prime numbers) are assumed to be one-way.

The above overview suggests that stronger assumptions give rise to more efficient pseudorandom generators. On the other hand, strong assumptions are sometimes not thoroughly analyzed so the constructions based on these assumptions cannot be fully trusted.

An alternative to number-theoretic pseudorandom generators are the generators that use block ciphers and cryptographic hash functions [KSF99; FS03; DSS00; BK05]. For instance, Yarrow-160 generator [KSF99] converts a seed $K \in \{0,1\}^{160}$ into the sequence $E_K(c), E_K(c+1), E_K(c+2), \ldots$, where c is a counter and E_K denotes Triple-DES block cipher keyed by K. Yarrow-160 generator and similar constructions are secure in the *ideal-cipher model* meaning that the security of these generators relies on the assumption that the corresponding block cipher is a pseudorandom permutation.

In this thesis, we focus on pseudorandom generators provably secure in the *standard model*. In particular, we focus on constructions based on the intractability of integer factorization, computing discrete logarithms, and related problems (e.g., the RSA problem and the decisional Diffie-Hellman problem). We do not consider pseudorandom generators that use block ciphers and cryptographic hash functions.

The algorithmic transformation of the adversary that breaks a scheme into a solver for the corresponding intractable problem is called *reduction*. The notion of reduction plays an important role not only in cryptology but also in computability theory and complexity theory.

1.5 Concrete security

In the early papers about pseudorandom generators (e.g., in [BM82; GMT82]) the security is analyzed in the asymptotic sense. The length of the pseudorandom sequence M is assumed to be polynomial in the seed length n. A pseudorandom generator is said to be *asymptotically secure* if, as n increases, no polynomial-time adversary can distinguish the pseudorandom sequences from uniformly random sequences with probability $1/2 + 1/f(n)$, for any polynomial f. To show that a pseudorandom generator is asymptotically secure, one usually comes up with a *polynomial-time reduction*, that is, one has to show that a polynomial-time distinguisher implies a polynomial-time solver for the underlying intractable problem.

As first emphasized by Bellare and Rogaway [BR96], asymptotic security says little about the security of a cryptographic scheme in practice for a particular choice of parameters (such as n) and against adversaries investing a specific amount of computational effort. *Concrete security* analysis of a cryptographic scheme is a way to determine the exact parameters of the scheme such that a desired level of security is reached. Concrete security analysis is in some sense a refined version of asymptotic security analysis. A lot of recent papers include concrete security analysis of digital signature schemes [KW03; Cor02], encryption schemes [BF01; BR06], and pseudorandom generators [FS00; Gen05; SPW06].

1.5.1 Tightness of security reductions

When analyzing concrete security of a cryptographic scheme, a crucial issue is *tightness* of the reduction. If breaking the scheme takes roughly the same computational effort as solving the underlying intractable problem, the reduction is called tight. In the opposite case, if the adversary is transformed into a relatively inefficient (although still polynomial-time) solver, the reduction is called not tight.

A tighter reduction guarantees security of the scheme for smaller parameters, which makes the scheme more efficient. To illustrate this important point, consider the following simple example.

Let $\mathcal{S}$ be a cryptographic scheme (e.g., a pseudorandom generator) characterized by a parameter n (e.g., the length of the seed) which is secure under the assumption that the discrete logarithm in a n-bit field is intractable (see also Figure 1.2). Suppose that for smaller n the scheme is more efficient. Our goal is to choose n as small as possible under the restriction that the scheme is at least as secure as solving the discrete logarithm problem in, say, 1200-bit field. There are two possibilities.

1. The reduction is tight so breaking the scheme is exactly as hard as solving the discrete logarithm problem. In terms of Figure 1.2, it means $T = T'$. Thus,

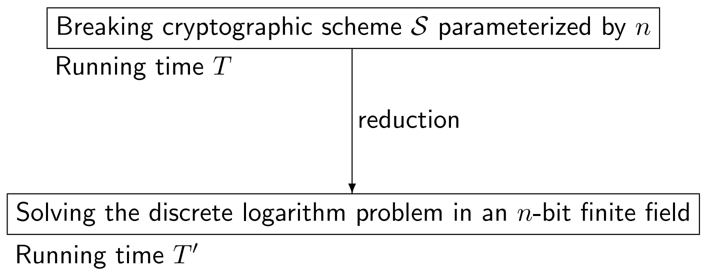
Breaking cryptographic scheme $\mathcal{S}$ parameterized by n
Running time T
reduction
Solving the discrete logarithm problem in an n-bit finite field
Running time T'

Figure 1.2: An adversary that breaks the scheme $\mathcal{S}$ can be transformed into a solver for the discrete logarithm problem.

the answer is $n = 1200$. This is the smallest possible value of n we can expect.

2. The reduction is *not* tight so $T' \gg T$. It means that breaking the scheme can be easier than solving the discrete logarithm problem. Thus, to achieve the same level of security we have to use $n \gg 1200$, which degrades the performance of the scheme. In this case, there are three possible ways to improve the efficiency of the scheme. Either the scheme itself can be modified or a new reduction can be constructed or another intractable problem can be used as a starting point.

One can argue that tightness of the reduction is not important when computing parameters of the scheme that provide a certain level of security [YY04; ECS]. So, in the second case of the above example, even though the reduction is not tight, one can think that the scheme is secure even for $n = 1200$. This argument is supported by the fact that, so far, there exists no natural and realistic provably secure scheme with a non-tight reduction such that this scheme is shown to be insecure when used with commonly accepted parameter sizes (cf. [KM06]). Nevertheless, it does not mean that such schemes do not exist. Moreover, in the second case, $\mathcal{S}$ is *provably secure* only for $n \gg 1200$. For smaller n, the security of the scheme has perhaps nothing to do with the intractability of the discrete logarithm problem.

In this thesis, we rely only on provable results. When computing parameters of a scheme that provide a certain level of security, we do take into account tightness of the reduction.

Early papers about pseudorandom generators, e.g., [GMT82; BOCS83; BM82; ACGS88; Kal88; VV84] present polynomial-time reductions. These results imply that the corresponding pseudorandom generators are asymptotically secure. However, a close look at the reductions shows that they are not tight and they ensure security of the pseudorandom generators only for unreasonably large lengths of the seed.

Design of provably secure schemes (in particular, pseudorandom generators) with tight reductions is an important problem in modern cryptography. This is one of the problems addressed in this thesis.

1.5.2 Tight reductions or weak assumptions?

It is not always clear how to compare reductions corresponding to different intractability assumptions. Consider a cryptographic scheme characterized by a parameter n, as in the previous section. Suppose, the scheme is secure under the assumption that the discrete logarithm problem is intractable but the reduction is *not tight*. Now, suppose that one can construct a *tight* reduction that connects the security of the scheme to the decisional Diffie-Hellman problem. The question is, which of the two reductions is better in the sense that it ensures the security of the scheme for a shorter n? Although the decisional Diffie-Hellman problem can be easier than the discrete logarithm problem, the exact "gap" between the two problems is not known (we discuss this issue in more detail in Section 2.4.2). Therefore, the above question is difficult to answer. We resolve this uncertainty by assuming that the decisional Diffie-Hellman problem and the discrete logarithm problem are equally hard in certain groups, in agreement with common practice. Similarly, the RSA problem is assumed to be as hard as integer factorization (see Section 2.3.2).

In general, to claim that a scheme is secure, it is not sufficient to construct a security reduction. It is also important to check that the underlying computational assumption is plausible.

Chapter 2

Preliminaries

What's in a name? A rose by any other name would smell as sweet.

William Shakespeare

In this chapter, we recall the formal definition of pseudorandom generator and discuss the state of the art in solving intractable problems, e.g., integer factorization and the discrete logarithm problem. We also set up conventions and introduce some notation used throughout the thesis.

2.1 Time units

A *unit of time* has to be set to measure the running time of the algorithms. Throughout this thesis, the unit of time is one CPU cycle of the 450 MHz Pentium II processor. We adopt the unit of time used by Lenstra and Verheul [LV01].

For instance, it is reported that one multiplication modulo a prime p takes about $(\log_2 p)^2/24$ time units [LV01].

In most of the examples throughout this thesis we require cryptographic schemes (e.g., pseudorandom generators) to be secure against all adversaries that run in time at most 2^{88} time units. The reason is that, according to the estimates given in [Len04], the security level of about 2^{88} time units, which corresponds to 2^{80} DES encryptions, will be sufficient for all commercial applications until 2018.

2.2 Formal definition of pseudorandom generator

Let X and Y be random variables taking on values in a finite set S. The *statistical distance* between X and Y is defined as

$$\Delta(X,Y) = \frac{1}{2}\sum_{s\in S} |\Pr[X = s] - \Pr[Y = s]|.$$

We say that algorithm $\mathcal{D}$ distinguishes X and Y with advantage ϵ if and only if

$$|\Pr[\mathcal{D}(X) = 1] - \Pr[\mathcal{D}(Y) = 1]\,| \geq \epsilon.$$

The statistical distance $\Delta(X, Y)$ is an upper bound on the advantage of any distinguisher for X and Y (see, e.g., [Lub94, Exercise 22]).

A formal definition of pseudorandom generator is proposed by Yao [Yao82]. Consider a deterministic algorithm PRG that transforms n-bit strings into M-bit strings, where $M > n$. Let U_l denote a random variable uniformly distributed on $\mathbb{Z}_l$, $l > 0$ (below l will be 2^M or 2^n). We say that an algorithm is T-*time* if it halts in at most T time units. A T-time algorithm $\mathcal{D}$ is called a (T, ϵ)-distinguisher for PRG if

$$|\Pr[\mathcal{D}(\mathsf{PRG}(U_{2^n})) = 1] - \Pr[\mathcal{D}(U_{2^M}) = 1]\,| \geq \epsilon. \tag{2.1}$$

One can think of the distinguisher as an algorithm that tries to guess whether the input is a sequence produced by PRG or it is a uniformly random sequence.

Definition 2.2.1 *Algorithm* PRG *is called a* (T, ϵ)*-secure* pseudorandom generator *if no* (T, ϵ)*-distinguisher exists for* PRG.

The distinguisher $\mathcal{D}$ is sometimes called a statistical test or an adversary. The parameter ϵ is called the advantage of the adversary. The parameter n denotes the size of the seed and the parameter M denotes the size of the pseudorandom sequence.

2.3 Integer factorization and the RSA problem

To analyze concrete security of pseudorandom generators based on the intractability of computing discrete logarithms, integer factorization, and related problems, the complexity of the most efficient methods for solving these problems has to be determined. In this section, we consider integer factorization and the RSA problem.

2.3.1 Integer factorization

The most efficient method for integer factorization today is the Number Field Sieve abbreviated as NFS (see, e.g., [LV01]). It factors an n-bit integer in asymptotic expected time

$$L(n) = \exp[(1.9229 + o(1))(n \ln 2)^{1/3}(\ln(n \ln 2))^{2/3}],$$

as n tends to infinity. To get an explicit formula for the complexity of the NFS, we assume that

$$L(n) = A \exp[1.9229(n \ln 2)^{1/3}(\ln(n \ln 2))^{2/3}],$$

where A is a constant, and estimate A from experimental data. Factoring a 512-bit integer is reported to take about $3 \cdot 10^{17}$ time units [LV01]. It implies that $A \approx 1.7 \cdot 10^{-2}$ and thus

$$L(n) = 1.7 \cdot 10^{-2} \exp[1.9229(n \ln 2)^{1/3}(\ln(n \ln 2))^{2/3}]. \tag{2.2}$$

2.3.2 The RSA problem

Related to integer factorization is the problem of finding e-th roots modulo a composite number N whose factors are not known. The latter problem is referred to as the RSA problem since it implies decrypting a ciphertext in the context of the RSA cryptosystem [RSA78] without any information about the private key. The formal definition of the RSA problem is below.

Definition 2.3.1 *Let N be a product of two primes and let $e > 1$ be coprime to $\phi(N)$, where ϕ denotes Euler's totient function. Let $y \in \mathbb{Z}_N^*$. The* RSA problem *is to find integer $x \in \mathbb{Z}_N^*$ such that $x^e \equiv y \bmod N$ given e, y, and N.*

Clearly, the RSA problem can be solved by factoring N. The reverse reduction is not known. Boneh and Venkatesan [BV98] give an evidence that the RSA problem may be easier than factoring. In fact, they show that any efficient algebraic reduction from factoring to solving the RSA problem with low public exponent e can be converted into an efficient factoring algorithm. Nevertheless, the common practice is to assume that the RSA problem is as hard as integer factorization.

Assumption 2.3.2 *No algorithm solves the RSA problem for an n-bit modulus in time T with probability ϵ if $T/\epsilon < L(n)$.*

Note that the NFS algorithm is only effective if run to completion [LV01]. In other words, running the NFS for 10% does not yield the solution with probability 0.1. Thus, the bound $T/\epsilon < L(n)$ used in Assumption 2.3.2 is a conservative bound.

2.3.3 The flexible RSA problem

In Chapter 6, the following variant of the RSA problem is used.

Definition 2.3.3 *Let N be a product of two primes and let $y \in \mathbb{Z}_N^*$. The* flexible RSA problem *is to find $x \in \mathbb{Z}_N^*$ and $e > 1$ such that e is coprime with $\phi(N)$ and $x^e \equiv y \bmod N$ given y and N.*

The flexible RSA problem is first considered by Barić and Pfitzmann [BP97]. As opposed to the ordinary RSA problem, in this case parameter e is not fixed so the adversary gets more freedom. Therefore, the flexible RSA problem is not harder than the ordinary RSA problem. On the other hand, at this moment the most efficient method for solving the flexible RSA problem is to solve the ordinary RSA problem.

Assumption 2.3.4 *No algorithm solves the flexible RSA problem for an n-bit modulus in time T with probability ϵ if $T/\epsilon < L(n)$.*

Since the flexible RSA problem is potentially easier than the ordinary RSA problem, we say that Assumption 2.3.4 is stronger than Assumption 2.3.2. Assumption 2.3.4 is referred to as the strong RSA assumption [CS00].

2.4 The discrete logarithm problem and its variants

This section is about the state of the art in solving the discrete logarithm problem and its variants.

2.4.1 The discrete logarithm problem

Let $\mathbb{G}$ be a multiplicative group of prime order q. For $x, y \in \mathbb{G}$, $x \neq 1$, and $s \in \mathbb{Z}_q$ such that $y = x^s$, s is called the discrete logarithm of y to the base x. We write $s = \log_x y$.

Definition 2.4.1 *The* discrete logarithm problem (the DL problem) *is to find* $\log_x y$ *given* x *and* y.

The complexity of the DL problem depends on the group $\mathbb{G}$. For instance, if $\mathbb{G}$ is a prime order subgroup of $\mathbb{Z}_p^*$, the most efficient methods for solving the DL problem in $\mathbb{G}$ are Pollard's rho method in $\mathbb{G}$ and the discrete logarithm variant of the Number Field Sieve (DLNFS) in the full multiplicative group $\mathbb{Z}_p^*$. The running time of Pollard's rho method is estimated to be $1.25\sqrt{q}$ group operations, where q denotes the order of $\mathbb{G}$ (see, e.g., [LV01]). Thus, for $n = \lceil \log_2(q+1) \rceil$ and $m = \lceil \log_2(p+1) \rceil$, the running time of Pollard's rho method is about $1.25 \cdot 2^{n/2} m^2/24$ time units. In turn, the running time of the DLNFS is about the same as the running time of the NFS (cf. [LV00]).

Throughout this thesis, we assume that no algorithm solves the DL problem in a subgroup of $\mathbb{Z}_p^*$ faster than Pollard's rho method and the DLNFS. Due to the random-self-reducibility of the DL problem, we can formulate this assumption as follows.

Assumption 2.4.2 *Let $\mathbb{G}$ be a subgroup of $\mathbb{Z}_p^*$ of prime order q. Let m be the bit length of p and let n be the bit length of q. Then no algorithm solves the DL problem in $\mathbb{G}$ in time T with probability ϵ if $T/\epsilon < \min[L(m),\ 1.25 \cdot 2^{n/2} m^2/24]$.*

If $\mathbb{G}$ is a group of points of an ordinary elliptic curve, the DLNFS is not applicable so the most efficient algorithm of solving the elliptic curve discrete logarithm problem (the ECDL problem) is the exponential Pollard's rho method.

2.4.2 The decisional Diffie-Hellman problem

The pseudorandom generator presented in Chapter 4 relies on the intractability of the following well-known problem.

Definition 2.4.3 *Let $X_{DDH} \in \mathbb{G}^4$ be a random variable uniformly distributed on the set consisting of all 4-tuples $(x, y, v, w) \in \mathbb{G}^4$ such that $\log_x v = \log_y w$ and let Y_{DDH} be chosen uniformly at random from $\mathbb{G}^4$. Algorithm $\mathcal{D}$ is said to solve the* decisional Diffie-Hellman problem (the DDH problem) *in $\mathbb{G}$ with advantage ϵ if it distinguishes the random variables X_{DDH} and Y_{DDH} with advantage ϵ, that is,*

$$|\Pr[\mathcal{D}(X_{DDH}) = 1] - \Pr[\mathcal{D}(Y_{DDH}) = 1]\ | \geq \epsilon.$$

The DDH problem originates from the security analysis of the Diffie-Hellman key exchange protocol [DH76]. Related to the DDH problem is the *computational Diffie-Hellman problem* (given x, y and x^s, compute y^s) abbreviated as the *CDH problem.*

Clearly, the DL problem is at least as hard as the CDH problem. The CDH problem is proved to be equivalent to the DL problem under certain conditions [Mau94; MW96]. Moreover, no groups are known such that the CDH problem is strictly easier to solve than the DL problem. Usually these two problems are assumed to be equally hard.

On the other hand, there exist groups (e.g., $\mathbb{Z}_p^*$) in which a random instance of the CDH problem is believed to be hard while the DDH problem is trivial. The latter groups are referred to as the non-DDH groups [GKR04] or the *gap groups* [BCP03]. Furthermore, Wolf [Wol99] shows that for *all* groups $\mathbb{G}$ an algorithm that solves the DDH problem in $\mathbb{G}$ is of no help for solving the CDH problem in $\mathbb{G}$.

However, the computational gap between the DDH problem and the CDH problem is difficult to estimate. It is believed that except for the gap groups, there is no way to solve the DDH problem rather than to solve the CDH problem. For constructing our pseudorandom generator in Chapter 4, we do *not* use the gap groups (we use, for instance, prime order subgroups of $\mathbb{Z}_p^*$). Therefore, to compute security parameters for the new generator, we assume that the DDH problem and the DL problem in this group are equally hard, in agreement with common practice.

The following assumption is about intractability of the DDH problem in a prime order subgroup of $\mathbb{Z}_p^*$.

Assumption 2.4.4 *Let $\mathbb{G}$ be a subgroup of $\mathbb{Z}_p^*$ of prime order q. Let m be the bit length of p and let n be the bit length of q. Then, no T-time algorithm solves the DDH problem in $\mathbb{G}$ with probability ϵ if $T/\epsilon < \min[L(m),\ 1.25 \cdot 2^{n/2} m^2/24]$.*

Similarly to the situation in Section 2.3, Assumption 2.4.4 is said to be stronger than Assumption 2.4.2.

2.4.3 The DLSE problem

Another variant of the discrete logarithm problem is the discrete logarithm with short exponents problem (the DLSE problem) introduced by van Oorschot and Wiener [vOW96].

Definition 2.4.5 *Let $x, y \in \mathbb{G}$ and let c be a positive integer. The c-*DLSE problem *is to find s, $0 \le s < 2^c$, such that $y = x^s$ given x, y, and c (if such an s exists).*

Clearly, the DLSE problem is not more intractable than the original discrete logarithm problem.

In the case of $\mathbb{Z}_p^*$, the fastest algorithms for solving the DLSE problem are the discrete logarithm variant of the NFS and Pollard's lambda method . The complexity of the latter is close to $2 \cdot 2^{c/2}$ multiplications in $\mathbb{Z}_p^*$, that is, $2^{c/2+1} n^2/24$ time units (cf. [Pol00]).

Assumption 2.4.6 *No T-time algorithm solves the DLSE problem in $\mathbb{Z}_p^*$ with probability ϵ if $T/\epsilon < \min[L(n),\ 2^{c/2+1}n^2/24]$.*

2.4.4 The x-logarithm problem

The x-logarithm problem [Bro06; BG07] is a variant of the discrete logarithm problem which arises in the setting of elliptic curve cryptography.

Let $\mathbb{F}_p$ be a prime field and let $E(\mathbb{F}_p)$ be an elliptic curve over $\mathbb{F}$. The curve forms a group under the standard chord-and-tangent addition. Assume that the curve is chosen in such a way that the order of the group $\#E(\mathbb{F}_p)$ is prime so any nonzero element of the group is a generator. Let $\mathrm{x} : E(\mathbb{F}_p) \mapsto \{0, 1, \ldots, p-1\}$ be the function that gives the x-coordinate of a point on the curve interpreted as an integer.

Definition 2.4.7 *Let $r \in_R \{0, 1, \ldots, \#E(\mathbb{F}_p) - 1\}$ and $R \in_R E(\mathbb{F}_p)$. Let P be a nonzero element of the group $E(\mathbb{F})$. The x-*logarithm problem* is to distinguish rP from $\mathrm{x}(R)P$, given P.*

In the above definition and throughout this thesis, notation "$s \in_R S$" means that the element x is chosen uniformly at random from the set S.

Brown [Bro06] conjectures that the x-logarithm problem is as hard as the ECDL problem. This fact, however, seems very hard to prove. For this thesis, we do not need an explicit assumption about the intractability of the x-logarithm problem.

Chapter 3

Cryptanalysis of the Dual Elliptic Curve pseudorandom generator

Anyone who considers arithmetical methods of producing random digits is, of course, in a state of sin.

John von Neumann

In this chapter, we analyse the security of the Dual Elliptic Curve generator that appears in NIST SP800-90 [BK05]. We show that even though breaking the generator is claimed to be as hard as solving a difficult problem the generator is in fact insecure. An efficient attack is presented.

This chapter is based on the joint work with B. Schoenmakers [SS06].

3.1 Introduction

The Dual Elliptic Curve pseudorandom generator (the DEC generator) is proposed by Barker and Kelsey [BK05]. It is claimed in [BK05, Section 10.3.1] that the pseudorandom generator is secure unless the adversary can solve the elliptic curve discrete logarithm problem (the ECDL problem, see Section 2.4.1) for the corresponding elliptic curve. However, the claim is supported only by an informal discussion. No security reduction is given, that is, it is not shown that an adversary that breaks the pseudorandom generator implies a solver for the ECDL problem.

We argue that the DEC generator is *insecure* and we provide experimental evidence of that. The attack does *not* imply solving the ECDL problem for the corresponding elliptic curve. Our method is sufficiently efficient so that it can be implemented on an ordinary PC.

Actually, the generator is insecure because pseudorandom bits are extracted from points of the elliptic curve improperly. The authors of [BK05] assume that the 240 least significant bits of the x-coordinate of a random point of the elliptic curve over the prime field $\mathbb{F}_p$, where $\lceil \log_2 p \rceil = 256$, are indistinguishable from 240 uniformly distributed random bits. We show that this is not the case. Based on this observation, we construct an algorithm (an adversary) that efficiently distinguishes the pseudorandom sequences produced by the DEC generator from the sequences of uniformly distributed random bits.

3.2 The DEC generator

Before describing the DEC generator, some notation has to be introduced.

Let $p = 2^{256} - 2^{224} + 2^{192} + 2^{96} - 1$. Let $E(\mathbb{F}_p)$ denote the elliptic curve over the prime field $\mathbb{F}_p$ consisting of all pairs $(x, y) \in \mathbb{F}_p \times \mathbb{F}_p$ such that

$$y^2 = x^3 + ax + b,$$

$a, b \in \mathbb{F}_p$, and a point at infinity $\mathcal{O}$. The elliptic curve equipped with the standard chord-and-tangent addition forms a group. The field elements a and b are chosen in such a way that the order of the group $\#E(\mathbb{F}_p)$ is prime so the group is cyclic and any nonzero element of the group is a generator (see also [BK05, Appendix A.1]). According to Hasse's theorem

$$|\#E(\mathbb{F}_p) - p - 1| \leq 2\sqrt{p},$$

so in our case $\#E(\mathbb{F}_p) \approx 2^{256}$.

Recall that $\mathrm{x} : E(\mathbb{F}_p) \mapsto \{0, 1, \ldots, p-1\}$ is the function that gives the x-coordinate of a point on the curve interpreted as an integer (see also Section 2.4.4). Let $L_i(s) = s \bmod 2^i$, for $s, i \in \mathbb{Z}$, $i > 0$.

Let $P = (x_P, y_P)$ and $Q = (x_Q, y_Q)$ be two points on the curve $E(\mathbb{F}_p)$. Note that in general the number α such that $P = \alpha Q$ is difficult to compute due to the intractability of the ECDL problem. The points P and Q are the system parameters of the DEC generator. Their coordinates are specified in [BK05, Appendix A.1]. The seed of the DEC generator is a random integer $s_0 \in_R \{0, 1, \ldots, \#E(\mathbb{F}_p) - 1\}$. The DEC generator (Algorithm 3.2.1) transforms the seed into the pseudorandom sequence of length $240k$, $k > 0$.

Algorithm 3.2.1 The Dual Elliptic Curve pseudorandom generator

```
Input: $s_0 \in \{0, 1, \ldots, \#E(\mathbb{F}_p) - 1\}$, system parameters $P, Q \in E(\mathbb{F}_p)$, $k > 0$
Output: $240k$ bits
  for $i = 1$ to $k$ do
    set $s_i \leftarrow \mathrm{x}(s_{i-1}P)$
    set $r_i \leftarrow L_{240}(\mathrm{x}(s_iQ))$
  end for
  return $r_1, \ldots, r_k$
```

Observe that for $i \geq 1$, the i-th state of the generator s_i is a number from the set $\{0, 1, \ldots, p-1\}$ while the seed s_0 is taken from the set $\{0, 1, \ldots, \#E(\mathbb{F}_p)-1\}$. It may seem to be a problem because $p \neq \#E(\mathbb{F}_p)$ so s_iP is not distributed uniformly at random in the set $E(\mathbb{F}_p)$. Fortunately, the statistical distance $\Delta(U_{\#E(\mathbb{F}_p)}, U_p) \approx 2^{-128}$, which is negligible, so the above fact does not affect the security of the pseudorandom generator (the notion of statistical distance is introduced in Section 2.2).

A more serious issue is that in the original version of the DEC generator [BK05] the seed s_0 is chosen uniformly at random from the set $\{0, 1, \ldots, 2^{256}-1\}$ rather than from the set $\{0, 1, \ldots, \#E(\mathbb{F}_p)-1\}$. Now, the statistical distance $\Delta(U_{2^{256}}, U_{\#E(\mathbb{F}_p)}) \approx 2^{-32}$, which is *not* negligible. Nevertheless, we do not know how to explore this weakness of the original version, since the value of s_0 is not directly used to produce an output of the pseudorandom generator. The attack presented in the next section is based on different ideas. The version described above (with $s_0 \in_R \{0, 1, \ldots, \#E(\mathbb{F}_p) - 1\}$) is due to [Bro06; BG07].

3.3 The distinguishing attack on the DEC generator

3.3.1 General idea

It is shown by Brown and [Bro06] that the sequence of points s_iQ is indistinguishable from the sequence of points chosen uniformly at random under the assumption that the DDH problem in $E(\mathbb{F}_p)$ and the x-logarithm problem in this group are intractable (see Section 2.4.1). Therefore, it is reasonable to assume that for all $i = 1, \ldots, k$, s_iQ behaves like a random point on the curve.

The points of the elliptic curve can be divided into 2^{240} equivalence classes as follows. We say that a point $R \in E(\mathbb{F}_p)$ belongs to i-th equivalence class, $i \in \{0, 1, \ldots, 2^{240} - 1\}$, if and only if $L_{240}(\mathrm{x}(R)) = i$.

Now, we introduce two random variables. Let X denote the size of i-th equivalence class for $i \in_R \{0, 1, \ldots, 2^{240} - 1\}$. Let Y denote the size of an equivalence class in which a random point $R \in_R E(\mathbb{F}_p)$ lies.

Remarkably, if random variables X and Y can be efficiently distinguished, that is, if there exists a distinguisher $\mathcal{D}$ with a reasonable running time such that

$$|\Pr[\mathcal{D}(X) = 1] - \Pr[\mathcal{D}(Y) = 1]| = \epsilon, \tag{3.1}$$

for a non-negligible ϵ, then the output of the DEC generator can be efficiently distinguished from the sequence of independent uniformly distributed random bits. Indeed, suppose X and Y can be distinguished. Consider a sequence of $240k$ bits. Our goal is to guess if the sequence is produced by the DEC generator or it is just a sequence of independent uniformly distributed bits. Let Z be the size of the equivalence class corresponding, say, to the first 240 bits of the sequence. If the sequence is produced by the DEC generator, Z is distributed similarly to Y; otherwise, Z is distributed similarly to X.

The question is, can the random variables X and Y be distinguished? Our experiments and also our empirical argument show that it is possible. The crucial point is that the expected value of Y is higher than the expected value of X. The reason why $\mathsf{E}(Y)$ exceeds $\mathsf{E}(X)$ is that when measuring the size of the equivalence class in which a randomly chosen point lies the larger equivalence classes have more chances to be hit. In fact, it turns out that $\mathsf{E}(Y) \approx \mathsf{E}(X)+1$. The details are below.

Lemma 3.3.1 *For the random variables X and Y defined above,*

$$\mathsf{E}(Y) = \mathsf{E}(X) + \frac{\mathsf{Var}(X)}{\mathsf{E}(X)}.$$

Proof: Clearly,

$$\mathsf{E}(X) = \frac{\#E(\mathbb{F}_p)}{2^{240}}.$$

Let $s > 0$. Let $C_1^s, C_2^s \dots, C_{t_s}^s$ be the equivalence classes of size s, where t_s denotes the total number of such equivalence classes. Observe that

$$t_s = 2^{240}\Pr[X = s].$$

Let $R \in_R E(\mathbb{F}_p)$. Then, X and Y are connected as follows.

$$\Pr[Y = s] = \sum_{i=1}^{t_s} \Pr[R \in C_i^s] = t_s \cdot \frac{s}{\#E(\mathbb{F}_p)} = \frac{2^{240}\Pr[X = s]s}{\#E(\mathbb{F}_p)} = \frac{\Pr[X = s]s}{\mathsf{E}(X)}. \quad (3.2)$$

It implies that

$$\mathsf{E}(Y) = \frac{1}{\mathsf{E}(X)} \sum_s s^2 \Pr[X = s].$$

Note that

$$\mathsf{Var}(X) = \mathsf{E}(X^2) - \mathsf{E}(X)^2 = \sum_s s^2 \Pr[X = s] - \mathsf{E}(X)^2.$$

Therefore,

$$\mathsf{E}(Y) = \frac{1}{\mathsf{E}(X)}(\mathsf{E}(X)^2 + \mathsf{Var}(X)) = \mathsf{E}(X) + \frac{\mathsf{Var}(X)}{\mathsf{E}(X)}.$$

□

To determine the distribution of the random variable X and, in particular, $\mathsf{Var}(X)$, we have performed a simulation. We have generated 1320000 random 240-bit strings and measured the sizes of the corresponding equivalence classes. The 240-bit strings have been divided into 330 files each consisting of 4000 strings. The analysis of one file have taken about 2 hours and 30 minutes on a 3GHz Linux machine with 1Gb of memory.

To produce the random bit strings, we have used the `RtlGenRandom()` generator of the Platform SDK. This pseudorandom generator is built according to FIPS 186-2 Appendix 3.1 with SHA1 as the iterated function [DSS00]. It gets the seed from the current system information (current process ID, current thread, current time, etc.). The on-line documentation and also our own evaluation indicate that the output of the generator is not biased so the generator is suitable for statistical measurements.

The simulation implies that $\mathsf{Var}[X] \approx \mathsf{E}(X)$, and therefore $\mathsf{E}(Y) \approx \mathsf{E}(X) + 1$.

3.3.2 Probability distributions of X and Y

The fact that $\mathsf{E}(Y) \approx \mathsf{E}(X) + 1$ suggests the following simple way to tell apart X and Y. Let $\mathcal{D}$ be a distinguisher that given Z, which is either X or Y, outputs 1 if and only if $Z > \mathsf{E}(X)$. Let ϵ be the distinguishing advantage of $\mathcal{D}$. To estimate ϵ, it is not sufficient to know the expected values and the variances of X and Y. It is important to determine the probability distributions of these random variables.

Let $R \in E(\mathbb{F}_p)$ and $i \in_R \{0, 1, \ldots, 2^{240} - 1\}$. The probability that R belongs to the i-th equivalence class is $1/2^{240}$. Now, suppose we consider all points of the curve one after the other and measure how many of them are in the i-th equivalence class. Assume that each point belongs to the i-th equivalence class with probability close to $1/2^{240}$. Then, the size of the i-th equivalence class (denoted by X) is distributed according to binomial distribution with parameters $n = \#E(\mathbb{F}_p)$ and $\delta = 1/2^{240}$, that is, for all $s \geq 0$,

$$\Pr[X = s] = \binom{n}{s} \delta^s (1 - \delta)^{n-s}.$$

Note that δ is quite small. It is well-known that for $\delta \to 0$ and for the fixed average value $\mathsf{E}(X) = n\delta$ (this limit is sometimes known as the law of rare events) the binomial distribution approaches the Poisson distribution with parameter $\lambda = n\delta$ so

$$\Pr[X = s] \approx \frac{\lambda^s e^{-\lambda}}{s!}.$$

In the case of the Poisson distribution, the expected value always equals the variance (see Lemma 3.3.1).

Our experiments confirm the above informal argument. They demonstrate that X is statistically close to a Poisson-distributed random variable with parameter $\lambda = \#E(\mathbb{F}_p)/2^{240}$ (see Figure 3.1). From now on, we assume that X is indeed a Poisson-distributed random variable.

Lemma 3.3.2 *If X is a Poisson-distributed random variable, then for all $s > 0$ $\Pr[Y = s] = \Pr[X = s - 1]$, so the distribution of Y is just a shifted distribution of X.*

Proof: Random variables X and Y are connected by Formula (3.2):

$$\Pr[Y = s] = \frac{\Pr[X = s]s}{\mathsf{E}(X)} = \frac{\lambda^s e^{-\lambda}}{s!} \cdot \frac{s}{\lambda} = \Pr[X = s - 1].$$

□

3.3.3 Success probability of the distinguishing attack

Finally, we estimate the advantage ϵ of the distinguisher $\mathcal{D}$ defined in the beginning of Section 3.3.2.

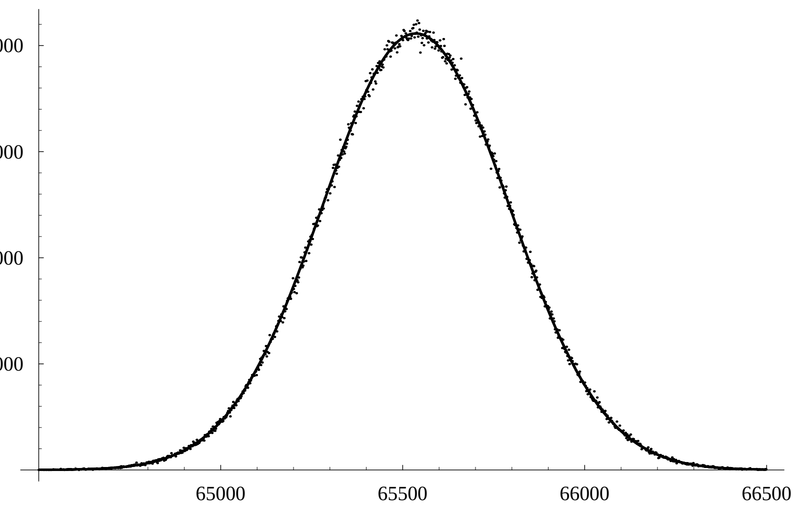

Figure 3.1: Number of equivalence classes as a function of their size (unscaled probability density function of random variable X). In total, 1320000 random 240-bit strings were generated and the sizes of the corresponding equivalence classes were measured. The simulation implies that the probability density function of random variable X is close to that of the Poisson distribution with parameter $\#E(\mathbb{F}_p)/2^{240} \approx 2^{16}$.

On one hand, $\Pr[X > \lambda] \approx 1/2$ (the probability density function of X is almost symmetric). On the other hand,

$$\Pr[Y > \lambda] = \int_{\lambda}^{\infty} \frac{\lambda^{s-1}e^{-\lambda}}{(s-1)!} ds \approx 0.50156.$$

Therefore, $\epsilon = |\Pr[\mathcal{D}(X) = 1] - \Pr[\mathcal{D}(Y) = 1]| \approx 0.00156$, which is a non-negligible value.

The distinguishing advantage can be further improved. Recall that the DEC generator outputs not one 240-bit string but k such strings. One can take into account all the output strings, add together the sizes of the corresponding equivalence classes, and compare the result to $k\lambda$. In terms of X and Y, this means that we consider a distinguisher $\mathcal{D}_k$ that is given $Z_1, \ldots, Z_k$, which are either all instances of X or all instances of Y. Whenever $Z_1 + \ldots + Z_k > k\lambda$, $\mathcal{D}_k$ outputs 1. In our experiments, we used $k = 4000$.

Note that the sum of k Poisson-distributed random variables with parameter λ is a Poisson-distributed random variables with parameter $k\lambda$. Therefore, $\Pr[X_1 + \ldots + X_k > k\lambda] \approx 1/2$ and

$$\Pr[Y_1 + \ldots + Y_k > k\lambda] = \int_{\lambda}^{\infty} \frac{(k\lambda)^{s-1}e^{-k\lambda}}{(s-1)!} ds \approx 0.59757,$$

which means that the output of the DEC generator with $k = 4000$ can be distinguished from a sequence of independent uniformly distributed random bits with advantage almost 10%.

In independent work, Gjøsteen [Gjø06] shows that there exists an algorithm that predicts the next bit of the DEC generator with advantage 0.0011. The work by Gjøsteen is based on ideas similar to those proposed in this chapter.

3.4 Conclusion

Note that the complexity of our attack is proportional to 2^{256-l}, where l is the number of bits extracted from a single point. So, for $l = 240$, the complexity of the attack is about 2^{16}. If one extracts significantly less than 240 bits (for instance, $l = 176$ bits) the attack becomes impractical. However, extracting less random bits does *not* guarantee that there exists no other attack that successfully breaks the pseudorandom generator. The reason is that the DEC generator is *not* provably secure, its security does *not* provably rely on the intractability of the ECDL problem.

The main conclusion of this chapter is that when designing a provably secure cryptographic scheme (e.g., a pseudorandom generator) one has to pay attention to the security proof (the reduction). An informal argument like the one in [BK05, Section 10.3.1] is definitely not good enough. The scheme with a certain choice of parameters can be claimed to be provably secure only if it is shown that breaking the scheme is as hard as solving a difficult problem faster than the fastest algorithm known so far.

In Chapter 4, we discuss possible ways to repair the DEC generator.

CHAPTER 4

Efficient pseudorandom generators based on the DDH assumption

Indistinguishable things are identical.

Gottfried Wilhelm Leibniz

In this chapter, a family of pseudorandom generators based on the DDH assumption is proposed. The new construction is a modified and generalized version of the DEC generator [BK05]. Although in Chapter 3 the original DEC generator is shown to be insecure, the modified version is provably secure and very efficient.

Our generator can be based on any group of prime order provided that an additional requirement is met (i.e., there exists an efficiently computable function that in some sense enumerates the elements of the group). Two specific instances are presented. The techniques used to design the instances, for example, the new probabilistic randomness extractor are of independent interest for other applications.

Additionally, we analyze the algorithm proposed by Juels et al. [JJSH00] that transforms uniformly random numbers into uniformly random bits. This algorithm can be used as a building block for the new pseudorandom generator.

This chapter is an extended version of the joint work with R. R. Farashahi and B. Schoenmakers [FSS07].

4.1 Introduction

As discussed in Section 1.4, a pseudorandom generator can be constructed from any one-way function. Thus, intractability of the discrete logarithm problem suffices to construct a pseudorandom generator. Such a construction was first proposed by Blum and Micali [BM82]. However, the Blum-Micali pseudorandom generator and similar ones are inefficient in the sense that only a single bit is output per modular exponentiation. In this chapter, we show that the stronger assumption that the decisional Diffie-Hellman problem is hard to solve (DDH assumption) gives rise to much more efficient pseudorandom generators.

The DDH assumption is introduced in Section 2.4.2. In comparison with many other assumptions, the DDH assumption is thoroughly studied (for more details about the intractability of the DDH problem, refer e.g. to [NR04]) and has become a basis for a wide variety of cryptographic schemes.

Security of our construction is tightly related to the intractability of the DDH problem.

4.1.1 Related work

This chapter is inspired by the publication of Barker and Kelsey [BK05], in which the the DEC generator is proposed. This generator is thoroughly analyzed in Chapter 3. Now, we briefly recall its basic idea.

Let P and Q be points on a prime order elliptic curve over a prime field $\mathbb{F}_p$ such that p is close to 2^{256}. Let q denote the order of the curve. On input s_0 chosen uniformly at random from $\mathbb{Z}_q$ the DEC generator produces two sequences of points s_iP and s_iQ such that s_i is set to be the x-coordinate of $s_{i-1}P$, $i = 1, 2, \dots, k$. The generator outputs k binary strings each string consisting of the 240 least significant bits of the x-coordinate of s_iQ. The sequence of points s_iQ is shown to be indistinguishable from the sequence of uniformly random points of the elliptic curve under the assumption that the DDH problem and the non-standard x-logarithm problem (see Section 2.4.1) are intractable in $\mathrm{E}(\mathbb{F}_p)$ [BG07]. However, as shown in Chapter 3, the binary sequence produced by the generator is distinguishable from uniform. The reason is that points of the elliptic curve are transformed into random bits in an improper way.

Some ideas of the DEC generator are present in the earlier work by Naor and Reingold [NR04]. Let p be a prime and let g be a generator of a subgroup of $\mathbb{Z}_p^*$ of prime order q. Let $a \in \mathbb{Z}_q$ be a fixed number. Naor and Reingold [NR04] propose a simple function G that on input $b \in \mathbb{Z}_q$ outputs (g^b, g^{ab}). If b is chosen uniformly at random, the output of the function is computationally indistinguishable from uniform under the DDH assumption in the subgroup. Note, however, that function G produces random elements of the subgroup rather than random bits and therefore it is not a pseudorandom generator in the sense of Definition 2.2.1 (converting random elements of the subgroup into random bits is a nontrivial problem). Moreover, although function G doubles the input it cannot be iterated to produce as much pseudorandomness as required by the application. Namely, it is not clear how to produce a new value of b given two group elements g^b and g^{ab}. Admittedly, the goal of Naor and Reingold [NR04] is to construct not a pseudorandom generator but a pseudorandom function, for which function G turns out to be a suitable building block.

4.1.2 Our contributions

A possible way to repair the DEC generator is to use a robust randomness extractor instead of just outputting the 240 least significant bits of the x-coordinates of the corresponding points on the elliptic curve. One of such extractors is proposed by Farashahi et al. [FPS07]. This extractor converts a uniformly random point of an elliptic curve over $\mathbb{F}_{2^n}$ for an even n to $n/2$ random bits statistically close to uniform (the statistical distance is shown to be about $2^{-n/2}$). Thus, for $n = 256$ the updated version of the DEC generator outputs 128 bits per iteration so it is $240/128 \approx 1.9$ times slower than the original generator. On the positive side, the result of Brown [Bro06] implies that the the updated version of the generator is provably secure under the DDH assumption and the x-logarithm assumption. A disadvantage of the updated version is that the x-logarithm assumption is relatively new so it is not extensively studied.

In this chapter, we modify and generalize the DEC generator so that the security of the modified version relies *only* on the DDH assumption. Compared to the original DEC generator, our generator can be based on any group of prime order meeting an additional requirement (i.e., there exists an efficiently computable function that in some sense enumerates the elements of the group). The new generator is more efficient than many other pseudorandom generators based on discrete log assumptions.

We present two specific instances of the new pseudorandom generator.

The first instance is based on the group of quadratic residues modulo a safe prime $p = 2q+1$. This instance uses an idea of Cramer and Shoup [CS04] who show that there exists a simple bijective function that maps quadratic residues modulo p to $\mathbb{Z}_q$.

The second instance is based on an arbitrary prime order subgroup of $\mathbb{Z}_p^*$, where p is prime but not necessarily a safe prime. To construct this instance, we first propose a surprisingly simple probabilistic randomness extractor that provided with some extra randomness converts a uniformly random element of the subgroup of order q to a uniformly random number in $\mathbb{Z}_q$, which in turn can be easily converted to a string of *uniformly random bits* using, for instance, algorithm Q_2 from [JJSH00] (for an overview of probabilistic randomness extractors, refer to [Sha02]). Note that all (probabilistic and deterministic) extractors known so far can only convert random elements of the subgroup to *bits that are statistically close to uniform.*

We derive the security parameters of the new pseudorandom generators from the corresponding security reductions. For this purpose, we make practical assumptions about intractability of the discrete logarithm problems in the corresponding groups.

4.2 DDH generator

In this section, the main result of the chapter is presented. We propose a new provably secure pseudorandom generator. We call it the *DDH generator*, since the security of this generator relies on the intractability of the DDH problem in the corresponding group.

4.2.1 Construction of the generator

Let $\mathbb{G}$ be a multiplicative group of prime order q and let $\mathsf{enum} : \mathbb{G} \times \mathbb{Z}_l \mapsto \mathbb{Z}_q \times \mathbb{Z}_l$, $l > 0$, be a bijection. Thus, on uniformly distributed input, function enum produces uniformly distributed output. Typically, but not necessarily, l is chosen to be small. The advantage of a smaller l is that the seed of the generator is shorter.

The seed of the DDH generator (Algorithm 4.2.1) consists of $s_0 \in_R \mathbb{Z}_q$ and randp_0, $\mathsf{randq}_0 \in_R \mathbb{Z}_l$. The DDH generator transforms the seed into the sequence of $k > 0$ pseudorandom numbers from $\mathbb{Z}_q$.

Algorithm 4.2.1 DDH generator

```
Input: $s_0 \in \mathbb{Z}_q$, $\mathsf{randp}_0 \in \mathbb{Z}_l$, $\mathsf{randq}_0 \in \mathbb{Z}_l$, system parameters $x, y \in_R \mathbb{G}$, $k > 0$
Output: $k$ pseudorandom integers from $\mathbb{Z}_q$
  for $i = 1$ to $k$ do
    set $(s_i, \mathsf{randp}_i) \leftarrow \mathsf{enum}(x^{s_{i-1}}, \mathsf{randp}_{i-1})$
    set $(\mathsf{output}_i, \mathsf{randq}_i) \leftarrow \mathsf{enum}(y^{s_{i-1}}, \mathsf{randq}_{i-1})$
  end for
  return $\mathsf{output}_1, \mathsf{output}_2, \ldots, \mathsf{output}_k$
```

Note that the random elements x and y are not a part of the seed. These two elements are public system parameters. In the security analysis of the generator we assume that x and y are known to the distinguisher.

4.2.2 Security analysis

The following theorem implies that under the DDH assumption for group $\mathbb{G}$ an output sequence of the DDH generator is indistinguishable from a sequence of uniformly random numbers in $\mathbb{Z}_q$.

Theorem 4.2.1 *Suppose there exists a T-time algorithm that distinguishes the output of the DDH generator from the sequence of independent uniformly distributed random numbers in $\mathbb{Z}_q$ with advantage ϵ. Then the DDH problem in $\mathbb{G}$ can be solved in time T with advantage ϵ/k.*

Proof: The proof follows the classical hybrid technique (see, e.g., [Gol01, Section 3.2.3]).

Suppose there exists a T-time algorithm $\mathcal{D}$ that distinguishes the output of the DDH generator from a sequence of independent uniformly distributed random numbers in $\mathbb{Z}_q$ with advantage ϵ, that is,

$$|\Pr[\mathcal{D}(\mathsf{output}_1, \mathsf{output}_2, \ldots, \mathsf{output}_k) = 1] - \Pr[\mathcal{D}(U) = 1]\,| \geq \epsilon,$$

where $U = (u_1, u_2, \ldots, u_k)$, $u_i \in_R \mathbb{Z}_q$, $i = 1, \ldots, k$. Consider the following hybrid random variables.

$$Z_i = (u_1, u_2, \ldots, u_i, \mathsf{output}_1, \mathsf{output}_2, \ldots, \mathsf{output}_{k-i}),$$

$i = 0, 1, \ldots, k$. Note that $Z_0 = (\mathsf{output}_1, \mathsf{output}_2, \ldots, \mathsf{output}_k)$ and $Z_k = U$.

Let $j \in_R \{0, 1, \ldots, k-1\}$. Then $|\Pr[\mathcal{D}(Z_j) = 1] - \Pr[\mathcal{D}(Z_{j+1}) = 1]| \geq \epsilon/k$, where the probability is taken not only over internal coin flips of $\mathcal{D}$ but also over the choice of j. Indeed,

$$\begin{aligned}
& |\Pr[\mathcal{D}(Z_j) = 1] - \Pr[\mathcal{D}(Z_{j+1}) = 1]\,| \\
={} & \sum_{i=0}^{k-1} \Pr[j = i] \cdot |\Pr[\mathcal{D}(Z_i) = 1] - \Pr[\mathcal{D}(Z_{i+1}) = 1]\,| \\
\geq{} & \frac{1}{k} \left| \sum_{i=0}^{k-1} \Pr[\mathcal{D}(Z_i) = 1] - \Pr[\mathcal{D}(Z_{i+1}) = 1] \right| \\
={} & \frac{1}{k} \left|\Pr[\mathcal{D}(Z_0) = 1] - \Pr[\mathcal{D}(Z_k) = 1]\right| \geq \epsilon/k.
\end{aligned}$$

Now, we show how to solve the DDH problem in $\mathbb{G}$ using the distinguisher $\mathcal{D}$ as a building block. Let $(x, y, v, w) \in \mathbb{G}^4$. A solver for the DDH problem decides if $\log_x v = \log_y w$ or v and w are independent uniformly distributed random elements of $\mathbb{G}$ as follows:

```
select $j \in_R \{0, 1, \ldots, k-1\}$
select $r_1, r_2, \ldots, r_{j-1} \in_R \mathbb{Z}_q$, $\mathsf{randp}_0 \in_R \mathbb{Z}_l$, $\mathsf{randq}_0 \in_R \mathbb{Z}_l$
set $(s_1, \mathsf{randp}_1) \leftarrow \mathsf{enum}(v, \mathsf{randp}_0)$
set $(r_j, \mathsf{randq}_1) \leftarrow \mathsf{enum}(w, \mathsf{randq}_0)$
for $i = 1$ to $k - j$ do
    set $(s_{i+1}, \mathsf{randp}_{i+1}) \leftarrow \mathsf{enum}(x^{s_i}, \mathsf{randp}_i)$
    set $(r_{i+j}, \mathsf{randq}_{i+1}) \leftarrow \mathsf{enum}(y^{s_i}, \mathsf{randq}_i)$
end for
set $Z \leftarrow (r_1, r_2, \ldots, r_k)$
return $\mathcal{D}(Z)$
```

If there exists $s_0 \in \mathbb{Z}_q$ such that $v = x^{s_0}$ and $w = y^{s_0}$ then r_j and r_{j+1} are distributed as the first and the second outputs of the DDH generator respectively, so Z is distributed as Z_j.

Otherwise, if v and w are independent uniformly distributed random elements of $\mathbb{G}$ then r_{j+1} is distributed as the first output of the DDH generator while r_j is uniformly distributed over $\mathbb{Z}_q$ and independent of r_{j+1}, so Z is distributed as Z_{j+1}.

Therefore, the above algorithm solves the DDH problem in $\mathbb{G}$ in time at most T with advantage ϵ/k. □

The DDH generator is not a pseudorandom generator in the sense of Definition 2.2.1. It outputs numbers in $\mathbb{Z}_q$ rather than bits. However, converting random numbers to random bits is a relatively easy problem. For instance, one can use Algorithm Q_2 from [JJSH00] which produces on average $n-2$ bits given a uniformly distributed random number U_q, where n denotes the bit length of q (see also Section

4.5). In the latter case, the average number of bits produced by the generator is $k(n-2)$.

For the sake of simplicity, in Sections 4.3 and 4.4, we assume that q is close to a power of 2, that is, $0 \leq (2^n - q)/2^n \leq \delta$ for a small δ. So, the uniform element U_q is statistically close to n uniformly random bits.

The following simple lemma is a well-known result. We reproduce the proof of this lemma for the sake of completeness.

Lemma 4.2.2 *Under the condition that $0 \leq (2^n - q)/2^n \leq \delta$, the statistical distance between U_q and U_{2^n} is bounded above by δ.*

Proof: Let $X \in_R \mathbb{Z}_{2^n}$ and $Y \in_R \mathbb{Z}_q$. Then

$$\begin{aligned}
2\Delta(U_q, U_{2^n}) &= \sum_{s=0}^{2^n-1} |\Pr[X=s] - \Pr[Y=s]| \\
&= \sum_{s=0}^{q-1} |\Pr[X=s] - \Pr[Y=s]| + \sum_{s=q}^{2^n-1} |\Pr[X=s] - \Pr[Y=s]| \\
&= \sum_{s=0}^{q-1} \left|\frac{1}{2^n} - \frac{1}{q}\right| + \sum_{s=q}^{2^n-1} \left|\frac{1}{2^n} - 0\right| = q \cdot \left|\frac{1}{2^n} - \frac{1}{q}\right| + (2^n - q) \cdot \frac{1}{2^n} \\
&= 2 \cdot \frac{2^n - q}{2^n} \leq 2\delta.
\end{aligned}$$

□

The next statement implies that if q is close to a power of 2, the DDH generator is a cryptographically secure pseudorandom generator under the DDH assumption in $\mathbb{G}$.

Corollary 4.2.3 *Let $0 \leq (2^n - q)/2^n \leq \delta$. Suppose the DDH generator is not (T, ϵ)-secure. Then, if $\epsilon/k - \delta \geq 0$, there exists an algorithm that solves the DDH problem in $\mathbb{G}$ in time at most T with advantage $\epsilon/k - \delta$.*

Proof: Suppose there exists a distinguisher $\mathcal{D}$ that runs in time at most T and

$$|\Pr[\mathcal{D}(\mathsf{output}_1, \mathsf{output}_2, \ldots, \mathsf{output}_k) = 1] - \Pr[\mathcal{D}(U_{2^{kn}}) = 1]| \geq \epsilon.$$

Let $u_i \in_R \mathbb{Z}_q$, $i = 1, 2, \ldots, k$, and $U = (u_1, \ldots, u_k)$. Lemma 4.2.2 implies that the statistical distance $\Delta(U, U_{2^{kn}}) \leq k\delta$. Thus,

$$|\Pr[\mathcal{D}(\mathsf{output}_1, \mathsf{output}_2, \ldots, \mathsf{output}_k) = 1] - \Pr[\mathcal{D}(U) = 1]| \geq \epsilon - k\delta.$$

Now, the statement follows from Theorem 4.2.1. □

4.3 Specific instances of the DDH generator

To implement the DDH generator, one has to choose the group $\mathbb{G}$ of prime order q and function enum that enumerates the group elements. In this section, we propose two specific instances of the DDH generator.

Throughout this section, we assume that q is close to a power of 2, that is, $0 \leq (2^n - q)/2^n \leq \delta$ for a small δ and some integer n. We like to emphasize that this assumption is made for the sake of simplicity only. M denotes the total number of pseudorandom bits produced by the generator.

4.3.1 Group of quadratic residues modulo safe prime

To construct the first instance of the DDH generator, we use an idea of Cramer and Shoup [CS04] who show that there exists a simple deterministic function that enumerates elements of the group of quadratic residues modulo safe prime.

Let p be a safe prime, $p = 2q + 1$, where q is prime. Let $\mathbb{G}_1$ be a group of nonzero quadratic residues modulo p. The order of $\mathbb{G}_1$ equals q. Consider the following function $\mathsf{enum}_1 : \mathbb{G}_1 \mapsto \mathbb{Z}_q$,

$$\mathsf{enum}_1(x) = \begin{cases} x, \text{ if } 1 \leq x < q; \\ p - x, \text{ if } q + 2 \leq x < p; \\ 0, \text{ otherwise.} \end{cases}$$

It is shown in [CS04] that function enum_1 is a bijection. For completeness, we reproduce the proof of this fact here.

Lemma 4.3.1 *Function* enum_1 *defined above is a bijection.*

Proof: Since q is odd, there are two possibilities: either $q \equiv 3 \bmod 4$ or $q \equiv 1 \bmod 4$. In both cases, $p \equiv 3 \bmod 4$ so $\left(\frac{-1}{p}\right) = -1$.

Now, we prove that for all $x \in \mathbb{G}_1$, we have $p - x \notin \mathbb{G}_1$. Indeed,

$$\left(\frac{p-x}{p}\right) = \left(\frac{-x}{p}\right) = -\left(\frac{x}{p}\right) = -1.$$

In particular, it implies that either $q \in \mathbb{G}_1$ or $q + 1 \in \mathbb{G}_1$ but not both. □

Note that enum_1 does not require any additional input so in terms of Section 4.2.1 $l = 1$.

Let $s_0 \in_R \mathbb{Z}_q$ be the seed. Generator PRG_1 (Algorithm 4.3.1) transforms the seed into a sequence of kn pseudorandom bits.

The next statement follows from Corollary 4.2.3.

Proposition 4.3.2 *Suppose pseudorandom generator* PRG_1 *is not* (T, ϵ)*-secure. Then, if* $\epsilon/k - \delta \geq 0$*, there exists an algorithm that solves the DDH problem in* $\mathbb{G}_1$ *in time at most* T *with advantage* $\epsilon/k - \delta$*.*

Algorithm 4.3.1 Generator PRG_1

```
Input: $s_0 \in \mathbb{Z}_q$, system parameters $x, y \in_R \mathbb{G}_1$, $k > 0$
Output: $kn$ pseudorandom bits
  for $i = 1$ to $k$ do
    set $s_i \leftarrow \mathsf{enum}_1(x^{s_{i-1}})$
    set $\mathsf{output}_i = \mathsf{enum}_1(y^{s_{i-1}})$
  end for
  return $\mathsf{output}_1, \mathsf{output}_2, \ldots, \mathsf{output}_k$
```

The seed length n plays the role of security parameter of the generator. Clearly, smaller n gives rise to a faster generator. On the other hand, for larger n the generator is more secure. Our goal is to select n as small as possible such that the generator is $(2^{88}, 0.01)$-secure.

Let $\delta = \epsilon/(2k)$. Then, it follows from Proposition 4.3.2 under the DDH assumption (Assumption 2.4.4) that the generator is (T, ϵ)-secure if $2kT/\epsilon < L(n)$, where $L(n)$ is the complexity of the discrete logarithm variant of the Number Field Sieve (cf. Section 2.4.2). Since $k = M/n$, we get

$$2MT/(n\epsilon) < L(n). \tag{4.1}$$

For $M = 2^{20}$, $T = 2^{88}$ and $\epsilon = 0.01$, the smallest parameter n that satisfies the above inequality is $n \simeq 1800$.

Recall that q satisfies $0 \leq (2^n - q)/2^n \leq \delta$ and we assume that $\delta = \epsilon/(2k)$. For $M = 2^{20}$, $n = 1800$, and $\epsilon = 0.01$, this condition implies that $0 < 2^{1800} - q < 2^{1783}$. There are plenty of safe primes $p = 2q + 1$ such that q satisfies the above condition.

4.3.2 Arbitrary prime order subgroup of $\mathbb{Z}_p^*$

In this section, we show that the DDH generator can not only be based on the group of quadratic residues modulo a safe prime but on any prime order subgroup of $\mathbb{Z}_p^*$, where p is a prime but not necessarily a safe prime.

Let q be a prime factor of $p-1$, $p-1 = lq$, $l \geq 2$, such that $\gcd(l, q) = 1$. If p is a safe prime then $l = 2$. Denote the subgroup of $\mathbb{Z}_p^*$ of order q by $\mathbb{G}_2$. Throughout this section, multiplication of integers is done modulo p.

Let $\mathsf{split} : \mathbb{Z}_p^* \mapsto \mathbb{Z}_q \times \mathbb{Z}_l$ denote a bijection that splits an element of $\mathbb{Z}_p^*$ into two smaller numbers. An example of split is a function that on input $z \in \mathbb{Z}_p^*$ returns $(z-1) \bmod q$ and $\lfloor (z-1)/q \rfloor$. Let $t \in \mathbb{Z}_p^*$ be an element of order l. Let $\mathsf{enum}_2 : \mathbb{G}_2 \times \mathbb{Z}_l \mapsto \mathbb{Z}_q \times \mathbb{Z}_l$ be the following function:

$$\mathsf{enum}_2(x, \mathsf{rand}) = \mathsf{split}(xt^{\mathsf{rand}}),$$

where $x \in \mathbb{G}_2$, $\mathsf{rand} \in \mathbb{Z}_l$. The following lemma shows that enum_2 is a bijection and thus it is suitable for building the DDH generator.

Lemma 4.3.3 *Function* enum_2 *defined above is a bijection.*

Proof: Define $f : \mathbb{G}_2 \times \mathbb{Z}_l \mapsto \mathbb{Z}_p^*$ by $f(x, \mathsf{rand}) = xt^{\mathsf{rand}}$, for $x \in \mathbb{G}_2$, $t \in \mathbb{Z}_p^*$ of order l, and $\mathsf{rand} \in \mathbb{Z}_l$. To prove the statement of the lemma, we first show that f is a bijection.

Suppose that $x_1 t^{\mathsf{rand}_1} = x_2 t^{\mathsf{rand}_2}$ for $x_i \in \mathbb{G}_2$, $\mathsf{rand}_i \in \mathbb{Z}_l$, $i = 1, 2$. Since $x_2 \in \mathbb{G}_2$, $x_2 \neq 0$. Then, $x_1/x_2 = t^{\mathsf{rand}_1 - \mathsf{rand}_2} \in \mathbb{G}_2$, so $t^{q(\mathsf{rand}_1 - \mathsf{rand}_2)} = 1$. Therefore, l divides $q(\mathsf{rand}_1 - \mathsf{rand}_2)$. Since $\gcd(q, l) = 1$, this implies that l divides $\mathsf{rand}_1 - \mathsf{rand}_2$. The latter implies that $\mathsf{rand}_1 = \mathsf{rand}_2$ and thus $x_1 = x_2$.

Therefore, f is indeed a bijection and thus enum_2 is also a bijection as a composition of two bijective functions. □

Let PRG_2 denote the instance of the DDH generator that uses the group $\mathbb{G}_2$ and the function enum_2 defined above. The next statement follows from Corollary 4.2.3.

Proposition 4.3.4 *Suppose pseudorandom generator* PRG_2 *is not* (T, ϵ)*-secure. Then, if* $\epsilon/k - \delta \geq 0$*, there exists an algorithm that solves the DDH problem in* $\mathbb{G}_2$ *in time at most* T *with advantage* $\epsilon/k - \delta$*.*

Let m denote the bit length of p. At each step $i = 1, 2, \dots, n$, pseudorandom generator PRG_2 computes $x^{s_{i-1}}$ and $y^{s_{i-1}}$ and then uses these elements to compute the corresponding outcomes of function enum_2. Therefore, each step implies two modular exponentiations with n-bit exponents and two modular exponentiations with $(m-n)$-bit exponents. Since PRG_2 outputs n bits per step the computational effort per output bit is proportional to m^3/n. Our goal is now to determine parameters m and n that minimize the computational effort under the condition that the generator is $(2^{88}, 0.01)$-secure.

As before, let $\delta = \epsilon/(2k)$. Then, it follows from Proposition 4.3.4 that under Assumption 2.4.4 the generator PRG_2 is (T, ϵ)-secure if

$$2MT/(n\epsilon) < \min[L(m),\ 1.25 \cdot 2^{n/2} m^2/24]. \tag{4.2}$$

For $M = 2^{20}$, $T = 2^{88}$, $\epsilon = 0.01$, the above condition is satisfied for $m \gtrsim 1800$, $n \gtrsim 180$. The computational effort is minimized if $n \simeq m$.

In comparison with PRG_1, the seed of PRG_2 is somewhat longer, although if $n \simeq m$ it is roughly of the same size. Moreover, PRG_2 is less efficient than PRG_1 in terms of computational effort since the computation of enum_2 implies a modular exponentiation while enum_1 implies at most 1 integer subtraction. A significant advantage of PRG_2 versus PRG_1 is that the former can be based on any prime order subgroup of $\mathbb{Z}_p^*$ for any prime p provided that the size of the subgroup is sufficiently large to resist Pollard's rho attack.

4.3.3 Discussion

Function enum_2 used as a building block of generator PRG_2 is of independent interest. The reason is that this function can be viewed as a *probabilistic randomness extractor* (for an overview of probabilistic randomness extractors, refer to [Sha02]). Provided with some extra randomness, it converts a uniformly random element of a subgroup of $\mathbb{Z}_p^*$ of order q to a uniformly random number in $\mathbb{Z}_q$, which in turn can be easily converted to a string of *uniformly random bits* using, for instance, algorithm Q_2 from [JJSH00] (cf. Section 4.5). Note that all (probabilistic and deterministic) extractors known so far can only convert random elements of the subgroup to *bits that are statistically close to uniform.*

Our extractor is more efficient than the known general purpose probabilistic randomness extractors (e.g., the universal hash functions [HILL99]) in terms of the number of extra random bits required. For instance, let $m = 1024$ and $n = 160$ (these parameters are used for the Digital Signature Algorithm developed by FIPS [DSS00]). Suppose the statistical distance to be reached is $\epsilon = 2^{-40}$. Then, the approach based on the universal hash functions converts a random element of the 160-bit subgroup to $n - 2\log_2(1/\epsilon) = 80$ bits requiring $m + n - 2\log_2(1/\epsilon) - 1 = 1103$ extra random bits. In contrast, our approach converts a random element of the subgroup to about 160 bits requiring only $m - n = 844$ extra random bits. Moreover, our extractor outputs uniformly random bits whereas the approach based on the universal hash functions outputs bits statistically close to uniform.

The recently proposed deterministic extractor by Fouque et al. [FPSZ06] does not require any extra randomness to produce the output. However, it extracts substantially less than half of the bits of a uniformly distributed random element of the subgroup. Our extractor does require extra randomness $\mathsf{rand} \in \mathbb{Z}_l$, $l \geq 1$, but one gets this randomness back in the sense that the extractor outputs not only the integer from $\mathbb{Z}_q$ but also an element of $\mathbb{Z}_l$. The crucial advantage of our extractor is that it *extracts all the bits of the subgroup element.*

The new extractor can be used not only for designing pseudorandom generators but also for key exchange protocols to convert the random group element shared by the parties involved to the random binary string.

4.4 Generator PRG_1 versus Gennaro's generator

In this section, we compare the first instance of the DDH generator, i.e., generator PRG_1, with the well-known Gennaro's generator [Gen05] in the setting of concrete security.

Security of Gennaro's generator relies on the intractability of the discrete logarithm with short exponent problem (the DLSE problem) in $\mathbb{Z}_p^*$. This problem is discussed in Section 2.4.3.

Now, we recall the main result of [Gen05].

Let g be a generator of $\mathbb{Z}_p^*$, where p is an n-bit safe prime. Let $x_1 \in_R \mathbb{Z}_{p-1}$ be the seed. Gennaro's generator (Algorithm 4.4.1) parameterized by c and k such that $0 < c < n$, $k > 0$ transforms the seed into a pseudorandom sequence of length $k(n - c - 1)$. For $x \geq 0$ and $j > 0$, let $\ell_j(x) \in \{0, 1\}$ denote the j-th least significant bit of x, so

$$x = \sum_j \ell_j(x) 2^{j-1}.$$

Algorithm 4.4.1 Gennaro's pseudorandom generator

```
Input: $x_1 \in \mathbb{Z}_{p-1}$, integer parameters $c$, $k$
Output: $k(n-c-1)$ pseudorandom bits
  for $i = 1$ to $k$ do
    set $\mathsf{output}_i \leftarrow \ell_2(x_i), \ell_3(x_i), \ldots, \ell_{n-c}(x_i)$
    set $x_{i+1} \leftarrow g^{\sum_{j=n-c+1}^{n} \ell_j(x_i)2^{j-1} + \ell_1(x_i)}$
  end for
  return $\mathsf{output}_1, \mathsf{output}_2, \ldots, \mathsf{output}_k$
```

The following statement is the concrete version of Theorem 2 of [Gen05].

Theorem 4.4.1 (Gennaro) *Suppose Gennaro's pseudorandom generator is not (T, ϵ)-secure. Then there exists an algorithm that solves the c-DLSE in $\mathbb{Z}_p^*$ in expected time* $16c(\ln c)(k/\epsilon)^3 T$.

Note that $k = M/(n-c-1)$, where M denotes the total number of pseudorandom bits output by the generator. The above theorem implies that under the DLSE assumption (Assumption 2.4.6) Gennaro's generator is (T, ϵ)-secure if

$$\frac{16c(\ln c)M^3 T}{\epsilon^3 (n-c-1)^3} < \min[L(n),\ 2^{c/2+1} n^2/24]. \tag{4.3}$$

Let $n = 3000, c = 225, M = 2^{20}$ as suggested by Gennaro [Gen05]. Figure 4.1 shows the values of T and ϵ for which Gennaro's generator is provably (T, ϵ)-secure in accordance with Condition (4.3). Namely, if $\log_2 T$ and $\log_2(1/\epsilon)$ are inside the black triangle then Gennaro's generator is (T, ϵ)-secure. The figure also shows the values of T and ϵ for which generator PRG_1 with $n = 1800$ is (T, ϵ)-secure. In comparison with Gennaro's generator, PRG_1 is secure for a wider range of parameters T and ϵ.

Next, we compare the computational efforts of these two generators.

Gennaro's generator outputs $(n-c-1)$ bits per modular exponentiation with c-bit exponent. The standard right-to-left exponentiation costs on average $c/2$ multiplications and c squarings. Assume that a squaring modulo p takes about 80% of the time of a multiplication modulo p (cf. [LV00]). Then, the average computational effort is $1.3cn^2/(24(n-c-1))$ time units per output bit. For $n = 3000$, $c = 225$, we get about 40000 time units per output bit.

Generator PRG_1 outputs n bits at the cost of 2 modular exponentiations with n-bit exponent. The average computational effort for $n = 1800$ is $2.6n^2/24 \approx 290000$.

So, although the DDH generator provides provable security for a wider range of parameters T and ϵ, it is about 7 times less efficient in terms of computational effort. There is a tradeoff between security and efficiency.

Finally, note that the DLSE problem has not been studied as extensively as the DDH problem which gives more credibility to our construction.

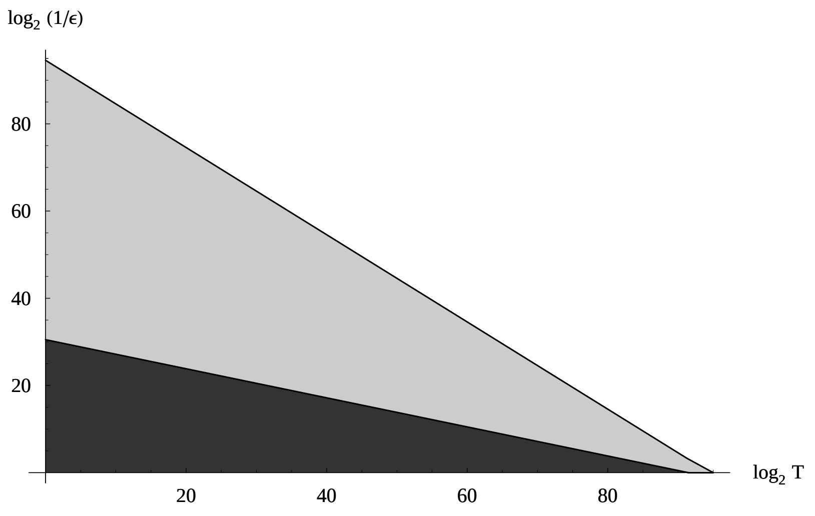

Figure 4.1: If the values of $\log_2 T$ and $\log_2(1/\epsilon)$ are inside the black triangle then Gennaro's generator with parameters $n = 3000$, $c = 225$, $M = 2^{20}$ is (T, ϵ)-secure. Generator PRG_1 with $n = 1800$ is secure not only for $\log_2 T$ and $\log_2(1/\epsilon)$ from the black triangle but also for the ones from the gray area.

4.5 Converting random numbers to random bits

In this section, we consider the algorithm proposed by Juels et al. [JJSH00] that transforms uniformly random numbers into uniformly random bits. This algorithm can be used as a building block for the DDH generator and also has some other important applications (e.g., key exchange protocols).

In the original paper [JJSH00], the authors analyze the expected number of bits one can extract from a single random integer. They show that on average at least $\lceil \log_2(b+1) \rceil - 2$ random bits can be extracted from a random integer uniformly distributed over an interval of length b.

We provide a more detailed analysis of the algorithm. We compute not only the average number of extracted bits but also its probability distribution and the variance.

4.5.1 The algorithm of Juels et al.

Let m be a random integer uniformly distributed over an interval $\{0, 1, \ldots, b-1\}$, $2^{n-1} \leq b < 2^n$, $n > 0$. The algorithm proposed by Juels et al. traverses m bit by bit starting at the most significant bit $\ell_n(m)$. Whenever $\ell_i(m) < \ell_i(b)$, $0 \leq i < n$, the extractor outputs the $i - 1$ least significant bits of m.

More formally, the algorithm is as follows.

Algorithm 4.5.1 The algorithm that converts random numbers to random bits

```
Input: random integer $m \in_R \mathbb{Z}_b$
Output: $i$ uniformly random bits, $0 \leq i < n$
  $i \leftarrow n$
  while $\ell_i(m) = \ell_i(b)$ do
    $i \leftarrow i - 1$
  end while
  return $\ell_1(m), \ldots, \ell_{i-1}(m)$
```

The algorithm can extract up to $n-1$ random bits. Under the condition that $\ell_i(m) < \ell_i(b)$, the i least significant bits of m are distributed uniformly.

4.5.2 Analysis of the algorithm

Let Y be a random variable that denotes the number of random bits extracted from an integer m chosen uniformly at random from an interval $\{0, 1, \ldots, b-1\}$, $2^{n-1} \leq b < 2^n$. We determine the probability distribution, the expected value, and the variance of the random variable Y.

The expression for the expected value of Y is already known from [JJSH00]. We reproduce this analysis for the sake of completeness and also provide more details.

Probability distribution and expected value

Theorem 4.5.1 *The probability distribution of random variable Y that represents the number of random bits extracted by Algorithm* 4.5.1 *from an integer $m \in_R \mathbb{Z}_b$ with $2^{n-1} \leq b < 2^n$ is as follows.*

$$\Pr[Y = i] = \frac{2^i \ell_{i+1}(b)}{b}, 0 \leq i < n.$$

Proof: Let g_i, $0 \leq i < n$, denote the event that the $n-i-1$ most significant bits of m are the same as the $n-i-1$ most significant bits of b, $\ell_{i+1}(m) = 0$ and $\ell_{i+1}(b) = 1$. Algorithm 4.5.1 extracts i bits if and only if event g_i happens. Then, for $0 \leq i < n$,

$$\Pr[Y = i] = \Pr(g_i) = \frac{2^i \ell_{i+1}(b)}{b}. \tag{4.4}$$

This completes the proof. □

The above result implies that the expected value of Y satisfies

$$\mathsf{E}(Y) = \sum_{i=1}^{n-1} \frac{2^i \ell_{i+1}(b) i}{b}. \tag{4.5}$$

Next, we show that $b = 2^n - 1$ is the worst case in the sense that the average number of extracted bits is minimal for this value of b.

Lemma 4.5.2 *Let $2^{n-1} \leq b < 2^n$, and suppose that $\ell_{i+1}(b) = 1$ for some i, $0 \leq i < n-1$. Put $\tilde{b} = b - 2^i$. Let $\mathsf{E}(Y)$ (resp., $\mathsf{E}(\tilde{Y})$) be the expected value of Y (resp., $\tilde{Y}$) corresponding to the upper bound b (resp., $\tilde{b}$). Then $\mathsf{E}(Y) < \mathsf{E}(\tilde{Y})$.*

Proof: Since the expected value satisfies (4.5), we have to prove that

$$\frac{\sum_{j=0}^{n-1} 2^j \ell_{j+1}(b) j}{\sum_{k=0}^{n-1} 2^k \ell_{k+1}(b)} < \frac{\sum_{j=0}^{n-1} 2^j \ell_{j+1}(b) j - 2^i i}{\sum_{k=0}^{n-1} 2^k \ell_{k+1}(b) - 2^i},$$

which is equivalent to

$$\sum_{j=0}^{n-1} 2^j \ell_{j+1}(b) j > i \sum_{j=0}^{n-1} 2^j \ell_{j+1}(b).$$

Since $i \leq n-2$, the result follows from

$$\begin{aligned} \sum_{j=0}^{n-1} 2^j \ell_{j+1}(b)(j-n+2) &= 2^{n-1} - \sum_{j=0}^{n-2} 2^j \ell_{j+1}(b)(n-j-2) \\ \geq 2^{n-1} - \sum_{j=0}^{n-2} 2^j (n-j-2) &= \sum_{j=0}^{n-1} 2^j (j-n+2) = n > 0, \end{aligned}$$

using that $\ell_n(b) = 1$. □

Let $b = 2^n - 1$. Then Equation (4.5) and Lemma 4.5.2 implies that

$$\mathsf{E}_{\min}(Y) = \sum_{i=1}^{n-1} \frac{2^i}{2^n - 1} i = n - 2 + \frac{n}{2^n - 1}. \tag{4.6}$$

Corollary 4.5.3 *The expected value $\mathsf{E}(Y)$ of the number of random bits extracted by Algorithm 4.5.1 from an integer $m \in_R \mathbb{Z}_b$ is more than $n-2$ for all n and b such that $2^{n-1} \leq b < 2^n$. If $b = 2^n - 1$, then $\mathsf{E}(Y) \to n-2$ as $n \to \infty$.*

Variance

Now, we determine the variance of random variable Y that represents the number of random bits extracted from an integer m chosen uniformly at random from an interval $\{0, 1, \ldots b-1\}$, $2^{n-1} \leq b < 2^n$.

Theorem 4.5.4 *The variance $\mathsf{Var}(Y)$ of the number of random bits extracted by Algorithm 4.5.1 from an integer $m \in_R \mathbb{Z}_b$, $2^{n-1} \leq b < 2^n$, satisfies the following equation:*

$$\mathsf{Var}(Y) = \sum_{i=0}^{n-1} \frac{2^i \ell_{i+1}(b)(i - \mathsf{E}(Y))^2}{b}.$$

Proof: By definition, $\mathsf{Var}(Y) = \mathsf{E}((Y-\mu)^2)$, where μ denotes $\mathsf{E}(Y)$. Recall that g_i, $0 \le i < n$, denotes the event that the $n-i-1$ most significant bits of m are the same as the $n-i-1$ most significant bits of b, $\ell_{i+1}(m) = 0$ and $\ell_{i+1}(b) = 1$. Then,

$$\mathsf{Var}(Y) = \sum_{i=0}^{n} \mathsf{E}((Y-\mu)^2 | g_i) \Pr(g_i),$$

Note that $\mathsf{E}((Y-\mu)^2|g_i) = (i-\mu)^2$. It follows from (4.4) that

$$\mathsf{Var}(Y) = \sum_{i=0}^{n-1} \frac{2^i \ell_{i+1}(b)(i-\mu)^2}{b}.$$

□

4.6 Conclusion

Independent of our work, Jiang [Jia06] recently proposed a pseudorandom generator which is also provably secure under the DDH assumption. The security properties of Jiang's generator are similar to ours (hence his generator compares similarly to Gennaro's generator). In comparison to Jiang's generator, our construction has two major advantages. Firstly, Jiang's generator can be based only on the group of quadratic residues modulo a safe prime while our construction extends to many other groups of prime order. Secondly, the seed of our generator PRG_1 is half as long as the seed of Jiang's generator.

The seed length is a critical issue for pseudorandom generators. For instance, if a pseudorandom generator is used as a keystream generator for a stream cipher the seed length corresponds to the length of the secret key. Also, from a theoretical point of view, the seed length is perhaps the most important parameter of a pseudorandom generator, as discussed in detail in a recent paper by Haitner et al. [HHR06].

Finally, we note that constructing an efficient provably secure pseudorandom generator which relies on the intractability of the DDH problem on an ordinary elliptic curve is an interesting open problem. For most ordinary elliptic curves, the best known methods for solving the elliptic curve discrete logarithm problem are the exponential square root attacks, so to reach a security level of 2^{88} time units it suffices to let the size of the group be about 2^{176}. Hence, such an elliptic curve based generator would allow for a considerable reduction of the seed length, potentially to a seed of 176 bits only.

To implement the DDH generator based on an elliptic curve, one has to construct an efficiently computable function that bijectively maps the points of the curve to $\mathbb{Z}_q$, where q is the order of the group. This function seems to be difficult to construct for ordinary elliptic curves. For some supersingular elliptic curves, the function can be constructed (see, e.g., [Kal88]). However, the latter curves cannot be used for the DDH generator since the DDH problem in these curves can be easily solved by computing Weil pairings [MOV93; JN03].

Recall that the updated version of the DEC generator considered at the beginning of Section 4.1.2 is secure not only under the DDH assumption but also under the non-standard assumption that the x-logarithm is intractable. The latter assumption seems to be difficult to get rid of.

Chapter 5

Concrete security of the RSA generator

No one can duplicate the
confidence that RSA offers after
20 years of cryptanalytic review.

Bruce Schneier

In this chapter, a new security proof for the RSA generator is proposed. Our reduction algorithm relies on the techniques of Fischlin and Schnorr [FS00], as well as ideas of Vazirani and Vazirani [VV84]. We combine these approaches in a novel way for the case that the generator outputs more than one bit per application of the RSA function. The new reduction is more efficient than all previously known reductions. We show how to select both the size of the modulus and the number of bits extracted on each iteration such that a desired level of provable security is reached, while minimizing the computational effort per output bit.

This chapter is based on the joint work with B. Schoenmakers [SS05].

5.1 Introduction

The RSA problem is the problem of decrypting a ciphertext in the context of the RSA cryptosystem [RSA78] without any information about the private key (this problem is stated formally in Section 2.3.2). The RSA generator is a provably secure pseudorandom generator based on the intractability of the RSA problem. This is one of the classical pseudorandom generators.

Before describing the RSA generator, we introduce some notation. Let N be an RSA modulus, i.e., let N be a product of two primes and let $e > 1$ be coprime with $\phi(N)$, where ϕ denotes Euler's totient function. We denote by $[x]_N$ the smallest nonnegative residue of x modulo N, that is, $[x]_N = x \bmod N$, $[x]_N \in \{0, 1, \ldots, N-1\}$. As before, $\ell_j(x)$ denotes the j-th least significant bit of x and $L_j(y) =$

$y \bmod 2^j$, for any $x \geq 0$, $j > 0$, and $y \in \mathbb{Z}$.

Let $x_0 \in_R \mathbb{Z}_N$ be a seed. The RSA generator (Algorithm 5.1.1) transforms the seed into the pseudorandom sequence of length $M > 0$ by iterating the RSA encryption function $E_N(x) = [x^e]_N$.

Algorithm 5.1.1 RSA generator with j output bits per iteration

```
Input: Modulus $N$, seed $x_0 \in_R \mathbb{Z}_N$, parameters $e,\ j,\ k$
Output: Pseudorandom sequence of length $M = kj$
  for $i = 1$ to $k$ do
    $x_i \leftarrow [x_{i-1}]_N$
    return $L_j(x_i)$
  end for
```

The first result concerning the security of the RSA generator is published by Goldwasser et al. [GMT82]. They show that a polynomial-time algorithm that recovers ($x \bmod 2$) from $E_N(x)$ can be transformed into a polynomial-time solver for the RSA problem. Note that Goldwasser et al. consider only algorithms that recover ($x \bmod 2$) from $E_N(x)$ with probability 1 so their result does *not* immediately imply that the RSA generator with one output bit per iteration ($j = 1$) is asymptotically secure.

The result of Goldwasser et al. is improved in a number of ways. Ben-Or et al. [BOCS83, Section 3] show that no polynomial-time algorithm can recover ($x \bmod 2$) from $E_N(x)$ with probability more than $3/4 + 1/f(\log N)$, for any polynomial f, unless there exists a polynomial-time solver for the RSA problem. In [BOCS83, Section 4] this result is generalized for almost any individual RSA message bit. Finally, Alexi et al. [ACGS88] prove that ($x \bmod 2$) cannot be recovered from $E_N(x)$ with probability more than $1/2 + 1/f(\log N)$, which implies that the RSA generator with one output bit per iteration is indeed asymptotically secure. Besides that, Alexi et al. consider a problem of simultaneous security of the RSA message bits and prove that the $O(\log \log N)$ least significant bits are secure. The latter result implies that the RSA generator is asymptotically secure for $j = O(\log \log N)$.

Vazirani and Vazirani [VV84] prove that the Blum-Blum-Shub generator [BBS86] based on the Rabin function $R_N(x) = [x^2]_N$, where $N = pq$, $p \equiv q \equiv 3 \bmod 4$, is also asymptotically secure for $j = O(\log \log N)$.

A disadvantage of the above results is that they establish asymptotic security of the pseudorandom generators rather than concrete security. As discussed in Section 1.5, asymptotic security implies that no polynomial time adversary can break the pseudorandom generator while, in practice, it is important to protect cryptographic systems against adversaries investing a specific amount of computational effort.

The reductions [GMT82; BOCS83; VV84; ACGS88] are polynomial-time but they are not tight. Therefore, according to these reductions the seed length of the RSA generator that corresponds to a reasonable level of security is enormously large. For instance, the reduction of Alexi et al. [ACGS88] implies that to protect against adversaries with running time 2^{120} and advantage 0.01 one has to use a 50000-bit modulus N (for $j = 6$, $M = 2^{20}$). Of course, one exponentiation modulo a 50000-bit number is a high price for just one output bit. Moreover, the results [BOCS83; VV84; ACGS88] do not explain how to choose the number of bits j to be output on each iteration, that is, the constant in front of $\log \log N$ is unclear. For instance, in [YY04; ECS] it is recommended to use the Blum-Blum-Shub generator with $\log_2 \log_2 N$ output bits per iteration. Koblitz and Menezes [KM06] show that this choice is not backed up by the security proof.

In this chapter, we solve the following problems. We tighten the security proof for the RSA generator in case $j > 1$ and we explain how to select both the size of the modulus and the number of bits extracted on each iteration such that a desired level of security is reached, while minimizing the computational effort per output bit.

The new reduction is more efficient than all previously known reductions. It shows that the RSA generator (for $j = 6$, $M = 2^{20}$) is provably secure against adversaries with running time 2^{120} and advantage 0.01 for 13000-bit modulus, which is still large but significantly less than 50000.

Our work is based on the approach of Fischlin and Schnorr [FS00] who propose a strong reduction for the RSA generator with one output bit per iteration. Using an idea of Vazirani and Vazirani (i.e., the computational XOR proposition) we extend the argument of [FS00] to the case in which more than one bit is extracted on each iteration. A similar generalization is suggested by Fischlin and Schnorr [FS00, Section 2] however they do not provide any sufficient detail.

Throughout this chapter, n denotes the bit length of the modulus N; we assume that $n > 2^9$.

5.2 The starting point of the security analysis

Our ultimate goal of the security analysis of the RSA generator is to transform an adversary that distinguishes sequences produced by the generator from uniformly random sequences into a solver for the RSA problem.

The starting point of the analysis is Lemma 5.2.1, which shows that if the RSA generator is not secure then one can construct an algorithm O_{xor} with certain properties. In Section 5.4, we show that algorithm O_{xor} can be used as an oracle for inversion of the RSA one-way function. The idea to use oracle O_{xor} for the security analysis of the RSA generator originates from [VV84].

Lemma 5.2.1 uses the following notation. Let $\pi \subseteq \{1, 2, \dots, j\}$ be a nonempty set. For an integer y, $y \geq 0$, let

$$\pi(y) = \sum_{i \in \pi} \ell_i(y) \bmod 2.$$

Note that the set and the corresponding XOR-function are denoted by the same symbol π.

Lemma 5.2.1 *Let $x \in_R \mathbb{Z}_N$ and let π be a random nonempty subset of the set $\{1, 2, \dots, j\}$ with uniform probability distribution. Suppose the RSA generator is not (T, ϵ)-secure. Then there exists an algorithm O_{xor} that on input $(\pi, E_N(x))$ guesses $\pi(x)$ with probability at least $1/2 + \delta$, where $\delta = (2^j - 1)^{-1}(\epsilon/M)$. Here, the probability is taken over choices of x and π and also over internal coin flips of O_{xor}. The running time $T_{O_{xor}}$ of O_{xor} satisfies $T_{O_{xor}} \le T + Mn^2/24$.*

Proof: The proof of the lemma uses the hybrid technique (see, e.g., [Gol01, Section 3.2.3]) and the computational XOR-proposition [VV84; Gol95].

In the first part of the proof, we construct an algorithm O_{dist} that on input $E_N(x)$ distinguishes $L_j(x)$ from $r \in_R \{0,1\}^j$ with advantage ϵ/k, i.e.,

$$|\Pr[O_{dist}(E_N(x), L_j(x)) = 1] - \Pr[O_{dist}(E_N(x), r) = 1]| \ge \epsilon/k,$$

where k is the number of iterations done by Algorithm 5.1.1. In the second part of the proof, we transform O_{dist} into O_{xor}.

Let $x_1, x_2, \dots, x_k$ be the sequence of numbers computed by the RSA generator with seed x, that is, $x_1 = E_N(x)$, $x_2 = E_N(x_1)$, ..., $x_k = E_N(x_{k-1})$. The output of the RSA generator is a random variable

$$Z_0 = (L_j(x_1), L_j(x_2), \dots, L_j(x_k)).$$

For $i = 1, 2, \dots, k$, consider hybrid random variables

$$Z_i = (r_1, r_2, \dots, r_i, L_j(x_1), L_j(x_2), \dots, L_j(x_{k-i})),$$

where $r_i \in_R \{0,1\}^j$.

Suppose there exists a T-time algorithm $\mathcal{D}$ that distinguishes the output of the RSA generator from a uniformly random sequence with advantage ϵ, that is,

$$|\Pr[\mathcal{D}(Z_0) = 1] - \Pr[\mathcal{D}(Z_k) = 1]| \ge \epsilon.$$

Let $s \in_R \{0, 1, \dots, k-1\}$. Following the hybrid technique, one can prove that

$$|\Pr[\mathcal{D}(Z_s) = 1] - \Pr[\mathcal{D}(Z_{s+1}) = 1]| \ge \epsilon/M.$$

Now, we construct algorithm O_{dist}. Let $(E_N(x), z)$, where either $z = L_j(x)$ or $z = r$, be the input. Algorithm O_{dist} distinguishes the two possibilities as follows.

```
let $s \in_R \{0, 1, \dots, k-1\}$
**for** $i = 1$ to $s$ **do**
  select $r_i \in_R \{0,1\}^j$
**end for**
compute $x_1 = E_N(x)$, $x_2 = E_N(x_1)$, ..., $x_{k-s-1} = E_N(x_{k-s-2})$
assign $Z \leftarrow (r_1, r_2, \dots, r_s, z, L_j(x_1), L_j(x_2), \dots, L_j(x_{k-s-1}))$
**return** $\mathcal{D}(Z)$
```

If $z = L_j(x)$ then Z is distributed as Z_s. In the opposite case, if $z = r$ then Z distributed as Z_{s+1}. Therefore,

$$|\Pr[O_{dist}(E_N(x), L_j(x)) = 1] - \Pr[O_{dist}(E_N(x), r) = 1]| \geq \epsilon/k,$$

which concludes the first part of the proof.

Without loss of generality, assume that

$$\Pr[O_{dist}(E_N(x), L_j(x)) = 1] - \Pr[O_{dist}(E_N(x), r) = 1] \geq \epsilon/k.$$

Now, we define algorithm O_{xor} in the same way as algorithm G is defined in [Gol95, Section 2.1.1].

```
let $\{r_1, r_2, \ldots, r_j\} \in_R \{0,1\}^j$
assign $r \leftarrow \sum_{i=1}^{j} r_i 2^{i-1}$
**if** $O_{dist}(E_N(x), r_1, r_2, \ldots, r_j) = 1$ **then**
  **return** $\pi(r)$
**else**
  **return** $1 - \pi(r)$
**end if**
```

The computational XOR-proposition [Gol95] implies that O_{xor} guesses $\pi(x)$ with probability at least $1/2 + \delta$, where $\delta = (2^j - 1)^{-1}(\epsilon/M)$. □

5.3 The simplified inversion algorithm

The key ingredient of the security analysis of the RSA generator, is a proof that the RSA one-way function E_N, can be inverted given an oracle of a particular type. In this section, we consider the related problem of inverting E_N given a more powerful oracle O_{lsb}, see below. The treatment of this case serves as a stepping stone to the general case.

5.3.1 Majority decision

Let O_{lsb} be a probabilistic algorithm that *for all* $\alpha \in \mathbb{Z}_N$, given $E_N(\alpha)$, guesses bit $\ell_1(\alpha)$ with advantage $\delta > 0$. Algorithm O_{lsb} is more powerful than O_{xor} in the sense that its advantage is the same for all inputs.

In order to determine the least significant bit of some number, one has to run O_{lsb} several times and then use the majority decision. The following lemma shows that running the oracle a certain number of times yields the desired bit with probability close to 1.

Lemma 5.3.1 *Let $\alpha \in \mathbb{Z}_N$ and let $X_1, X_2, \ldots, X_m$ be the outputs of O_{lsb} on input $E_N(\alpha)$. Let X be the majority decision, that is,*

$$X = \left[\frac{1}{m}\sum_{i=1}^{m} X_i > \frac{1}{2}\right].$$

Then, for $m = \frac{1}{2}\ln(n/\gamma)\delta^{-2}$, $0 < \gamma < 1$, *the probability that* $X = \ell_1(\alpha)$ *is at least* $1 - \gamma/n$. *The probability is taken over internal coin flips of* O_{lsb}.

Proof: Without loss of generality, assume that $\ell_1(\alpha) = 0$. Then the majority decision errs if

$$\frac{1}{m}\sum_{i=1}^{m} X_i > \frac{1}{2}. \tag{5.1}$$

Since for each $\alpha \in \mathbb{Z}_N$ the probability that O_{lsb} successfully guesses $\ell_1(\alpha)$ equals $\frac{1}{2} + \delta$, the expected value $\mathsf{E}[X_i]$ is $\mathsf{E}[X_i] = \frac{1}{2} - \delta$, $i = 1, 2, \ldots, m$. Inequality (5.1) implies that

$$\frac{1}{m}\sum_{i=1}^{m} X_i - \mathsf{E}[X_i] > \delta.$$

Note that $X_1, X_2, \ldots, X_m$ are mutually independent so Hoeffding's bound [Hoe63] gives

$$\Pr\left[\frac{1}{m}\sum_{i=1}^{m} X_i - \mathsf{E}[X_i] > \delta\right] \le \exp\left(-2m\delta^2\right).$$

This implies that for $m = \frac{1}{2}\ln(n/\gamma)\delta^{-2}$ the majority decision errs with probability at most γ/n. □

Majority decision is used as a tool for constructing reductions for pseudorandom generators in many papers, e.g., in [ACGS88; VV84; FS00].

5.3.2 Binary division

The main tool of the inversion algorithm is the *binary division* technique [GMT82; FS00], which is a means to solve the following problem: recover a value $\alpha \in \mathbb{Z}_N$, given $\ell_1(\alpha)$, $\ell_1([2^{-1}\alpha]_N)$, $\ldots$, $\ell_1([2^{-(n-1)}\alpha]_N)$.

The solution to this problem is given in terms of rational approximations. For a rational number u_0, $0 \le u_0 < 1$, we call βN a *rational approximation* of integer $\alpha \in \mathbb{Z}_N$, with *error* $|\alpha - u_0 N|$. Given a rational approximation $u_0 N$ for α we can get a rational approximation $u_1 N$ for $\alpha_1 = [2^{-1}\alpha]_N$ for which the error is reduced by a factor of 2 as follows. If α is even, then $\alpha_1 = \alpha/2$ so put $u_1 = u_0/2$; otherwise, $\alpha_1 = (\alpha + N)/2$ so put $u_1 = (u_0 + 1)/2$. Then we have $|\alpha_1 - u_1 N| = \frac{1}{2}|\alpha - u_0 N|$. Note that to determine u_1, the only required information on α is its parity.

Given $\ell_1(\alpha), \ell_1([2^{-1}\alpha]_N), \ldots, \ell_1([2^{-(n-1)}\alpha]_N)$, the value of α can be recovered as follows. Put $u_0 = 1/2$, then $u_0 N$ is a rational approximation of α with error at most $N/2$. Next, we apply the above technique n times to obtain rational approximations $\beta_1 N, \ldots, \beta_n N$ for $[2^{-1}\alpha]_N, \ldots, [2^{-n}\alpha]_N$ respectively, at each step reducing the error by a factor of 2. We thus get a rational approximation $u_n N$ to $[2^{-n}\alpha]_N$, for which the error is less than $N/2^{n+1} < 1/2$. The closest integer to $u_n N$ is therefore equal to $[2^{-n}\alpha]_N$, and from this value we find α.

5.3.3 The simplified algorithm

Algorithm 5.3.1 inverts the RSA one-way function E_N as follows. It picks a random $a \in_R \mathbb{Z}_N$ and approximates $[ax]_N$ by $N/2$ (which means $u_0 = 1/2$). Next, the algorithm determines the parity of $[ax]_N$ via O_{lsb} and computes the rational approximation $u_1 N$ for $[a_1 x]_N$ (we denote $a_t = [2^{-t}a]_N$, $t = 1, 2, \ldots, n$) such that $|[a_1 x]_N - u_1 N| < \frac{1}{2}|[ax]_N - u_0 N|$, as described in Section 5.3.2. After that, the algorithm determines the parity of $[a_1 x]_N$, computes the rational approximation $u_2 N$ for $[a_2 x]_N$, and so on. Finally, a precise rational approximation $u_n N$ for $[a_n x]_N$ is computed and x is recovered.

Algorithm 5.3.1 The simplified inversion algorithm for the RSA one-way function (uses oracle O_{lsb})

```
Input: Modulus $N$, $E_N(x)$ for $x \in \mathbb{Z}_N$, parameter $\gamma$ such that $0 < \gamma < 1$
Output: $x$
  $m \leftarrow \frac{1}{2}\ln(n/\gamma)\delta^{-2}$
  $u_0 \leftarrow 1/2$
  repeat
    $a \in_R \mathbb{Z}_N$
    for $t = 0$ to $n - 1$ do
      $a_t \leftarrow [2^{-t}a]_N$
      compute $E_N([a_t x]_N) = E_N(a_t)E_N(x) \bmod N$
      run $O_{lsb}$ on input $E_N([a_t x]_N)$ $m$ times
      $l_t \leftarrow$ majority output bit
      $a_{t+1} \leftarrow [2^{-1}a_t]_N$
      $u_{t+1} \leftarrow (u_t + l_t)/2$
    end for
    $x' \leftarrow a_n^{-1}\lfloor u_n N + 1/2 \rfloor \bmod N$
  until $E_N(x') \neq E_N(x)$
  return $x'$
```

Each parity bit $\ell_1([a_t x]_N)$, $t = 0, 1, \ldots, n-1$ is determined by majority decision. If at some step $t \in \{0, 1, \ldots, n-1\}$ the majority decision is not correct, the algorithm finds it out only at the last step by comparing $E_N(x')$ with $E_N(x)$. Therefore, an "early abort" in the case that one of the majority decisions is incorrect is not possible. If all the majority decisions are correct, we get $l_t = \ell_1([a_t x]_N)$. Otherwise, the algorithm restarts with a different multiplier $a \in_R \mathbb{Z}_N$.

It follows from Lemma 5.3.1 that a single majority decision is correct with probability at least $1 - \gamma/n$. Thus, the probability that $E_N(x') = E_N(x)$ for each random multiplier a is at least $1 - \gamma$. Therefore, the expected running time of the simplified inversion algorithm is at most $\frac{1}{2}(1-\gamma)^{-1} n \ln(n/\gamma)\delta^{-2} T_{O_{lsb}}$, where $T_{O_{lsb}}$ is the running time of O_{lsb}. For instance, for $\gamma = 1/2$ the running time is essentially $n(\ln n)\delta^{-2} T_{O_{lsb}}$.

5.4 The inversion algorithm

Now, we construct an algorithm that retrieves $x \in \mathbb{Z}_N$ from $E_N(x)$ using algorithm O_{xor} as an oracle. Recall that O_{xor} on input $(\pi, E_N(x))$, $\pi \subseteq \{0, 1, \ldots, j\}$, guesses $\pi(x)$ with probability $1/2 + \delta$, where $\delta = (2^j - 1)^{-1}(\epsilon/M)$.

The inversion algorithm described in this section is based on the same ideas as Algorithm 5.3.1. The main difference between these two algorithms is that, in comparison with O_{lsb}, the advantage of O_{xor} is, in general, not the same for all input values. To determine $\ell_1([a_t x]_N)$, $t \in \{1, 2, \ldots, n-1\}$ via O_{xor}, it does *not* suffice to run the oracle on the same input. Instead, the input values have to be "randomized".

Tightening the rational approximations.

Suppose $E_N(x)$ is given for $x \in \mathbb{Z}_N$. The goal is to recover x using O_{xor} as an oracle.

Let $a, b \in_R \mathbb{Z}_N$ (a and b are used to randomize the inputs of O_{xor}). Let $u_0 N$ and vN be rational approximations of $[ax]_N$ and $[bx]_N$. To invert the RSA one-way function, we search through a certain set of quadruples $(u_0 N, vN, l_{a,0}, l_b)$, where $l_{a,0}, l_b \in \{0, 1\}$, so that for at least one of them

$$\begin{aligned} &l_{a,0} = \ell_1([ax]_N),\ l_b = \ell_1([bx]_N), \\ &|[ax]_N - u_0 N| \leq \eta_a N,\ |[bx]_N - vN| \leq \eta_b N, \end{aligned} \tag{5.2}$$

where $\eta_a = 2^{-j-4}\delta^3, \eta_b = 2^{-j-4}\delta$. Condition (5.2) implies that we have to try at most $1/(\eta_a \eta_b)$ quadruples. For the rest of the section, we assume that this condition holds.

The basic principle of the inversion algorithm is as follows. Let $a_t = [2^{-t}a]_N$, $t = 1, 2, \ldots, n$. Provided that the bits $\ell_1([a_t x]_N)$, $t = 0, \ldots, n-1$, are determined, the binary division technique yields rational approximations $u_t N$ for $[a_t x]_N$ such that

$$|[a_t x]_N - u_t N| \leq \frac{\eta_a N}{2^t}, t = 1, 2, \ldots, n.$$

Therefore, $|[a_n x]_N - u_n N| < 1/2$, i.e. the closest integer to $u_n N$ is $[a_n x]_N$ so $x = [a_n^{-1} \lfloor u_n N + \frac{1}{2} \rfloor]_N$.

Below we show how to determine the bits $\ell_1([a_t x]_N)$, $t = 1, 2, \ldots, n-1$.

How to determine $\ell_1([a_t x]_N)$?

Consider step t of the inversion algorithm for $1 \leq t < n$. At this step the rational approximation $u_t N$ for $[a_t x]_N$ is known. The goal is to determine bit $\ell_1([a_t x]_N)$.

Let i be an integer from a multiset $\sigma_t \subset \{-\lceil 8n\delta^{-2} \rceil, -\lceil 8n\delta^{-2} \rceil + 1, \ldots, \lceil 8n\delta^{-2} \rceil\}$. (we define the multisets later in this section). The idea is to measure bit $\ell_1([a_t x]_N)$ a certain number of times (for all $i \in \sigma_t$) so that each measurement is correct with probability slightly higher than 1/2. After that, the bit is determined by majority decision. The details follow.

Let $c_{t,i} = a_t(1 + 2i) + b$. Then

$$[c_{t,i}x]_N = ([a_t x]_N(1 + 2i) + [bx]_N) \bmod N.$$

Let $w_{t,i} = u_t(1 + 2i) + v,\ \tilde{w}_{t,i} = w_{t,i} - \lfloor w_{t,i} \rfloor$. Clearly, $\tilde{w}_{t,i}N$ is an approximation of $[c_{t,i}x]_N$, while $w_{t,i}N$ is an approximation of $[a_t x]_N(1 + 2i) + [bx]_N$. If the error of the rational approximation $w_{t,i}N$ is small enough then

$$[2^j c_{t,i}x]_N = 2^j ([a_t x]_N(1 + 2i) + [bx]_N) - \lfloor 2^j w_{t,i} \rfloor N. \tag{5.3}$$

The latter condition is crucial for the rest of the analysis. It turns out that if Condition (5.3) holds for some $i \in \sigma_t$ then the i-th measurement of $\ell_1([a_t x]_N)$ is correct with probability $1/2+\delta$ (this probability cannot be higher since O_{xor} guesses correctly with probability $1/2 + \delta$). In Section 7.4, we analyze the probability that this condition holds. For the rest of this section we assume that it does hold.

The following lemma provides a way to determine $\ell_1([a_t x]_N)$.

Lemma 5.4.1 *If Condition* (5.3) *holds then for every nonempty subset* $\pi_{t,i} \subseteq \{1, 2, \dots, j\}$ *and* $r = \max\{k \mid k \in \pi_{t,i}\}$*, we have*

$$\begin{aligned}\ell_1([a_t x]_N) = \pi_{t,i}([2^{r-1}c_{t,i}x]_N) + \pi_{t,i}(L_r(-\lfloor 2^{r-1}\tilde{w}_{t,i}\rfloor N)) +\\ \ell_1([bx]_N) + \lfloor w_{t,i} \rfloor \bmod 2.\end{aligned} \tag{5.4}$$

The value of r used in Lemma 5.4.1 actually depends on t and i. We write r instead of $r_{t,i}$ to avoid cumbersome notation.

Lemma 5.4.1 is proved in Section 5.4.2.

Now, suppose that $\pi_{t,i}$ is a *random* nonempty subset of $\{1, 2, \dots, j\}$ with uniform probability distribution. Then, in Eq. (5.4), the component $\pi_{t,i}([2^{r-1}c_{t,i}x]_N)$ can be replaced by the oracle reply $O_{xor}(\pi_{t,i},\ E_N([2^{r-1}c_{t,i}x]_N))$. Recall, however, that the oracle gives the correct output only with probability slightly higher than 1/2.

The majority decision on bit $\ell_1([a_t x]_N)$ works as follows. If for the majority of indices $i \in \sigma_t$

$$\begin{aligned}O_{xor}(\pi_{t,i}, E_N([2^{r-1}c_{t,i}x]_N)) \equiv \pi_{t,i}(L_r(-\lfloor 2^{r-1}\tilde{w}_{t,i}\rfloor N)) +\\ \ell_1([bx]_N) + \lfloor w_{t,i} \rfloor \bmod 2,\end{aligned} \tag{5.5}$$

decide that $\ell_1([a_t x]_N) = 0$, otherwise decide that $\ell_1([a_t x]_N) = 1$.

Multisets σ_t.

Now, we formally define the multisets σ_t, $t = 1, 2, \dots, n - 1$, mentioned above.

For $t < \log_2 n + 4$, denote $m_t = 2^t\delta^{-2}$. Let

$$\sigma_t = \{i \mid |1 + 2i| < m_t\},\ t = 1, 2, \dots, \log_2 n + 3.$$

As t grows we choose a larger value for m_t so more measurements of the least significant bit $\ell_1([a_t x]_N)$ are done. Therefore, the majority decisions become more reliable as t grows. We cannot choose large m_t for small t for the following reason. For small t, the error $|u_t N - [a_t x]_N|$ is large. If m_t is also large then $[a_t x]_N(1 + 2i) + [bx]_N$ can differ much from $w_{t,i} = u_t(1 + 2i) + v$ so that (5.3) does not hold and (5.5) cannot be used for the majority decision.

However, it is *not* true that our goal is to maximize the number of measurements of $\ell_1([a_t x]_N)$ for all $t = 1, 2, \ldots, n-1$. Although a large number of measurements implies a small error probability of the majority decision, it also means that the oracle O_{xor} is used many times, which in turn affects the total running time of the inversion algorithm. In this sense, the number of the measurements has to meet a certain tradeoff.

Define $\rho = \{i \mid |1+2i| < 16n\delta^{-2}\}$. We randomly select $m = 2\delta^{-2}\log_2 n$ elements $\sigma = \{i_1, i_2, \ldots, i_m\}$ *with repetition* from ρ and let

$$\sigma_t = \sigma,\ m_t = m,\ t = \log_2 n + 4, \ldots, n-1.$$

For $t \geq \log_2 n + 4$, σ_t is chosen to be a random subset of ρ rather than the whole set to reduce the number of runs of O_{xor} without increasing much the error probability of the majority decisions.

Note that for all $t = 1, 2, \ldots, n-1$, σ_t has cardinality m_t.

5.4.1 Formal description of the algorithm

Algorithm 5.4.1 is a formal version of the inversion algorithm described above. On input $(E_N(x),\ N,\ j,\ \delta)$, where $x \in \mathbb{Z}_N$, Algorithm 5.4.1 outputs either x or the failure message denoted by $\perp$. The algorithm has access to the oracle O_{xor} that given $E_N(x)$ and $\pi \subseteq \{1, 2, \ldots, j\}$ guesses $\pi(x)$ with advantage δ.

On step t, the goal of the algorithm is to determine $\ell_1([a_t x]_N)$. This bit is determined via the majority decision. Note that $g_{t,i} = O_{xor}(\pi_{t,i}, E_N([2^{r-1}c_{t,i}x]_N))$ and $e_i = l_b + \pi_{t,i}(L_r(-\lfloor 2^{r-1}\tilde{w}_{t,i}\rfloor N) + \lfloor w_{t,i}\rfloor \bmod 2$ (see also (5.5)). If for a majority of indices $i \in \sigma_t$ we have $g_{t,i} = e_i$ then the majority decision outputs 0, otherwise it outputs 1. If the majority decision is correct then $l_{a,t} = \ell_1([a_t x]_N)$.

5.4.2 Proof of Lemma 5.4.1

Recall that Lemma 5.4.1 states that if Condition (5.3) holds, that is, if

$$[2^j c_{t,i}x]_N = 2^j\left([a_t x]_N(1 + 2i) + [bx]_N\right) - \lfloor 2^j w_{t,i}\rfloor N$$

then for every nonempty subset $\pi_{t,i} \subseteq \{1, 2, \ldots, j\}$ and $r = \max\{k \mid k \in \pi_{t,i}\}$, we have

$$\begin{aligned}\ell_1([a_t x]_N) = {} & \pi_{t,i}([2^{r-1}c_{t,i}x]_N) + \pi_{t,i}(L_r(-\lfloor 2^{r-1}\tilde{w}_{t,i}\rfloor N)) + \\ & \ell_1([bx]_N) + \lfloor w_{t,i}\rfloor \bmod 2.\end{aligned}$$

To show this statement, we first prove two claims.

Algorithm 5.4.1 The inversion algorithm for the RSA one-way function (uses O_{xor} as an oracle)

```
Input: $E_N(x)$ for $x \in \mathbb{Z}_N$, modulus $N$, parameters $j$, $\delta$
Output: $x$ or $\perp$
  $n \leftarrow \lceil \log_2(N+1) \rceil$
  let $a, b \in_R \mathbb{Z}_N$
  {First part: oracle calls}
  for $t = 1$ to $n-1$ do
    $a_t \leftarrow [2^{-t}a]_N$
    for all $i \in \sigma_t$ do
      $c_{t,i} \leftarrow a_t(1+2i) + b$
      select a random nonempty set $\pi_{t,i} \subseteq \{1, 2, \ldots, j\}$
      $r \leftarrow \max\{k \mid k \in \pi_{t,i}\}$
      $g_{t,i} \leftarrow O_{xor}(\pi_{t,i}, E_N([2^{r-1}c_{t,i}x]_N))$
    end for
  end for
  {Second part: tightening the rational approximations}
  rational $\eta_a \leftarrow 2^{-j-4}\delta^3$, $\eta_b \leftarrow 2^{-j-4}\delta$
  for $\tilde{u} = 0$ to $\lfloor 1/(2\eta_a) \rfloor$, $\tilde{v} = 0$ to $\lfloor 1/(2\eta_b) \rfloor$, $l_{a,0} = 0$ to $1$, $l_b = 0$ to $1$ do
    rational $u_0 \leftarrow 2\eta_a\tilde{u}, v \leftarrow 2\eta_b\tilde{v}$
    for $t = 1$ to $n-1$ do
      rational $u_t \leftarrow \frac{1}{2}(l_{a,t-1} + u_{t-1})$
      for all $i \in \sigma_t$ do
        rational $w_{t,i} \leftarrow u_t(1+2i) + v$
        $r \leftarrow \max\{k \mid k \in \pi_{t,i}\}$
        $\tilde{w}_{t,i} \leftarrow w_{t,i} - \lfloor w_{t,i} \rfloor$
        $e_i \leftarrow l_b + \pi_{t,i}(L_r(-\lfloor 2^{r-1}\tilde{w}_{t,i} \rfloor N)) + \lfloor w_{t,i} \rfloor \bmod 2$
      end for
      $l_{a,t} \leftarrow \text{MajorityDecision}(g_{t,*} + e_* \bmod 2)$
    end for
    $x' \leftarrow [a_n^{-1}\lfloor u_n N + \frac{1}{2} \rfloor]_N$
    if $E_N(x') = E_N(x)$ then
      return $x'$
    else
      return $\perp$
    end if
  end for
```

Claim 5.4.2 *Let α be an arbitrary integer. Consider a rational number w such that*

$$[2^j\alpha]_N = 2^j\alpha - \lfloor 2^j w\rfloor N \tag{5.6}$$

for some integer $j \geq 1$. Then

(i) $[\alpha]_N = \alpha - \lfloor w\rfloor N$.

(ii) *For all r, $1 \leq r \leq j$, $[2^{r-1}\alpha]_N = 2^{r-1}[\alpha]_N - \lfloor 2^{r-1}\tilde{w}\rfloor N$, where $\tilde{w} = w \bmod 1$.*

Proof: Denote

$$\mu = \frac{\alpha}{N} \in \mathbb{Q};$$
$$\tilde{\mu} = \mu - \lfloor \mu \rfloor.$$

Note that $\mu N = \alpha$ and $\tilde{\mu}N = [\alpha]_N$. Therefore, $[2^k\alpha]_N = 2^k\alpha - \lfloor 2^k\mu\rfloor N$, $[2^k\alpha]_N = 2^k[\alpha]_N - \lfloor 2^k\tilde{\mu}\rfloor N$ for all $k \geq 0$.

1. Suppose that Statement (i) is false, i.e. $[\alpha]_N \neq \alpha - \lfloor w\rfloor N$. Hence, $\lfloor w\rfloor \neq \lfloor \mu\rfloor$. Therefore $\lfloor 2w\rfloor \neq \lfloor 2\mu\rfloor, \ldots, \lfloor 2^j w\rfloor \neq \lfloor 2^j\mu\rfloor$. We have

$$[2^j\alpha]_N \neq 2^j\alpha - \lfloor 2^j w\rfloor N.$$

This contradicts Condition (5.6) so Statement (i) is valid.

2. Now, suppose that Statement (ii) is false, i.e. there exists r, $1 \leq r \leq j$, such that

$$[2^{r-1}\alpha]_N \neq 2^{r-1}[\alpha]_N - \lfloor 2^{r-1}\tilde{w}\rfloor N.$$

Hence, $\lfloor 2^{r-1}\tilde{\mu}\rfloor \neq \lfloor 2^{r-1}\tilde{w}\rfloor$. It follows that $\lfloor 2^r\tilde{\mu}\rfloor \neq \lfloor 2^r\tilde{w}\rfloor, \ldots, \lfloor 2^j\tilde{\mu}\rfloor \neq \lfloor 2^j\tilde{w}\rfloor$. The j least-significant bits of $\lfloor 2^j\mu\rfloor$ equal $\lfloor 2^j\tilde{\mu}\rfloor$, and the j least-significant bits of $\lfloor 2^j w\rfloor$ equal $\lfloor 2^j\tilde{w}\rfloor$. Hence, $\lfloor 2^j\mu\rfloor \neq \lfloor 2^j w\rfloor$ and

$$[2^j\alpha]_N \neq 2^j\alpha - \lfloor 2^j w\rfloor N.$$

The contradiction proves the Statement (ii).

□

Let $\alpha = [a_t x]_N(1+2i) + [bx]_N$, $w = w_{t,i}$. Then, the Statement (i) of Claim 5.4.2 implies that if Condition (5.3) holds then

$$[c_{t,i}x]_N = [a_t x]_N(1+2i) + [bx]_N - \lfloor w_{t,i}\rfloor N.$$

Since $\ell_1(N) = 1$, we get

$$\ell_1([c_{t,i}x]_N) = \ell_1([a_t x]_N) + \ell_1([bx]_N) + \lfloor w_{t,i}\rfloor \bmod 2. \tag{5.7}$$

Statement (ii) of Claim 5.4.2 is used to prove another claim.

Claim 5.4.3 *If Condition* (5.3) *holds for a pair* (t,i) *such that* $t \in \{1,2,\ldots,n-1\}$, $i \in \sigma_t$ *then for every nonempty subset* $\pi_{t,i} \subseteq \{1,2,\ldots,j\}$ *and* $r = \max\{k \mid k \in \pi_{t,i}\}$, *we have*

$$\ell_1([c_{t,i}x]_N) = \pi_{t,i}([2^{r-1}c_{t,i}x]_N) + \pi_{t,i}(L_r(-\lfloor 2^{r-1}\tilde{w}_{t,i}\rfloor N)) \bmod 2. \tag{5.8}$$

Proof: To prove the claim we show that

$$\pi_{t,i}([2^{r-1}c_{t,i}x]_N) = \pi_{t,i}(2^{r-1}[c_{t,i}x]_N - \lfloor 2^{r-1}\tilde{w}_{t,i}\rfloor N) \tag{5.9}$$

and

$$\begin{aligned}\pi_{t,i}(2^{r-1}[c_{t,i}x]_N - \lfloor 2^{r-1}\tilde{w}_{t,i}\rfloor N) &= \\ \ell_1([c_{t,i}x]_N) + \pi_{t,i}(L_r(-\lfloor 2^{r-1}\tilde{w}_{t,i}\rfloor N)) \bmod 2.\end{aligned} \tag{5.10}$$

The second statement of Claim 5.4.2 implies that if Condition (5.3) holds then for all r, $1 \le r \le j$,

$$[2^{r-1}c_{t,i}x]_N = 2^{r-1}[c_{t,i}x]_N - \lfloor 2^{r-1}\tilde{w}_{t,i}\rfloor N. \tag{5.11}$$

Applying function $\pi_{t,i}$ to both sides of (5.11) gives (5.9). To prove (5.10) we first note that

$$\begin{aligned}&L_r(2^{r-1}[c_{t,i}x]_N - \lfloor 2^{r-1}\tilde{w}_{t,i}\rfloor N) = \\ &(2^{r-1}\ell_1([c_{t,i}x]_N) + L_r(-\lfloor 2^{r-1}\tilde{w}_{t,i}\rfloor N)) \bmod 2^r.\end{aligned} \tag{5.12}$$

It follows from (5.11) that $2^{r-1}[c_{t,i}x]_N - \lfloor 2^{r-1}\tilde{w}_{t,i}\rfloor N \ge 0$. Hence, in this case L_r corresponds to the r least-significant bits. Thus applying function $\pi_{t,i}$ to the left-hand side of (5.12) gives

$$\pi_{t,i}(L_r(2^{r-1}[c_{t,i}x]_N - \lfloor 2^{r-1}\tilde{w}_{t,i}\rfloor N)) = \pi_{t,i}(2^{r-1}[c_{t,i}x]_N - \lfloor 2^{r-1}\tilde{w}_{t,i}\rfloor N). \tag{5.13}$$

Then we apply $\pi_{t,i}$ to the right-hand side of (5.12):

$$\begin{aligned}&\pi_{t,i}((2^{r-1}\ell_1([c_{t,i}x]_N) + L_r(-\lfloor 2^{r-1}\tilde{w}_{t,i}\rfloor N)) \bmod 2^r) = \\ &\pi_{t,i}(2^{r-1}\ell_1([c_{t,i}x]_N) + L_r(-\lfloor 2^{r-1}\tilde{w}_{t,i}\rfloor N)) = \\ &\ell_1([c_{t,i}x]_N) + \pi_{t,i}(L_r(-\lfloor 2^{r-1}\tilde{w}_{t,i}\rfloor N)) \bmod 2,\end{aligned} \tag{5.14}$$

since $\pi_{t,i} \subseteq \{1,2,\ldots,r\}$, $r \in \pi_{t,i}$. Equalities (5.12), (5.13), and (5.14) result in (5.10). This completes the proof of the claim. □

Equations (5.7) and (5.8) give

$$\begin{aligned}\ell_1([a_tx]_N) = \pi_{t,i}([2^{r-1}c_{t,i}x]_N) + \pi_{t,i}(L_r(-\lfloor 2^{r-1}\tilde{w}_{t,i}\rfloor N)) &+ \\ \ell_1([bx]_N) + \lfloor w_{t,i}\rfloor \bmod 2,&\end{aligned}$$

which concludes the proof of Lemma 5.4.1.

5.5 Analysis of the inversion algorithm

In this section, we determine the success probability of Algorithm 5.4.1. We prove the following lemma.

Lemma 5.5.1 *Algorithm 5.4.1, given modulus N, $E_N(x)$ for $x \in \mathbb{Z}_N$, parameters j and δ, outputs x with probability at least $2/9$, where the probability is taken over internal coin flips of the algorithm (which includes the coin flips of O_{xor}).*

Note that the success probability of Algorithm 5.4.1 does not depend on N, δ, and j. This is not counterintuitive because actually the running time of the algorithm depends on these parameters (see Section 5.6).

The proof presented below is analogous to the proof of a similar statement in [FS00, Section 2].

Recall that the inversion algorithm is as follows. For $a, b \in_R \mathbb{Z}_N$, we search through a certain set of quadruples $(u_0N, vN, l_{a,0}, l_b)$ such that for at least one of them Condition (5.2) holds, i.e. $l_{a,0} = \ell_1([ax]_N)$, $l_b = \ell_1([bx]_N)$; $|[ax]_N - u_0N| \leq \eta_a N$, $|[bx]_N - vN| \leq \eta_b N$. Throughout this section we only consider a quadruple for which the condition holds (for the other quadruples we assume that the algorithm outputs x with probability 0).

At each step t, $1 \leq t < n$, the goal of the inversion algorithm is to determine $\ell_1([a_t x]_N)$. Using O_{xor}, this bit is determined by majority decision, which depends on a certain number of measurements. For all $i \in \sigma_t$, the i-th measurement is set to 0 if (5.5) holds, otherwise it is set to 1. The majority decision is correct if the majority of the measurements is correct.

Consider step t, $1 \leq t < n$. Assume that for all $t' < t$ we have determined correctly the bits $\ell_1([a_{t'} x]_N)$. There exist two reasons why for some $i \in \sigma_t$ the i-th measurement can be incorrect.

- The error of the rational approximation $w_{t,i}N$ is too large so that Condition (5.3) does not hold.
- Oracle O_{xor} outputs a wrong bit (recall that it outputs the correct bit only with probability $1/2 + \delta$).

5.5.1 Probability that the rational approximation is imprecise

Claim 5.5.2 *Assume that Condition (5.2) holds and the set of bits $\ell_1([a_{t'} x]_N)$, $t' < t$, is determined correctly. Then the probability that Condition (5.3) does not hold for some $i \in \sigma_t$ is at most $\delta/4$. Here the probability is taken over all choices of random multipliers $a, b \in_R \mathbb{Z}_N$.*

Proof: Let us rewrite (5.3) again:

$$[2^j c_{t,i} x]_N = 2^j \left([a_t x]_N (1 + 2i) + [bx]_N\right) - \lfloor 2^j w_{t,i} \rfloor N.$$

Intuitively, (5.3) does not hold if and only if there exists a multiple of N between $2^j([a_t x]_N(1 + 2i) + [bx]_N)$ and $2^j w_{t,i} N$. Denote $\Delta_{t,i} = 2^j w_{t,i} N - 2^j([a_t x]_N(1+2i)+$

$[bx]_N)$. Then, a multiple of N can exist between $2^j([a_t x]_N(1+2i)+[bx]_N)$ and $2^j w_{t,i}N$ only if

$$|\Delta_{t,i}| \geq \left|2^j([a_t x]_N(1+2i)+[bx]_N)\right|_N = \left|2^j c_{t,i}x\right|_N,$$

where $|z|_N = \min([z]_N, N-[z]_N)$ denotes the distance from z to the closest multiple of N, for any $z \in \mathbb{Z}$.

If (5.2) holds and for all $t' < t$ the bits $\ell_1([a_{t'}x]_N)$ are determined correctly then

$$|[a_t x]_N - u_t N| = 2^{-t}([ax]_N - u_0 N) \leq 2^{-t-j-4}\delta^3 N,$$
$$|[bx]_N - v| \leq 2^{-j-4}\delta N.$$

Due to the choice of the multisets σ_t, $2^{-t}\delta^2|1+2i| \leq 1$ for $i \in \sigma_t$ (see also the end of Section 5.4). Therefore, the triangular inequality gives

$$|\Delta_{t,i}| = 2^j\,|u_t N(1+2i) - [a_t x]_N(1+2i) + vN - [bx]_N| \leq$$
$$\frac{\delta}{16}(2^{-t}\delta^2|1+2i|+1)N \leq \frac{\delta}{8}N.$$

Thus (5.3) does not hold only if $\left|2^j c_{t,i}x\right|_N \leq \delta N/8$. Since $c_{t,i}$ is uniformly distributed in $\mathbb{Z}_N$, the probability that (5.3) does not hold is at most $\delta/4$. This completes the proof of Claim 5.5.2. □

5.5.2 Error probability of the majority decisions

The i-th measurement of $\ell_1([a_t x]_N)$ is correct if (5.3) holds and the output of O_{xor} is correct. Following the notation of [FS00], we define Boolean variables X_i such that $X_i = 1$ implies that the i-th measurement is incorrect:

$X_i = 1$ if and only if (5.3) does not hold or $O_{xor}([c_{t,i}x]_N, \pi_{t,i}) \neq \pi_{t,i}([c_{t,i}x]_N)$.

It is shown in [FS00] that for any fixed t, $1 \leq t < n$, the $c_{t,i}$'s are pairwise independent. Thus Boolean variables X_i, $i \in \sigma_t$, are also pairwise independent.

The majority decision errs if and only if more than half of the measurements are incorrect, that is,

$$\frac{1}{m_t}\sum_{i\in\sigma_t} X_i > \frac{1}{2}. \tag{5.15}$$

Due to the different choice of σ_t for $t < \log_2 n + 4$ and for $t \geq \log_2 n + 4$ (see Section 5.4) we divide our analysis into two parts.

Case $t < \log_2 n + 4$.

Consider step t with $t < \log_2 n + 4$. Since O_{xor} guesses correctly with probability $\frac{1}{2}+\delta$, Claim 5.5.2 implies that the expected value $\mathsf{E}[X_i] \leq 1/2 - 3\delta/4$. Thus, it follows from Inequality (5.15) that the majority decision errs if and only if

$$\frac{1}{m_t}\sum_{i\in\sigma_t} X_i - \mathsf{E}[X_i] \geq \frac{3}{4}\delta.$$

Since the variance of any Boolean variable is at most 1/4 we have $\mathsf{Var}(X_i) \leq 1/4$. Then, Chebyshev's inequality for μ_t pairwise independent random variables X_i gives

$$\Pr\left[\frac{1}{m_t}\sum_{i\in\sigma_t} X_i - \mathsf{E}[X_i] \geq \frac{3}{4}\delta\right] \leq \left(\frac{3}{4}\delta\right)^{-2} \mathsf{Var}\left(\frac{1}{m_t}\sum_{i\in\sigma_t} X_i\right) \leq \frac{4}{9m_t\delta^2}.$$

Here the probability is taken over all choices of random multipliers $a, b \in_R \mathbb{Z}_N$, and internal coin flips of O_{xor}.

Since $m_t = 2^t\delta^{-2}$, the majority decision for $\ell_1([a_t x]_N)$ errs with probability $\frac{4}{9}2^{-t}$. Thus the probability that at least one of the majority decisions for $t < \log_2 n + 4$ errs is at most 4/9.

Case $t \geq \log_2 n + 4$.

Consider step t with $t \geq \log_2 n + 4$. The technique we use in this case is called the subsample majority decision. It is proposed by Fischlin and Schnorr [FS00]. Instead of using indices from a large sample $\rho = \{i \mid |1 + 2i| < 16n\delta^{-2}\}$ we use only indices from a small random subsample $\sigma = \{i_1, \ldots, i_m\} \subset \rho$, where $m = 2\delta^{-2}\log_2 n$ (see also Section 5.4). Although the original random variables X_i's, $i \in \rho$, are just pairwise independent we note that the random variables $X_{i_1}, \ldots, X_{i_m}$ are mutually independent. Therefore, for the latter random variables we can use a stronger bound instead of Chebyshev's inequality, namely Hoeffding's bound [Hoe63].

Similarly to the previous case, the majority decision errs if and only if

$$\frac{1}{m}\sum_{i_s\in\sigma} X_{i_s} - \mathsf{E}[X_i] \geq \frac{3}{4}\delta. \tag{5.16}$$

Let

$$X = \frac{1}{|\rho|}\sum_{i\in\rho} X_i.$$

Then, Condition (5.16) implies that either

$$\frac{1}{m}\sum_{i_s\in\sigma} X_{i_s} - X \geq \frac{1}{2}\delta$$

or $X - \mathsf{E}[X_i] \geq \delta/4$. Chebyshev's inequality for pairwise independent X_i's, $i \in \rho$, gives $\Pr[X - \mathsf{E}(X_i) \geq \delta/4] \leq 4/(|\rho|\delta^2)$. Hoeffding's bound [Hoe63] implies that for a fixed set of X_i's, $i \in \rho$, and a random subsample $\sigma \subset \rho$

$$\Pr\left[\frac{1}{m}\sum_{i_s\in\sigma} X_{i_s} - X \geq \frac{1}{2}\delta\right] \leq \exp\left(-2m\left(\frac{\delta}{2}\right)^2\right) = \exp\left(-\frac{1}{2}m\delta^2\right). \tag{5.17}$$

Since $m = 2\delta^{-2}\log_2 n$ and $|\rho| = 16n\delta^{-2}$ the majority decision at each step t with $t \geq \log_2 n + 4$ errs with probability at most $4/(|\rho|\delta^2) + \exp(-m\delta^2/2) = 1/(4n) +$

$n^{-1/\ln 2} < 1/(3n)$ for $n > 2^9$. Thus the probability that at least one of the subsample majority decisions for $t \geq \log_2 n + 4$ errs is at most 1/3.

Therefore the inversion algorithm of Section 5.4, given $E_N(x)$, j, and N, outputs x with probability at least $1 - (4/9 + 1/3) = 2/9$. It completes the proof of Lemma 5.5.1.

5.6 Main result

In this section, we determine the running time of the inversion algorithm and state the main result of this chapter. We give a concrete bound for the running time and the success probability of the adversary that aim to distinguish the output of the RSA generator from a uniformly random sequence.

5.6.1 Computational cost of arithmetic operations

To determine the running time of Algorithm 5.4.1 and to express the result of Fischlin and Schnorr [FS00] in terms of our time units, the computational cost of several arithmetic operations has to be estimated.

We have performed a simulation using the GMP library [GNU] and the eBATS tools [eBA]. The simulation shows that adding two l-bit numbers for l in the range 32 to 128 takes about 300 time units. In turn, multiplying an l_1-bit integer by an l_2-bit integer takes about $l_1 l_2/70$ time units provided that l_1 is in the range 32 to 128 and l_2 is in the range 4096 to 16384. We have also analyzed the computational cost of each iteration of the inner loop in the second part of Algorithm 5.4.1. The computational cost of each iteration is dominated by the computational cost of the multiplication $u_t \cdot i$, which is confirmed by our simulation.

The above estimates fit all our data points with error less than 10%. Lemma 5.6.1 and the subsequent results rely on these estimates.

5.6.2 Running time of the inversion algorithm

Lemma 5.6.1 *The expected running time of Algorithm* 5.4.1 *is at most*

$$2n(\log_2 n + 8)\delta^{-2}(T_{O_{xor}} + 2^{2j+8}\delta^{-4}\Omega),$$

where $\Omega = (n + 3\log_2(1/\delta) + j + 4)\log_2(8n\delta^{-2})/70$.

Proof: First, note that the total number of measurements of the least significant bits $\ell_1([a_t x]_N)$, $t = 1, 2, \ldots, n-1$, is

$$\sum_{t=1}^{n-1} m_t = \sum_{t=1}^{\log_2 n+3} 2^t\delta^{-2} + (n - 1 - (\log_2 n + 3))2\delta^{-2}\log_2 n < 2n(\log_2 n + 8)\delta^{-2}.$$

Therefore, the computational cost of the first part of Algorithm 5.4.1 is at most $2n(\log_2 n + 8)\delta^{-2}T_{O_{xor}}$.

In the second part of Algorithm 5.4.1, some arithmetic operations are performed. As mentioned in Section 5.6.1, the computational cost of one iteration of the inner loop in the second part of the algorithm is about the same as that of the multiplication $u_t \cdot i$. The bit length of u_t is $t + \log_2(1/\eta_a) < n + \log_2(1/\eta_a)$; the bit length of i is at most $\log_2(8n\delta^{-2})$ (for a practical choice of parameters n, j, and ϵ, the bit length of i is in the range 32 to 128 while the sum $n+\log_2(1/\eta_a)$ is in the range 4096 to 16384). Therefore, the computational cost of the multiplication $u_t \cdot i$ is less than $(n + \log_2(1/\eta_a)) \log_2(8n\delta^{-2})/70$. Hence the total computational cost of the second part of the algorithm is bounded by the product of the following factors:

1. Number of quadruples $(u_0N, vN, l_{a,0}, l_b)$, which equals $1/(\eta_a\eta_b)$;
2. Number of iterations of the inner loop for each quadruple, which is less than $2n(\log_2 n + 8)\delta^{-2}$;
3. Computational cost of one iteration of the inner loop, which is less than $\Omega = (n + \log_2(1/\eta_a)) \log_2(8n\delta^{-2})/70$.

Since $\eta_a = 2^{-j-4}\delta^3, \eta_b = 2^{-j-4}\delta$, the computational cost of the second part is at most $2^{2j+9}\delta^{-6}n(\log_2 n + 8)\Omega$.

All in all, the expected running time of Algorithm 5.4.1 is at most $2n(\log_2 n + 8)\delta^{-2}(T_{O_{xor}} + 2^{2j+8}\delta^{-4}\Omega)$ time units. □

5.6.3 Main theorem

Now, we are ready to state the main theorem of this chapter which states that the RSA generator is secure under the RSA assumption (Assumption 2.3.2).

Theorem 5.6.2 *Under Assumption* 2.3.2*, the RSA generator is* (T, ϵ)*-secure if*

$$T \le \frac{L(n)}{9n(\log_2 n + 8)\delta^{-2}} - 2^{2j+8}\Omega\delta^{-4} - Mn^2/24, \tag{5.18}$$

where $\delta = (2^j - 1)^{-1}(\epsilon/M)$ *and* $\Omega = (n + 3\log_2(1/\delta) + j + 4)\log_2(8n\delta^{-2})/70$.

Proof: Suppose the RSA generator is *not* (T, ϵ)-secure. Then, it follows from Lemma 5.2.1 that there exists an algorithm O_{xor} that on input $(\pi, E_N(x))$ guesses $\pi(x)$ with probability at least $1/2 + \delta$, where $\delta = (2^j - 1)^{-1}(\epsilon/M)$. The running time $T_{O_{xor}}$ of O_{xor} satisfies $T_{O_{xor}} \le T + Mn^2/24$. Finally, Lemmas 5.5.1 and 5.6.1 imply that O_{xor} can be used to solve the RSA problem in expected time less than $9n(\log_2 n + 8)\delta^{-2}(T + Mn^2/24 + 2^{2j+8}\delta^{-4}\Omega)$. According to Assumption 2.3.2, no algorithm can solve the RSA problem faster than the NFS algorithm, which means that $9n(\log_2 n + 8)\delta^{-2}(T + Mn^2/24 + 2^{2j+8}\delta^{-4}\Omega) \le L(n)$. □

Note that $Mn^2/24 \ll 2^{2j+8}\Omega\delta^{-4}$, for a practical choice of parameters. For instance, if $\epsilon = 0.01$, $M = 2^{20}$, $n = 13000$, $j = 6$ then $Mn^2/24 \simeq 2^{43}$ while $2^{2j+8}\Omega\delta^{-4} \simeq 2^{165}$.

5.6.4 Comparison with known results

In this section, we compare Theorem 5.6.2 with the results of [ACGS88; FS00]. One can observe that our security reduction is not tight, meaning that the running time of the solver for the RSA problem is significantly larger than the running time of the adversary that breaks the pseudorandom generator. However, it turns out that our security proof is still tighter than the one of [ACGS88].

Alexi et al. [ACGS88] prove the following result (actually, in [ACGS88] this result is stated without paying attention to the specific constant in the denominator but the constant is mentioned in the proof).

Theorem 5.6.3 (Alexi et al.) *Under Assumption* 2.3.2*, the RSA generator is* (T,ϵ)*-secure if*

$$T \le \frac{L(n)}{2^{4j+21} n^3 (\epsilon/M)^{-8}}. \tag{5.19}$$

The latter result is weaker than Theorem 5.6.2 both in asymptotic and concrete sense. This is because the right-hand side of Condition (5.19) contains M^8 in the denominator while the first component of the right-hand side of Condition (5.18) contains just M^2. For instance, let $M = 2^{20}$ and $j = 6$. Suppose the RSA generator has to be protected against adversaries with running time at most $T = 2^{120}$ and advantage $\epsilon = 0.01$. Then, Theorem 5.6.2 guarantees security if n is at least 13000. In contrast, Theorem 5.6.3 guarantees security only for $n \simeq 50000$.

The following lemma is due to Fischlin and Schnorr [FS00]. It is about the RSA generator with 1 output bit per iteration.

Theorem 5.6.4 (Fischlin and Schnorr) *Under Assumption* 2.3.2*, the RSA generator with* 1 *output bit per iteration is* (T,ϵ)*-secure if*

$$T \le \frac{L(n)}{3n(\log_2 n + 8)(\epsilon/M)^{-2}} - \frac{2^8 n (\epsilon/M)^{-2} \log_2(8n(\epsilon/M)^{-1})}{\log_2 n + 8} \cdot 300 - Mn^2/24. \tag{5.20}$$

The factor of 300 in the second term on the right-hand side of Condition (5.20) is due to the fact that addition of integers of bit length up to 176, which is done in the algorithm described in [FS00, Section 4], takes about 300 time units (cf. Section 5.6.1).

The first component in Condition (5.20) is essentially the same as the first component in Condition (5.18) for $j = 1$. On the other hand, the absolute value of the second component in (5.20) is smaller than the absolute value of the second component in (5.18) by a factor of $(\epsilon/M)^{-2}$ so for $j = 1$ the reduction [FS00] is more efficient than our reduction. The reason is that there is a trick in the reduction [FS00] (namely, processing all approximate locations simultaneously) that allows to decrease the second component. We do not know how to apply this trick for $j > 1$.

5.6.5 Choice of parameters for the RSA generator

The following example shows how to choose parameters of the RSA generator in order to minimize the computational effort for a given level of provable security.

Suppose our goal is to generate a sequence of $M = 2^{20}$ bits such that no adversary can distinguish this sequence from uniformly random binary sequence with advantage $\epsilon = 0.01$ in time T, which is specified later in this section. The question is what length of the modulus n and parameter j should be used to minimize the computational effort per output bit.

Conditions (5.18) and (5.20) connect characteristics of the adversary (T, ϵ) with parameters of the RSA generator (M, n, j) for $j \geq 1$ and $j = 1$ respectively. In order to find the optimal values of n and j we fix T, ϵ, M and analyze n as a function of j.

For public exponents e with small Hamming weight, e.g., for $e = 3$, the computational work of the RSA generator is proportional to n^2/j (each modular multiplication costs $O(n^2)$ time units, so the generation of an output sequence takes $O(Mn^2/j)$ time units).

Figure 5.1 presents the computational work of the RSA generator for j in the range 1 to 15. On this figure, we consider two cases $T = 2^{120}$ and $T = 2^{88}$. There are three values of the computational work for $j = 1$ on the figure. Two of them (marked by stars) result from Condition (5.20) for $T = 2^{120}$ and $T = 2^{88}$ respectively while the third one (denoted by a diamond) results from Condition (5.18). Note that our reduction yields almost the same points for all T in the range $2^{88} \leq T \leq 2^{120}$.

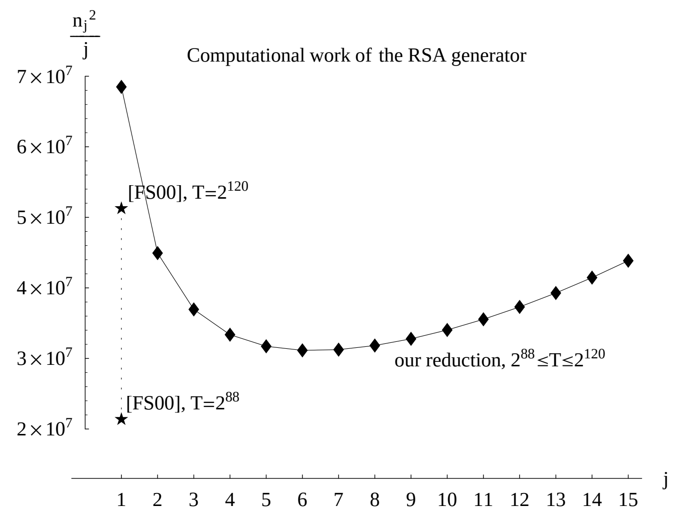

Figure 5.1: Computational work of the RSA generator as a function of the number of bits extracted per iteration. Here we fix $M = 2^{20}$, $\epsilon = 0.01$ and consider two cases $T = 2^{120}$ and $T = 2^{88}$. For $T = 2^{120}$, extraction of 6 bits per iteration makes the RSA generator about 5/3 times faster in comparison with the 1-bit case. On the other hand, for $T = 2^{88}$ the computational work is minimal for $j = 1$.

It turns out that for $T = 2^{120}$ extraction of 6 bits per iteration makes the RSA generator about 5/3 times faster in comparison with the 1-bit case. However, even for $j = 6$ the RSA generator is quite slow since the corresponding length of the modulus is $n \simeq 13000$. On the other hand, for $T = 2^{88}$ the computational work is minimal for $j = 1$. The reason is that for $T = 2^{88}$ the second component in (5.18) is relatively large.

5.6.6 Sensitivity of the results

Note that Conditions (5.18) and (5.20) contain constants (e.g., 70 and 300) that characterize the computational cost of certain arithmetic operations. Although these constants are fairly accurately estimated in Section 5.6.1, they can in general depend on the platform. The coefficient $1.7 \cdot 10^{-2}$ in Formula (2.2) also depends on the platform.

Fortunately, Conditions (5.18) and (5.20) are not very sensitive to small changes in these constants. Consider, for instance, the following equation derived from Condition (5.18).

$$c_1 \cdot \frac{L(n)}{9n(\log_2 n + 8)\delta^{-2}} - c_2 \cdot 2^{2j+8}\Omega\delta^{-4} - c_3 \cdot Mn^2/24 - T = 0, \tag{5.21}$$

where all the parameters are as above and c_1, c_2, c_3 are positive constants. Let $M = 2^{20}$, $j = 6$, $\epsilon = 0.01$, and $T = 2^{88}$. Then for all c_i, $i = 1, 2, 3$, such that $-4 \leq \log_2 c_i \leq 4$ the roots n of Equation (5.21) do not differ more than 10%. A similar observation can be done for the equation derived from Condition (5.20). It shows that our analysis is meaningful.

5.7 The Blum-Blum-Shub generator

Let $N = pq$, $p \equiv q \equiv 3 \bmod 4$. Let QR_N be the set of quadratic residues modulo N. As briefly mentioned in Section 5.1, the Blum-Blum-Shub generator (the BBS generator) is a pseudorandom generator that on input $x \in_R QR_N$ iterates the Rabin function $R_N(x) = [x^2]_N$ and outputs $L_j(x)$, for a fixed j such that $1 \leq j < n$. Since squaring is faster than general exponentiation, the BBS generator is more efficient than the RSA generator, for the same number of output bits.

In two recent papers [FS00, Section 6] and [SS05], the modification of the BBS generator is analyzed. The modified generator iterates the function $R'_N(x) = |[x^2]_N|_N$, where $|\cdot|_N$ denotes the distance to the closest multiple of N, as before. Fischlin and Schnorr [FS00, Section 6] construct a relatively tight reduction for the BBS generator with $j = 1$. Sidorenko and Schoenmakers [SS05] generalize their result for the case $j > 1$.

The reductions for the BBS generator are similar to the reductions for the RSA generator. The main difference is that in the former case not all values $i \in \sigma_t$ can be used for the majority decision on step t but only those for which $2^{r-1}c_{t,i} \in QR_N$, $1 \leq r \leq j$ (see Section 5.4).

Let $j = 1$, for simplicity. Recall that $c_{t,i} = a_t(1 + 2i) + b$, $b \in_R \mathbb{Z}_N$. The argument in [FS00, Section 6] is based on the result of Peralta [Per92] which implies that for every fixed $t \in \{1, 2, \ldots, n-1\}$ the sequence of Jacobi symbols of $c_{t,i}$, $i \in \sigma_t$, deviates from the uniform distribution on $\{\pm 1\}^{m_t}$ by at most $m_t(3+\sqrt{p'})/p'$, where m_t is the size of σ_t, $p' = \min\{p, q\}$.

Unfortunately, the way the result of Peralta [Per92] is interpreted in [FS00, Section 6] is not correct. In fact, the result implies that the probability that the sequence of Jacobi symbols of $c_{t,i}$, $i \in \sigma_t$, matches any particular element of $\{\pm 1\}^{m_t}$ is in the range $1/2^{m_t} \pm m_t(3+\sqrt{p'})/p'$. Note that $m_t \gg n$ for some values of t (see [FS00, Section 2]) so $1/2^{m_t} \ll m_t(3+\sqrt{p'})/p'$. Therefore, the result of Peralta does *not* necessarily imply that the fraction of $c_{t,i}$'s with Jacobi symbol $+1$ is close to $1/2$ so it is not clear if there are enough measurements to make the majority decisions reliable.

For $j > 1$, as considered in [SS05], a similar problem occurs. In this case the sequence of numbers $2^{r-1}c_{t,i}$ cannot even be represented as $c'_{t,i} + b'$, where $b' \in_R \mathbb{Z}_N$ and $c'_{t,i}$ does not depend on b' so the result [Per92] is not applicable. That is why we have chosen to focus on the RSA generator here.

5.8 Conclusion

According to Theorem 3 from [FS00], inverting the RSA one-way function $E_N(x)$ in the oracle access model (if the only source of information about x is an oracle that on input $E_N(x)$ predicts one bit of x with probability $1/2+\delta$) requires $(\ln 2/4)n\delta^{-2}$ oracle calls. Recall that our inversion algorithm calls the oracle on average less than $9n(\log_2 n + 8)\delta^{-2}$ times. Therefore, the number of oracle calls in our inversion algorithm is optimal up to a factor of $\log n$. On the other hand, besides using the oracle our inversion algorithm also involves some additional computations, which result in the second component in Condition (5.18). Although in some situations this component turns out to be not significant in other situations it can be too large (cf. Section 5.6.5). Reducing the second component in (5.18) is an important open problem.

A new way of analyzing the RSA generator is proposed in a recent paper of Steinfeld et al. [SPW06]. They show that under a stronger assumption (namely, under the assumption that the small-solution RSA problem is intractable) the generator remains secure if it outputs $O(n)$ bits per iteration. Thus, their approach gives rise to the first pseudorandom generator beating the rate $O(\log n)$ bits per modular multiplication. Note, however, that the strong assumption used by Steinfeld et al. [SPW06] is not as thoroughly studied as the original RSA assumption. This is similar to our result of Chapter 4, where we have shown that under the DDH assumption (rather than the weaker DL assumption) one gets an efficient pseudorandom generator with a tight security reduction. The difference is that the DDH assumption is a standard assumption, in contrast with the intractability of the small-solution RSA problem.

Chapter 6

Pseudorandom generator as a building block of a secure scheme

A chain is no stronger than its weakest link.

English proverb

In this chapter, we consider a pseudorandom generator as a part of a more complex cryptographic system. In particular, we analyze the concrete security of probabilistic digital signature schemes and public key encryption schemes that use a pseudorandom generator as a source of randomness. We show that if the scheme is secure under the assumption of the availability of a source of independent uniformly distributed random bits it remains secure if one uses the output of a secure pseudorandom generator instead of this source.

6.1 Introduction

Pseudorandom generators have many applications in the field of cryptography. One of the most important applications is to provide randomness for other cryptographic schemes, e.g., probabilistic digital signature schemes and public key encryption schemes. Security of the latter schemes is usually proved under the assumption that a source of independent uniformly distributed random bits is available. However, as discussed in Chapter 1, random bits are often difficult to get. In this chapter, we show that the *composition* of a secure cryptographic scheme and a secure pseudorandom generator results in a secure scheme. That is, if a scheme is secure under the assumption that a source of independent uniformly distributed random bits is available it remains secure if one uses the output of a secure pseudorandom generator instead of this source.

The framework of universal composability proposed by Canetti [Can01] and independently by Pfitzmann and Waidner [PW00] implies that a cryptographic protocol that uses another protocol as a subroutine is secure provided that the building blocks are secure. Security of protocols is defined by comparing the protocol execution to an ideal process for carrying out the task at hand. A protocol is said to be secure if no polynomial-time adversary can tell the difference between the protocol execution and the ideal process for this functionality, except for negligible probability (for more details, refer to [Can01]).

According to Definition 2.2.1, an output of a secure pseudorandom generator is indistinguishable from a sequence of independent uniformly distributed random bits. Therefore, the composition theorem [Can01; PW00] implies that if a probabilistic cryptographic scheme is secure under the assumption that the input bits are uniformly random then the corresponding scheme that uses pseudorandom bits instead of uniformly random bits is also secure.

Note, however, that the composition theorem [Can01; PW00] is stated in terms of asymptotic security. That is, the theorem implies that the composed scheme is secure against polynomial-time adversaries. The contribution of this chapter is concrete security analysis of probabilistic cryptographic schemes (i.e., digital signature schemes and public key encryption schemes) that use pseudorandom generators as a source of randomness, which means that the running time and the success probability of the adversary are limited by concrete values of T and ϵ (the concrete security approach is discussed in detail in Section 1.5).

We show how to compute concrete parameters of cryptographic schemes such that the corresponding compositions reach a desired level of security. As an example, concrete parameters for two schemes designed by Cramer and Shoup [CS00; CS98] combined with the pseudorandom generator PRG_1 presented in Section 4.3.1 are computed.

6.2 Notions of security

In this section, we recall the commonly used notions of security for digital signature schemes and public key encryption schemes.

6.2.1 Signature schemes

Let S be a probabilistic signature scheme consisting of a key generation algorithm, signing algorithm, and verification algorithm. In this chapter, we consider a standard notion of security for signature schemes, called existential unforgeability under an adaptive chosen message attack [GMR88], which is defined using the following game between a challenger and an adversary.

First, the challenger runs the key generation algorithm of S. Then the public key is given to the adversary. Proceeding adaptively, the adversary makes queries to a signing oracle, to sign at most q_{sig} messages of his choice.

The goal of the adversary is then to forge a signature for a message that has not been signed by the oracle on the previous step. Let p be the probability that the adversary outputs a valid forgery taken over the coin flips made by the adversary, the key generation algorithm, and the signing oracle. We will refer to p as the success probability of the adversary. The adversary (T, ϵ, q_{sig})-breaks the scheme S if he runs in time at most T and achieves $p \geq \epsilon$.

Definition 6.2.1 *A signature scheme* S *is* (T, ϵ, q_{sig})*-existentially unforgeable under an adaptive chosen message attack if no adversary* (T, ϵ, q_{sig})*-breaks* S.

6.2.2 Public key encryption schemes

Let E be a probabilistic public key encryption scheme consisting of a key generation algorithm, encryption algorithm, and decryption algorithm. A standard notion of security for public key encryption schemes is security against adaptive chosen ciphertext attacks IND-CCA2 [GM84; NY90; RS92]. This notion is defined using the following game between a challenger and an adversary.

First, the challenger runs the key generation algorithm of E and gives the public key to the adversary. Next, the adversary makes queries to a decryption oracle, to decrypt ciphertexts of his choice. The queries may depend on the previous queries and answers. Then the adversary chooses two messages m_0 and m_1 and sends them to the challenger. The challenger chooses a bit $b \in \{0, 1\}$ uniformly at random and gives an encryption of m_b to the adversary. After receiving the ciphertext from the challenger, the adversary continues to query the decryption oracle, subject only to the restriction that the queries should be different from the output of the challenger.

At the end of the game the adversary outputs $b' \in \{0, 1\}$, as his guess of the value b. If the probability that $b' = b$ is $1/2 + \epsilon$, then the advantage of the adversary is defined to be ϵ. Here the probability is taken over the coin flips made by the adversary, the key generation algorithm, and the decryption oracle.

The adversary (T, ϵ, q_{dec})-breaks the encryption scheme E if he queries the decryption oracle at most q_{dec} times, runs in time at most T, and guesses b with advantage at least ϵ.

Definition 6.2.2 *An encryption scheme* E *is* (T, ϵ, q_{dec})*-secure if no adversary* (T, ϵ, q_{dec})*-breaks the scheme.*

6.3 Concrete security of the compositions

In this section, we show that if a scheme and a pseudorandom generator are secure then the composition of the scheme and the pseudorandom generator, that is, the scheme that uses the output of the pseudorandom generator as a source of randomness is also secure. We start with an informal argument that demonstrates the basic idea of the approach.

Consider an abstract cryptographic scheme C that makes use of uniformly random bits. Suppose C is $(T, \epsilon_{\mathsf{C}})$-secure for some parameters $T, \epsilon_{\mathsf{C}} > 0$ according to a certain definition of security under the assumption that the input bits are independent and uniformly distributed. Let M be the total number of random bits used by the scheme during the period of time when it is observable by the adversary.

Now, let PRG be a $(T, \epsilon_{\mathsf{PRG}})$-secure pseudorandom generator that outputs M bits, where $\epsilon_{\mathsf{PRG}} > 0$. Denote by $\mathsf{C}_{\mathsf{PRG}}$ the scheme which is the same as C except that it uses pseudorandom bits produced by PRG (on uniformly random input) instead of uniformly random bits.

We now proceed to show that under the above conditions, the composition $\mathsf{C}_{\mathsf{PRG}}$ is $(T, \epsilon_{\mathsf{C}} + \epsilon_{\mathsf{PRG}})$-secure. Indeed, suppose this is not the case, i.e., there exists an adversary $\mathcal{A}$ that $(T, \epsilon_{\mathsf{C}} + \epsilon_{\mathsf{PRG}})$-breaks the composition $\mathsf{C}_{\mathsf{PRG}}$. The trick is to ask $\mathcal{A}$ to break the original scheme C. It can happen that when trying to break C the adversary does not halt in time T. In this case, we make $\mathcal{A}$ stop and assume that it fails to break the scheme in time T. Let ϵ be the probability that $\mathcal{A}$ breaks C. There are two possibilities for ϵ.

1. Case $\epsilon < \epsilon_{\mathsf{C}}$. In this case, $\mathcal{A}$ provides a $(T, \epsilon_{\mathsf{PRG}})$-distinguisher for the pseudorandom generator PRG. Indeed, consider algorithm $\mathcal{D}$ that is given either a pseudorandom sequence produced by PRG or a sequence of M uniformly random bits. Let r denote the input of $\mathcal{D}$. $\mathcal{D}$ proceeds as follows. $\mathcal{D}$ asks $\mathcal{A}$ to break the scheme C that uses r as a source of randomness. If $\mathcal{A}$ succeeds $\mathcal{D}$ outputs 1, otherwise $\mathcal{D}$ outputs 0. Then,

$$\Pr[\mathcal{D}(\mathsf{PRG}(U_n)) = 1] = \Pr[\mathcal{A} \text{ breaks } \mathsf{C}_{\mathsf{PRG}}] \geq \epsilon_{\mathsf{C}} + \epsilon_{\mathsf{PRG}}$$

   and

$$\Pr[\mathcal{D}(U_M) = 1]| = \Pr[\mathcal{A} \text{ breaks } \mathsf{C}] = \epsilon < \epsilon_{\mathsf{C}},$$

   where n denotes the seed length of the generator PRG. Therefore,

$$|\Pr[\mathcal{D}(\mathsf{PRG}(U_n)) = 1] - \Pr[\mathcal{D}(U_M) = 1]| > \epsilon_{\mathsf{PRG}}.$$

2. Case $\epsilon \geq \epsilon_{\mathsf{C}}$. In this case, $\mathcal{A}$ $(T, \epsilon_{\mathsf{C}})$-breaks the scheme C.

In both cases we have a contradiction since PRG is assumed to be a $(T, \epsilon_{\mathsf{PRG}})$-secure pseudorandom generator and C is assumed to be a $(T, \epsilon_{\mathsf{C}})$-secure scheme. The contradiction proves that the composition $\mathsf{C}_{\mathsf{PRG}}$ is indeed $(T, \epsilon_{\mathsf{C}} + \epsilon_{\mathsf{PRG}})$-secure.

Although the above argument is informal, the reader can notice that *it is suitable for a wide range of probabilistic cryptographic schemes*. In this chapter, we focus only on digital signature schemes and public key encryption schemes.

6.3.1 Signature schemes composed with pseudorandom generators

Let S be a probabilistic signature scheme. Let l_{sig} be the number of random bits used by S to produce a single signature. Let PRG be a pseudorandom generator that produces $q_{sig} l_{sig}$ bits for some $q_{sig} > 0$. Consider the scheme $\mathsf{S}_{\mathsf{PRG}}$ which is identical to S except that it uses PRG as a source of randomness. More precisely, the signing algorithm of $\mathsf{S}_{\mathsf{PRG}}$ first initializes PRG with uniformly random seed and then uses the successive bits of its output to produce signatures.

Let $r \in \{0,1\}^{q_{sig} l_{sig}}$ be the pseudorandom sequence generated by PRG on uniformly random input. We define security of $\mathsf{S}_{\mathsf{PRG}}$ in a similar way as described in

Section 6.2.1. The only difference is that now the signing oracle uses the successive bits of r (instead of uniformly random bits) to compute signatures. The goal of the adversary is to forge a signature for a message that has not been signed by the oracle. The adversary (T, ϵ, q_{sig})-breaks the scheme $\mathsf{S}_{\mathsf{PRG}}$ if he runs in time at most T and outputs a valid forgery with probability at least ϵ.

Definition 6.3.1 *The composition* $\mathsf{S}_{\mathsf{PRG}}$ *of a signature scheme* S *and a pseudorandom generator* PRG *is* (T, ϵ, q_{sig})*-existentially unforgeable under an adaptive chosen message attack if no adversary* (T, ϵ, q_{sig})*-breaks* $\mathsf{S}_{\mathsf{PRG}}$.

The following theorem shows that if a signature scheme and a pseudorandom generator are secure then their composition is also secure.

Theorem 6.3.2 *If* S *is* $(T, \epsilon_{\mathsf{S}}, q_{sig})$*-existentially unforgeable,* PRG *is* $(T, \epsilon_{\mathsf{PRG}})$*-secure, and* $\epsilon_{\mathsf{S}} + \epsilon_{\mathsf{PRG}} < 1$ *then* $\mathsf{S}_{\mathsf{PRG}}$ *is* $(T, \epsilon_{\mathsf{S}} + \epsilon_{\mathsf{PRG}}, q_{sig})$*-existentially unforgeable.*

Proof: The proof of this theorem 6.3.2 is a formal version of the argument in the beginning of Section 6.3 for the case of probabilistic signature schemes.

Suppose there exists an adversary $\mathcal{F}$ that $(T, \epsilon_{\mathsf{S}}+\epsilon_{\mathsf{PRG}}, q_{sig})$-breaks $\mathsf{S}_{\mathsf{PRG}}$. Denote $M = q_{sig} l_{sig}$. Let $\mathcal{C}$ be an algorithm (challenger) that on input $r \in \{0,1\}^M$ does the following. $\mathcal{C}$ runs the key generation algorithm of S. Next, $\mathcal{C}$ gives the public key to the adversary $\mathcal{F}$ and runs $\mathcal{F}$ to forge a signature for S. Whenever $\mathcal{F}$ asks $\mathcal{C}$ to sign a message, $\mathcal{C}$ produces the signature using the successive bits of r. Since the length of r is $M = q_{sig} l_{sig}$, $\mathcal{C}$ has enough bits to produce q_{sig} signatures. If $\mathcal{F}$ does not halt in time T or asks for more than q_{sig} signatures, we make $\mathcal{F}$ stop and assume that it fails to forge a signature in time T. In case $\mathcal{F}$ forges a signature for a message that has not been signed by $\mathcal{C}$ on the previous step in time T, $\mathcal{C}$ outputs 1.

For $r = \mathsf{PRG}(U_n)$, where n denotes the seed length of the generator PRG, $\mathcal{F}$ forges a signature with probability at least $\epsilon_{\mathsf{S}} + \epsilon_{\mathsf{PRG}}$. Let ϵ be the probability that $\mathcal{F}$ forges a signature for $r = U_M$. There are two possibilities for ϵ.

1. Case $\epsilon < \epsilon_{\mathsf{S}}$. In this case, $\mathcal{C}$ is a $(T, \epsilon_{\mathsf{PRG}})$-distinguisher for the generator PRG. Indeed,

   $$\Pr[\mathcal{C}(\mathsf{PRG}(U_n)) = 1] = \Pr[\mathcal{F} \text{ forges a signature for } \mathsf{S}_{\mathsf{PRG}}] \geq \epsilon_{\mathsf{S}} + \epsilon_{\mathsf{PRG}}$$

   and

   $$\Pr[\mathcal{C}(U_M) = 1] = \Pr[\mathcal{F} \text{ forges a signature for } \mathsf{S}] = \epsilon < \epsilon_{\mathsf{S}}$$

   so

   $$|\Pr[\mathcal{C}(\mathsf{PRG}(U_n)) = 1] - \Pr[\mathcal{C}(U_M) = 1]| > \epsilon_{\mathsf{PRG}}.$$

2. Case $\epsilon \geq \epsilon_{\mathsf{S}}$. In this case, $\mathcal{F}$ $(T, \epsilon_{\mathsf{S}}, q_{sig})$-breaks S.

The contradiction proves the theorem. □

6.3.2 Encryption schemes composed with pseudorandom generators

Let E be a probabilistic encryption scheme and let l_{enc} be the number of random bits used by E to produce a single encryption. Let PRG be a pseudorandom generator that produces $q_{enc}l_{enc}$ bits for some $q_{enc} > 0$. We denote by $\mathsf{E}_{\mathsf{PRG}}$ the encryption scheme which is identical to E except that it uses PRG as a source of randomness. To be precise, the encryption algorithm of $\mathsf{E}_{\mathsf{PRG}}$ first initializes PRG with uniformly random seed and then uses the successive bits of its output to encrypt messages.

Let $r \in \{0,1\}^{q_{enc}l_{enc}}$ be the pseudorandom sequence generated by PRG on uniformly random input. To define the security of $\mathsf{E}_{\mathsf{PRG}}$, we slightly change the standard setting discussed in Section 6.2.2. We let the adversary access not only the decryption oracle but also the *encryption oracle* that uses the subsequent bits of r to encrypt messages. After the adversary has chosen two messages m_0 and m_1 a message m_b, $b \in_R \{0,1\}$, is encrypted using the current bits of r. The encryption is given to the adversary. Then, the adversary continues to query the encryption oracle and the decryption oracle. As before, the goal of the adversary is to guess b. The adversary $(T, \epsilon, q_{enc}, q_{dec})$-breaks the encryption scheme $\mathsf{E}_{\mathsf{PRG}}$ if he queries the encryption (decryption) oracle at most q_{enc} (q_{dec}) times, runs in time at most T, and guesses b with advantage at least ϵ.

Definition 6.3.3 *The composition* $\mathsf{E}_{\mathsf{PRG}}$ *of a public key encryption scheme* E *and a pseudorandom generator* PRG *is* $(T, \epsilon, q_{enc}, q_{dec})$*-secure if no adversary* $(T, \epsilon, q_{enc}, q_{dec})$*-breaks the scheme.*

Note that, compared to Definition 6.2.2, the above definition contains an extra parameter q_{enc}. One may wonder whether it makes sense to let the adversary access the encryption oracle. The adversary knows the public key so he can compute the encryptions himself. Note, however, that in our situation the random bits used by the encryption oracle and the random bits used to encrypt the challenge message m_b are from the same pseudorandom sequence r. Therefore, having access to the encryption oracle can, in principle, help the adversary to break the scheme.

The following theorem shows that if an encryption scheme and a pseudorandom generator are secure then their composition is also secure. We omit the proof since it is similar to the proof of Theorem 6.3.2.

Theorem 6.3.4 *If* E *is* $(T, \epsilon_{\mathsf{E}}, q_{dec})$*-secure,* PRG *is* $(T, \epsilon_{\mathsf{PRG}})$*-secure, and* $\epsilon_{\mathsf{E}} + \epsilon_{\mathsf{PRG}} < 1/2$ *then* $\mathsf{E}_{\mathsf{PRG}}$ *is* $(T, \epsilon_{\mathsf{E}} + \epsilon_{\mathsf{PRG}}, q_{enc}, q_{dec})$*-secure.*

6.4 Examples

In this section, we discuss a specific digital signature scheme, an encryption scheme and a pseudorandom generator. We compute parameters of the schemes and of the pseudorandom generator (the size of the secret key, the seed length) such that the compositions are secure against adversaries with $T = 2^{88}$ and $\epsilon = 0.01$.

6.4.1 Cramer-Shoup signature scheme

Consider the signature scheme proposed by Cramer and Shoup [CS00], which relies on the strong RSA assumption (see Section 2.3.3).

The scheme is parameterized by two positive integers l and l' such that $l+1 < l'$. A building block of the scheme is a collision-resistant hash function $H : \{0,1\}^* \mapsto \{0,1\}^l$ whose output can be interpreted as a positive integer less than 2^l (a hash function is said to be collision-resistant if it is infeasible to find two different inputs α, β such that $H(\alpha) = H(\beta)$). In the original paper [CS00] it is suggested to use SHA1.

The scheme includes three algorithms: key generation, signature generation, and signature verification.

Key generation. Two random l'-bit safe primes p and q are chosen, the modulus $N = pq$ is computed. Also chosen are a random $(l+1)$-bit prime e' and random $h, x \in QR_N$, where QR_N denotes the set of quadratic residues modulo N.

The public key is (N, h, x, e'). The private key is (p, q).

Signature generation. To sign an arbitrary bit string m, a random $(l+1)$-bit prime $e \neq e'$ is chosen, and a random $y' \in QR_N$ is chosen. The equation $y^e = xh^{H(x')} \mod N$ is solved for $y \in \mathbb{Z}_N^*$, where x' satisfies $(y')^{e'} = x'h^{H(m)} \mod N$. Note that y can be calculated using the factorization of N in the private key.

The signature is (e, y, y').

Signature Verification. To verify a signature (e, y, y') on a message m, it is first checked that e is an odd $(l+1)$-bit number, $e \neq e'$. Then, $x' = (y')^{e'}h^{-H(m)} \mod N$ is computed. Finally, it is checked whether $x = y^e h^{-H(x')} \mod N$.

Based on the security proof of Cramer and Shoup [CS00], we get the following proposition.

Proposition 6.4.1 *Suppose there exists an adversary that $(T, \epsilon_{\mathsf{S}}, q_{sig})$-breaks the above scheme. Then either a collision for the hash function H can be found in time $q_{sig}T$ or the flexible RSA problem with modulus N can be solved in time $q_{sig}T$ with probability ϵ_{S}.*

It follows from Theorem 6.3.2 that a signature scheme $\mathsf{S}_{\mathsf{PRG}}$ is secure against adversaries with $T = 2^{88}$ and $\epsilon = 0.01$ if the original scheme S is secure against adversaries with $T = 2^{88}$ and $\epsilon_{\mathsf{S}} = 0.005$ and the pseudorandom generator PRG is secure against adversaries with $T = 2^{88}$ and $\epsilon_{\mathsf{PRG}} = 0.005$ (the choice $\epsilon_{\mathsf{S}} = \epsilon_{\mathsf{PRG}}$ is made for the sake simplicity). We now use Proposition 6.4.1 to determine parameters l and n such that the Cramer-Shoup signature scheme is secure against adversaries with $T = 2^{88}$ and $\epsilon_{\mathsf{S}} = 0.005$.

Proposition 6.4.1 implies that under the strong RSA assumption (Assumption 2.3.4) the Cramer-Shoup signature scheme is $(T, \epsilon_{\mathsf{S}}, q_{sig})$-secure if

(i) no collision can be found for the hash function H in time $q_{sig}T$;

(ii) $q_{sig}T/\epsilon_{\mathsf{S}} \leq L(n)$, where n is the bit length of N, L-function is defined by Equation (2.2).

Let $q_{sig} = 2^{30}$ so the adversary is allowed to query the signing oracle up to one billion times (the value $q_{sig} = 2^{30}$ is also used, for instance, in [BR96; Cor02]). Assume that one application of the hash function H takes about 2^8 time units so $T = 2^{88}$ time units are approximately equal to 2^{80} applications of H (cf. [SKW+99]). Then, the block size l that satisfies Condition (i) is $l \gtrsim 2 \cdot (80 + 30) = 220$. For $\epsilon_{\mathsf{S}} = 0.005$, Condition (ii) is satisfied for $n \gtrsim 2700$.

Since $l \gtrsim 220$, the hash function SHA1 suggested by the designers of the scheme does not fit the above security level. Another hash function with a longer output has to be used, e.g., SHA224 with 224-bit output. The reason for the somewhat high requirements for l and n is that the security reduction in Proposition 6.4.1 is *not tight* meaning that a T-time adversary for the scheme implies $(q_{sig}T)$-time solver for the flexible RSA problem. If it were not for the factor of q_{sig} the requirements would be $l \gtrsim 160, n \gtrsim 1500$.

6.4.2 Cramer-Shoup encryption scheme

Now, we consider the Cramer-Shoup encryption scheme [CS98]. Security of this scheme relies on the DDH problem discussed in Section 2.4.2 and in Chapter 4.

Let $\mathbb{G}$ be a multiplicative group of order q.

The encryption scheme [CS98] uses a function $H : \{0,1\}^* \mapsto \{0,1\}^l$, $l > 0$, chosen at random from the family of universal one-way hash functions (a family of hash functions is said to be universal one-way if it is infeasible to choose an input x, draw a random hash function H, and then find a different y such that $H(x) = H(y)$). Similarly to the situation in the previous section, the output of the hash function is interpreted as a number between 0 and $2^l - 1$. Note that if a family of hash functions is collision-resistant then it is universal one-way. Therefore, in comparison with the signature scheme discussed above, in this case the requirement for the hash function is weaker.

The scheme includes three algorithms: key generation, encryption, and decryption.

Key generation. Random elements $g_1, g_2 \in \mathbb{G}$, $x_1, x_2, y_1, y_2, z \in \mathbb{Z}_q$ are chosen. The group elements $c = g_1^{x_1} g_2^{x_2}, d = g_1^{y_1} g_2^{y_2}, h = g_1^z$ are computed.

The public key is (g_1, g_2, c, d, h, H), and the private key is (x_1, x_2, y_1, y_2, z).

Encryption. To encrypt a message $m \in \mathbb{G}$, a random element $r \in \mathbb{Z}_q$ is chosen and the group elements $u_1 = g_1^r, u_2 = g_2^r, e = h^r m$ are computed. Then, $\alpha = H(u_1, u_2, e)$ and $v = c^r d^{r\alpha}$ are computed.

The ciphertext is (u_1, u_2, e, v).

Decryption. To decrypt a ciphertext (u_1, u_2, e, v), first $\alpha = H(u_1, u_2, e)$ is computed and the condition $u_1^{x_1 + \alpha y_1} u_2^{x_2 + \alpha y_2} = v$ is checked. If this condition does not hold, the decryption algorithm outputs “reject”; otherwise, it outputs $m = e/u_1^z$.

The following proposition is proved in [CS98].

Proposition 6.4.2 *Suppose there exists an adversary that $(T, \epsilon_{\mathsf{E}}, q_{dec})$-breaks the above encryption scheme. Then either the universal one-way function H can be broken in time T or the decisional Diffie-Hellman problem in group $\mathbb{G}$ can be solved in time T with probability ϵ_{E}.*

Note that the running time of the solver for the DDH problem does not depend on q_{dec}. As opposed to the Cramer-Shoup signature scheme discussed above, the Cramer-Shoup encryption scheme has a tight security reduction.

Similarly to the situation in the previous section, Theorem 6.3.4 implies that for an encryption scheme $\mathsf{E}_{\mathsf{PRG}}$ to be secure against adversaries with $T = 2^{88}$ and $\epsilon = 0.01$ it suffices to show that the original scheme E and the pseudorandom generator PRG are both secure against adversaries with $T = 2^{88}$ and $\epsilon_{\mathsf{E}} = \epsilon_{\mathsf{PRG}} = 0.005$. We now use Proposition 6.4.2 to determine parameters l, m, and n such that the Cramer-Shoup encryption scheme is secure against adversaries with $T = 2^{88}$ and $\epsilon_{\mathsf{E}} = 0.005$.

Assume, for instance, that $\mathbb{G}$ is a subgroup of $\mathbb{Z}_p^*$ of prime order q. Let m be the bit length of p and let n be the bit length of q. Then, it follows from Proposition 6.4.2 that under the DDH assumption (Assumption 2.4.4) the encryption scheme is $(2^{88}, 0.005, q_{dec})$-secure for all $q_{dec} > 0$ if $l \gtrsim 160$ and $2^{88}/0.005 < \min[L(m),\ 1.25 \cdot 2^{n/2}m^2/24]$. The latter condition holds for $m \gtrsim 1500, n \gtrsim 160$. Thus, the scheme is provably secure for relatively short parameters thanks to the tight security reduction.

6.4.3 Concrete security of the compositions

Finally, we consider the pseudorandom generator PRG_1 described in detail in Section 4.3.1. We compute the seed length of the generator such that the compositions of the pseudorandom generator with the two schemes discussed above reach the desired level of security.

Recall that the generator PRG_1 is shown to be (T, ϵ)-secure if Condition (4.1) is satisfied.

Composition of PRG_1 and the Cramer-Shoup signature scheme

Note that the Cramer-Shoup signature scheme is probabilistic. Let us count how many random bits are used by the signature scheme to produce $q_{sig} = 2^{30}$ signatures. To produce one signature, the scheme generates a random $(l+1)$-bit prime and a random element of QR_N (here we use the notation of Section 6.4.1). According to the prime number theorem, to generate a random $(l+1)$-bit prime number one has to try on average $\log_2(l+1)$ random $(l+1)$-bit numbers, which costs about $l \log_2 l$ random bits. In turn, generation of a random element from QR_N costs about n

random bits, where n denotes the bit length of N, as before. Thus the total cost of one signature is about $l \log_2 l + n$ bits. For $l = 220$, $n = 2700$, the total number of random bits required to produce 2^{30} signatures is of order 2^{42}.

Now suppose that the signature scheme uses generator PRG_1 to produce the random bits. For $M = 2^{42}$, $T = 2^{88}$, $\epsilon = 0.005$, Condition (4.1) implies that the generator reaches the security level for the seed length at least 2800.

Composition of PRG_1 and the Cramer-Shoup encryption scheme

Now, suppose that PRG_1 is used for the encryption scheme [CS98] to encrypt $q_{enc} = 2^{30}$ messages. Each encryption makes use of one random element of $\mathbb{Z}_q$ (see the notation of Section 6.4.2) so encrypting q_{enc} messages requires in total $M = q_{enc}n$ random bits. For $n = 160$, $M \simeq 2^{37}$.

For $M = 2^{37}$, $T = 2^{88}$, $\epsilon = 0.005$, Condition (4.1) implies that the generator reaches the security level for the seed length at least 2600. The secure seed length is slightly less in this case than the previous case because in this case less random bits have to be generated.

6.5 Conclusion

This chapter gives the concrete security analysis of probabilistic cryptographic schemes such as digital signature schemes and public key encryption schemes composed with pseudorandom generators.

The examples discussed in Section 6.4 illustrate how parameters of the compositions depend on the tightness of the security reductions. A scheme (a signature schemes or a public key encryption scheme) with a tighter security reduction is provably secure for smaller parameters so it requires less random bits which in turn implies that a pseudorandom generator with a shorter seed can be used.

CHAPTER 7

Generating random numbers from an interval

Nothing is random, only uncertain.

Gail Gasram

In this chapter we revisit the problem of generating uniformly random nonnegative integers below a certain bound b, given a source of random bits. We first consider the problem focussing on the randomness complexity, analyzing how many random bits are needed beyond the minimum of $n = \lceil \log_2 b \rceil$ bits. The folklore solutions require a non-constant number of bits beyond n. We propose a new algorithm with randomness complexity of $n+3$ bits which is only slightly larger than for Knuth-Yao's algorithm with optimal randomness complexity of $n+1$ bits. We fully analyze the randomness complexity of our algorithm, giving the expected number of random bits needed (and also the variance and the probability distribution for the number of random bits needed).

We then consider the problem in the setting of secure multiparty computation. Given a protocol for secure multiparty generation of random bits, the task is to securely generate a random integer in a certain interval. Our new algorithm leads to protocols for this task of essentially minimal computational complexity. Compared to Knuth-Yao's algorithm (and also the folkore algorithms), we achieve an improvement by at least a factor of 4 for the computational complexity. The round complexity of our protocols is favorable for small values of n, but asymptotically worse than for the other algorithms.

7.1 Introduction

A wide variety of cryptographic applications of randomness is discussed in Chapter 1. In particular, many cryptographic algorithms such as, for instance, the Diffie-Hellman key exchange protocol [DH76] and the ElGamal encryption scheme [Gam85] assume that the participants generate random numbers uniformly distributed in certain intervals.

Many public key cryptographic schemes critically depend on random numbers modulo a prime or an RSA modulus. As convincingly demonstrated by Bleichenbacher's attack on the generation of DSA one-time keys [Ble02], even the slightest deviation from uniform can be used to break such cryptographic schemes. The seemingly innocent bias introduced by reducing a random 160-bit string modulo a prime q of bit length 160 allows an attacker with access to a sufficient number of DSA signatures to recover the private key. The fix suggested by NIST [DSS00] is to reduce a random 320-bit string modulo q.

Note that most of the sources of randomness are designed to produce random bits rather than random numbers. Therefore, converting random bits to random numbers is an important problem. This problem is somewhat opposite to the one considered in Section 4.5 where the goal is to extract random bits from random numbers.

One of the essential characteristics of the algorithms for converting random bits to random numbers is their *randomness complexity*, which is the average amount of random bits used to produce one random number. If the length of the interval is an integral power of two, say, $b = 2^n$ one can simply concatenate n random bits, and convert these to a number. This simple algorithm has randomness complexity n. For $2^{n-1} < b \leq 2^n$, the above problem is solved conclusively by Knuth and Yao [KY76] who propose an algorithm with optimal randomness complexity. For the worst-case bound b, that is, $b = 2^{n-1} + 1$ Knuth-Yao's algorithm requires on average less than $n + 1$ random bits per output number.

In this chapter, we revisit the problem of converting random bits to random numbers. We summarize the existing algorithms for solving this problem and also present a new algorithm (see Sections 7.2 and 7.3). The new algorithm is fully analyzed in Section 7.4.

We remark that randomness complexity is not the only performance measure that plays a crucial role. In some situations, also other performance measures turn out to be important. One of such situations arises in the setting of *secure multiparty computation.* More precisely, we address the following problem. Several participants aim to generate a shared random number uniformly distributed in an interval $\{0, 1, \ldots, b - 1\}$, $b > 0$, in such a way that the random number can be reconstructed only if a certain number of the participants open their shares. The algorithms that convert random bits to random numbers can be used to design protocols for solving this problem.

The main performance measures of protocols in the setting of secure multiparty computation are computational complexity, round complexity, and communication complexity. Computational complexity is indicated by the size of the circuit to be computed by the participants while round complexity is indicated by the depth of the circuit. Communication complexity characterizes the amount of communication between the participants. Usually, communication complexity is strongly related to the computational complexity. In Section 7.6, we discuss the protocols for generating shared random numbers and analyze their computational complexity and round complexity. We show that the protocol induced by our new algorithm has *essentially minimal* computational complexity.

7.2 State of the art

In this section, we summarize the existing algorithms for converting random bits to random numbers from an interval $\{0, 1, \ldots, b-1\}$, where $2^{n-1} < b \leq 2^n$, $n > 0$.

7.2.1 Folklore algorithms

First, we recall two folklore algorithms, which are in widespread use and have been described in many places (see, e.g., [Sho05] for a recent treatment). For a nonnegative integer x, $\ell_i(x)$ denotes the i-th least significant bit of x, as before.

Algorithm 7.2.1 Generate-and-compare algorithm

```
Input: Bound $b > 0$, source of random bits
Output: Uniformly distributed random number between 0 and $b-1$
  $n \leftarrow \lceil \log_2 b \rceil$
  repeat
    select $\{\ell_1(m), \ell_2(m), \ldots, \ell_n(m)\} \in_R \{0,1\}^n$
  until $m < b$
  return $m$
```

Algorithm 7.2.2 Generate-and-reduce algorithm

```
Input: Bound $b > 0$, source of random bits, security parameter $k > 0$
Output: Random number between 0 and $b-1$ with statistical distance $2^{-k}$ from
  uniform
  $n \leftarrow \lceil \log_2 b \rceil$
  $\{\ell_1(m), \ell_2(m), \ldots, \ell_{n+k}(m)\} \in_R \{0,1\}^{n+k}$
  $m \leftarrow m \bmod b$
  return $m$
```

Algorithm 7.2.1 is a Las Vegas algorithm meaning that its output is perfectly uniform on $\{0, 1, \ldots, b-1\}$ but it may run indefinitely. On the other hand, the running time of Algorithm 7.2.2 is constant but the statistical distance from uniform is as large as 2^{-k}, where k is the security parameter. An algorithm combining constant running time and perfect uniform output does not exist if $2^{n-1} < b \leq 2^n$, see, e.g., [KY76]: suppose the algorithm uses exactly t random bits (using any unused random bits before terminating), then there are 2^t equally likely execution paths, but it is impossible to partition these paths into b equally large groups.

The folklore algorithms are commonly presented without paying much attention to the randomness complexity. The number of loop iterations for Algorithm 7.2.1

is $2^n/b$ on average, which is almost equal to 2 in the worst case when $b = 2^{n-1} + 1$. Hence, Algorithm 7.2.1 uses about $2n$ random bits, which means an overhead of n bits, whereas Algorithm 7.2.2 always uses $n + k$ random bits, for an overhead of k bits.

Algorithms 7.2.1 and 7.2.2 are online algorithms in the sense that these algorithms generate exactly as many random numbers as required by the application. A folklore offline algorithm generates a batch of random numbers in $\{0, 1, \ldots, b-1\}$ in one go, by searching for a power of b that is only slightly smaller than a power of 2, say $b^u \lesssim 2^v$. The disadvantage of such an offline method is that u numbers have to be generated at the same time (and stored for later use), where u may be prohibitively large. In this chapter, we focus on online algorithms only.

7.2.2 Knuth-Yao's algorithm

As mentioned above, the randomness complexity of the folklore algorithms is far from satisfactory. Knuth and Yao [KY76] propose an algorithm with *optimal* randomness complexity, meaning that the average amount of random bits required to produce a single random number is as small as possible.

Knuth-Yao's algorithm can be represented as a binary tree, which is cut as follows. For $i = 1, 2, \ldots,$ if the total number of nodes of depth i is at least b then the tree is cut in such a way that exactly b nodes become leaf nodes, which can be indexed by $\{0, 1, \ldots, b-1\}$. The initial state of the algorithm is the root node of the tree. At each step a random bit is generated and the current state is set to be one of the children nodes depending on the value of the bit. If at some point the current state is a leaf node, the algorithm halts and outputs the index of the leaf node.

Figure 7.1 illustrates Knuth-Yao's algorithm for $b = 13$. The root node is the one in the left bottom corner. The squares are the leaf nodes; the values inside the squares are the outputs of the algorithm. The formal version of Figure 7.1 is Algorithm 7.2.3.

Knuth and Yao prove that the randomness complexity of Algorithm 7.2.3 is $b\nu(1/b)$, where

$$\nu(a) = \sum_{m \geq 0} \frac{2^m a - \lfloor 2^m a \rfloor}{2^m},$$

for $0 \leq a \leq 1$ [KY76]. It can be shown that $n < b\nu(1/b) < n+1$, where $n = \lceil \log_2 b \rceil$. For $b = 2^{n-1} + 1$, the randomness complexity is very close to $n + 1$.

Strictly speaking, Knuth and Yao [KY76] propose not one algorithm but a family of algorithms in the sense that they do not specify the way the leaf nodes are chosen and indexed. For instance, Figure 7.2 represents another version of Knuth-Yao's algorithm. As opposed to the version shown in Figure 7.1, the leaf nodes in Figure 7.2 are indexed from bottom to top and the infinite loop corresponds to all-zeros input.

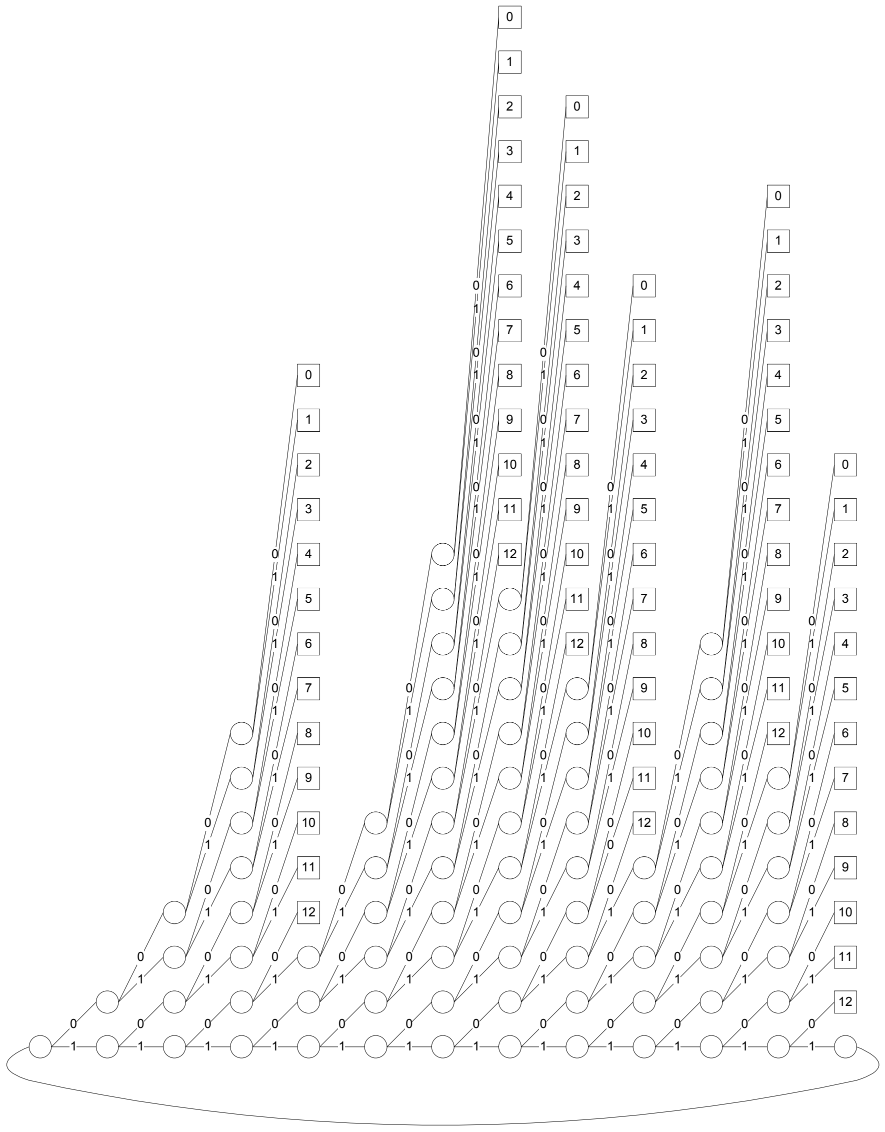

Figure 7.1: The version of Knuth-Yao's algorithm that corresponds to Algorithm 7.2.3 for $b = 13$. The algorithm starts at the left bottom corner and ends with one of the values between 0 and 12 depending on the random bits.

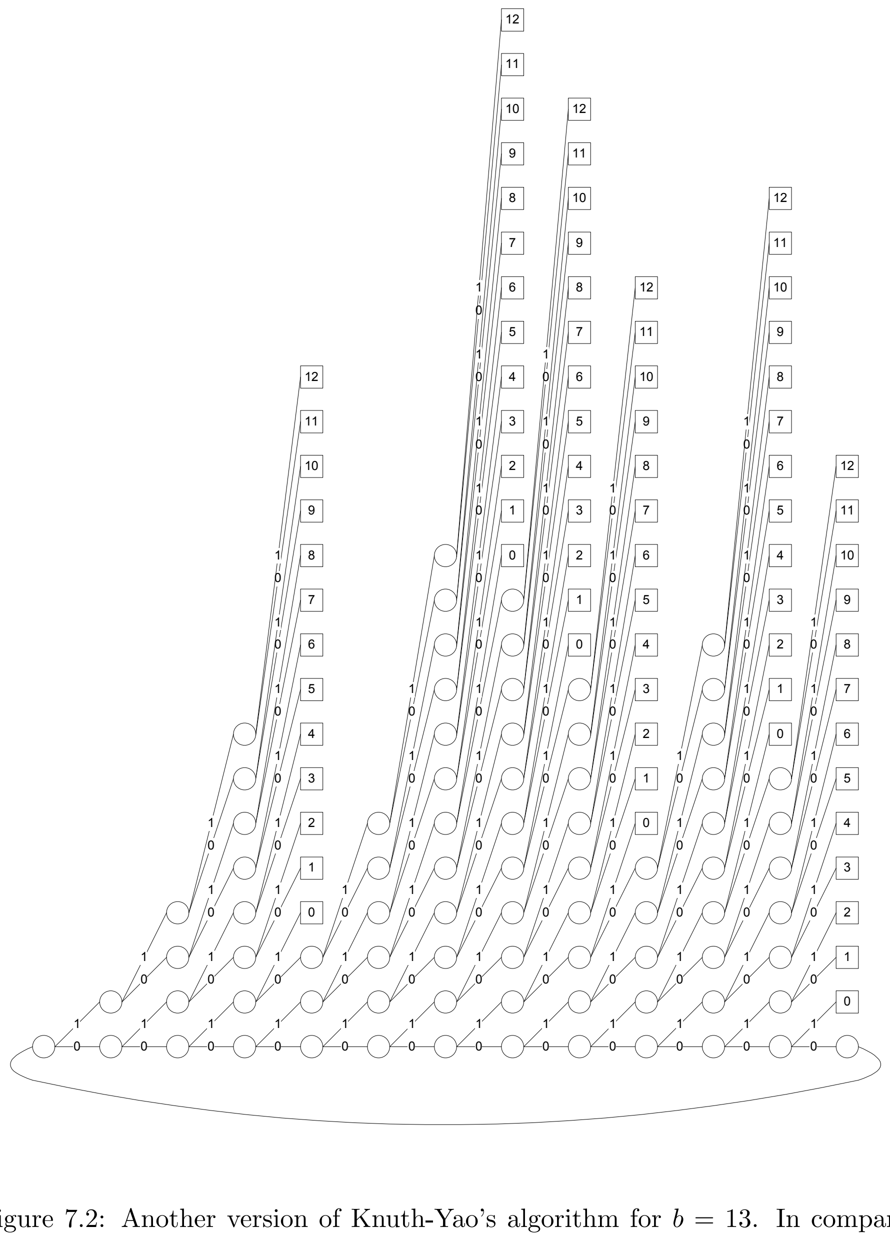

Figure 7.2: Another version of Knuth-Yao's algorithm for $b = 13$. In comparison with the version shown in Figure 7.1, the leaf nodes are indexed from bottom to top and the infinite loop corresponds to all-zeros input.

Algorithm 7.2.3 Knuth-Yao's algorithm

Input: Bound b such that $2^{n-1} < b \leq 2^n$, source of random bits
Output: Uniformly distributed random number between 0 and $b-1$

```
$m \leftarrow 0$
$i \leftarrow 1$
loop
  $s \in_R \{0,1\}$
  $m \leftarrow 2m + s$
  $h \leftarrow 2^i \bmod 2b$
  if $h \geq b$ then
    if $m \leq b$ then
      return $m$
    else
      $m \leftarrow m - b$
    end if
  end if
  $i \leftarrow i + 1$
end loop
```

7.3 Bit-by-bit algorithm

In this section, we present a new algorithm for generating random numbers uniformly distributed over an interval $\{0, \ldots, b-1\}$. The average randomness complexity of the new algorithm is at most $n+3$ bits, which is just slightly higher than the randomness complexity of Algorithm 7.2.3.

Let $2^{n-1} < b \leq 2^n$. Let $b'_i = \ell_i(b-1)$, $i = 1, 2, \ldots, n$. The bit-by-bit algorithm (Algorithm 7.3.1) generates a random number m from the interval $\{0, 1, \ldots, b-1\}$ starting from the most significant bit. For $i = n, n-1, \ldots$, the bit $\ell_i(m)$ is generated and compared with b'_i. Assume that $\ell_j(m) = b'_j$ for all j such that $i < j \leq n$. Then, if $\ell_i(m) > b'_i$ the algorithm restarts by setting $i = n$. If $\ell_i(m) < b'_i$ then $m < b$ regardless of the remaining bits of m, so these bits can be chosen at random with no further checking.

In Algorithm 7.3.1, bit r is used as a flag indicating whether the current bit $\ell_i(m)$ has to be compared with b'_i or not. Whenever r becomes equal to 1 we do not have to check the remaining bits.

Every time index i is set to n we say that a new run starts. We call a run *successful* if it finishes with outputting a value of m, otherwise we call it *failing*. The overall process may include a few failing runs and always ends with a successful run. The length of the run is the number of random bits generated during the run. The length of a successful run is always n. If $b = 2^n$, there are no failing runs at all.

Similar to Algorithm 7.2.1, the probability that a certain run is successful is $p = b/2^n$. Moreover, conditioning on the latter event, one sees that m is uniformly distributed over $\{0, 1, \ldots, b-1\}$, as required.

Algorithm 7.3.1 Bit-by-bit algorithm

```
Input: Bound $b > 0$, source of random bits
Output: Uniformly distributed random number between 0 and $b - 1$
  $n \leftarrow \lceil \log_2 b \rceil$
  $r \leftarrow 0$;  $i \leftarrow n$
  while $i > 0$ do
    $\ell_i(m) \in_R \{0, 1\}$
    if $r = 1$ then
      $i \leftarrow i - 1$
    else if $\ell_i(m) > b'_i$ then
      $i \leftarrow n$
    else if $\ell_i(m) < b'_i$ then
      $r \leftarrow 1$;  $i \leftarrow i - 1$
    else
      $i \leftarrow i - 1$
    end if
  end while
  return $m$
```

The bit-by-bit algorithm can be implemented recursively, see Algorithm 7.3.2.

Algorithm 7.3.2 The recursive version of the bit-by-bit algorithm

```
Input: Bound $b > 0$, source of random bits
Output: Uniformly distributed random number between 0 and $b - 1$
  if $b = 1$ then
    return 0
  else
    repeat
      $m \leftarrow$ Algorithm 7.3.2 on input $(b + (b \bmod 2))/2$
      $s \in_R \{0, 1\}$
      $m \leftarrow 2m + s$
    until $m < b$
    return $m$
  end if
```

Algorithm 7.3.2 is indeed the same as Algorithm 7.3.1 since Algorithm 7.3.2 also generates the random number starting from its most significant bit (the least significant bit is the last to be generated) and restarts as soon as it detects that the random number exceeds $b - 1$.

7.4 Analysis of the bit-by-bit algorithm

Throughout the analysis, let $s = \{s_1, s_2, \dots\}$ denote the sequence of all random bits used by Algorithm 7.3.1 to generate a single random number m. We will condition on the event f_i, $1 \le i \le n$, defined as the event that s starts with a failing run of length i. Then f_i happens if and only if s starts with i bits $s_1, s_2, \dots, s_i$ such that $s_1 = b'_n$, $s_2 = b'_{n-1}$, $\dots$, $s_{i-1} = b'_{n-i+2}$, $s_i = 1$ and $b'_{n-i+1} = 0$. And by f_0 we denote the event that s does not contain any failing runs and thus the total length of s is n. The probabilities for these events are: $\Pr(f_0) = p$ and $\Pr(f_i) = (1 - b'_{n-i+1})/2^i$, for $1 \le i \le n$, where

$$p = 1 - \sum_{i=1}^{n} \frac{1 - b'_{n-i+1}}{2^i} = b/2^n. \tag{7.1}$$

Note that $\Pr(f_1) = 0$, since $b'_n = 1$.

Let X denote the length of s so X represents the number of random bits used by Algorithm 7.3.1 to generate a single random integer uniformly distributed over an interval $\{0, 1, \dots, b-1\}$, $2^{n-1} < b \le 2^n$. Our goal is to determine the expected value, the variance, and the probability distribution of X.

7.4.1 Expected value

Theorem 7.4.1 *The expected value of the number of random bits X used by Algorithm 7.3.1 to generate a single integer uniformly distributed over the interval $\{0, \dots, b-1\}$, $2^{n-1} < b \le 2^n$, is given by*

$$\mathsf{E}(X) = n + p^{-1} \sum_{i=2}^{n} \frac{(1 - b'_{n-i+1})i}{2^i}, \tag{7.2}$$

where $p = b/2^n$, b'_i denotes the i-th least significant bit of $b-1$, $i = 1, 2, \dots, n$.

Proof: We evaluate the expected value conditioned on f_i:

$$\mathsf{E}(X) = \sum_{i=0}^{n} \mathsf{E}(X|f_i) \Pr(f_i).$$

Clearly, $\mathsf{E}(X|f_0) = n$. And, $\mathsf{E}(X|f_i) = i + \mathsf{E}(X)$ for $0 < i \le n$, since after a failing run the state of the algorithm is the same as its initial state. Thus,

$$\mathsf{E}(X) = np + \sum_{i=2}^{n} (i + \mathsf{E}(X)) \frac{1 - b'_{n-i+1}}{2^i},$$

which implies that

$$\mathsf{E}(X) \left(1 - \sum_{i=2}^{n} \frac{1 - b'_{n-i+1}}{2^i} \right) = np + \sum_{i=2}^{n} \frac{(1 - b'_{n-i+1})i}{2^i}.$$

Since $b'_n = 1$ and using (7.1), the theorem follows. □

Theorem 7.4.1 implies that the expected value $\mathsf{E}(X)$ is maximal if $b'_i = 0$, for $0 < i < n$, that is, $b = 2^{n-1} + 1$. Indeed, in this case the sum on the right-hand side in (7.2) is maximal and p is minimal. So, the worst-case bound b of length n is $b = 2^{n-1} + 1$, for which Theorem 7.4.1 gives

$$\mathsf{E}_{\max}(X) = n + \frac{2^n}{2^{n-1}+1} \sum_{i=2}^{n} \frac{i}{2^i} = n + 3 - \frac{n+5}{2^{n-1}+1}. \tag{7.3}$$

Corollary 7.4.2 *The expected value of the number of random bits X used by Algorithm 7.3.1 to generate a single integer uniformly distributed over an interval $\{0, 1, \ldots, b-1\}$ is less than $n+3$ for all n and b such that $2^{n-1} < b \leq 2^n$. If $b = 2^{n-1} + 1$, then $\mathsf{E}(X) \to n + 3$ as $n \to \infty$.*

Further (numerical) analysis shows that show that the typical number of bits x used by Algorithm 7.3.1 to generate a single random integer is even smaller than $n+3$: namely, about $n+1.13$ bits when averaged over all possible bounds b, $2^{n-1} < b \leq 2^n$.

Algorithm	Randomness complexity
Generate-and-compare (7.2.1)	$2n$
Generate-and-reduce (7.2.2)	$n+k$
Knuth-Yao (7.2.3)	$n+1$
Bit-by-bit (7.3.1)	$n+3$

Table 7.1: Average randomness complexity of the algorithms that generate random integers from an interval $\{0, 1, \ldots, b-1\}$ for the worst-case bound b, i.e. $b = 2^{n-1}+1$. Here k is the security parameter of Algorithm 7.2.2.

7.4.2 Variance

Theorem 7.4.3 *The variance of the number of random bits X used by Algorithm 7.3.1 to generate a single integer uniformly distributed over the interval $\{0, \ldots, b-1\}$, $2^{n-1} < b \leq 2^n$, satisfies the following equation:*

$$\mathsf{Var}(X) = (\mathsf{E}(X) - n)^2 + p^{-1} \sum_{i=2}^{n} \frac{i^2(1 - b'_{n-i+1})}{2^i},$$

where $p = b/2^n$, b'_i denotes the i-th least significant bit of $b-1$, $i = 1, \ldots, n$, $\mathsf{E}(X)$ is the expected value of X.

Proof: By definition, $\mathsf{Var}(X) = \mathsf{E}((X - \mu)^2)$, where μ denotes $\mathsf{E}(X)$. Recall that

f_i denotes the event that s starts with a failing run of length i, $0 \le i \le n$. Thus,

$$\mathsf{Var}(X) = \sum_{i=0}^{n} \mathsf{E}((X-\mu)^2 | f_i) \Pr(f_i),$$

Note that $\mathsf{E}((X-\mu)^2|f_0) = (\mu - n)^2$, and $\mathsf{E}((X-\mu)^2|f_i) = \mathsf{E}((i+X-\mu)^2) = i^2 + \mathsf{Var}(X)$, for $1 \le i \le n$. Since $\Pr(f_0) = b/2^n$, $\Pr(f_1) = 0$, and $\Pr(f_i) = (1-b'_{n-i+1})/2^i$, $2 \le i \le n$, we have

$$\mathsf{Var}(X) = (\mu - n)^2 p + \sum_{i=2}^{n} \frac{(i^2 + \mathsf{Var}(X))(1-b'_{n-i+1})}{2^i},$$

which implies, using Equation (7.1),

$$\mathsf{Var}(X) = (\mu - n)^2 + p^{-1} \sum_{i=2}^{n} \frac{i^2(1-b'_{n-i+1})}{2^i}.$$

□

As a conclusion, Theorems 7.4.1 and 7.4.3 yield that $\mathsf{E}(X) = n + p^{-1}a_1$ and $\mathsf{Var}(X) = (\mathsf{E}(X) - n)^2 + p^{-1}a_2$, where

$$a_t = \sum_{i=2}^{n} \frac{i^t(1-b'_{n-i+1})}{2^i},\ t \ge 0.$$

Note that $p = 1 - a_0$.

We see that also the variance $\mathsf{Var}(X)$ is maximal for $b = 2^{n-1}+1$: the term $(\mathsf{E}(X)-n)^2$ is maximal and the term $p^{-1}a_2$ is also maximal, since a_2 is maximal while p is minimal. Theorem 7.4.3 thus gives

$$\mathsf{Var}_{\max}(X) = \frac{2(5 \cdot 2^{2n+1} - 2^n n^2 - 5 \cdot 2^{n+1} n - 7 \cdot 2^n - 4)}{(2^n+2)^2}.$$

Note that $\mathsf{Var}_{\max}(X) \to 20$ as $n \to \infty$.

7.4.3 Probability Distribution

Next, we determine the probability distribution of the random variable X that represents the number of random bits used by Algorithm 7.3.1 to generate a single random integer. Define $p_i = \Pr[X = i]$, $i \ge 0$.

Theorem 7.4.4 *Let $2^{n-1} < b \le 2^n$ and let b'_i be the i-th least significant bit of $b-1$, $i = 1, \ldots, n$. Then $p_0 = \ldots = p_{n-1} = 0$, $p_n = p$,*

$$p_j = \sum_{i=2}^{n} \frac{(1-b'_{n-i+1})p_{j-i}}{2^i},\ j > n,$$

where $p = b/2^n$.

Proof: For $0 \le j \le n$, the result is obvious. Assume that $j > n$. Recall that f_i denotes the event that s starts with a failing run of length i, $0 \le i \le n$. Then,

$$p_j = \sum_{i=0}^{n} \Pr[X = j | f_i] \Pr(f_i).$$

Clearly, $\Pr[X = j | f_0] = 0$. Note that $\Pr[X = j | f_i] = p_{j-i}$, $i = 1, \ldots, n$. Again using $\Pr(f_0) = p$, $\Pr(f_1) = 0$, and $\Pr(f_i) = (1 - b'_{n-i+1})/2^i$, $2 \le i \le n$, we get

$$p_j = \sum_{i=2}^{n} \frac{(1 - b'_{n-i+1}) p_{j-i}}{2^i}.$$

□

It follows from Theorem 7.4.4 that $p_j = 2^{n-j} q_j p$, $j \ge 0$, where $q_0 = \ldots = q_{n-1} = 0$, $q_n = 1$, $q_j = (1 - b'_{n-1}) q_{j-2} + \ldots + (1 - b'_1) q_{j-n}$, $j > n$. The following lemma gives an upper bound for q_j, $j > n + 1$.

Lemma 7.4.5 *For $j > n + 1$, we have $q_j \le \mathfrak{F}_{j-n-1}$, where $\mathfrak{F}_i$ denotes the i-th Fibonacci number. The equality holds if and only if $b = 2^{n-1} + 1$ and $n + 1 < j \le 2n$.*

Proof: The Fibonacci numbers $\mathfrak{F}_k$, $k \ge 1$, can be defined as follows. $\mathfrak{F}_1 = 1$, $\mathfrak{F}_2 = 1$,

$$\mathfrak{F}_k = \sum_{i=1}^{k-2} \mathfrak{F}_i + 1, \ k > 2. \tag{7.4}$$

1. Let $n + 1 < j \le 2n$. Clearly, $q_{n+2} = \mathfrak{F}_1$, $q_{n+3} = \mathfrak{F}_2$. Observe that for $n + 3 < j \le 2n$, $q_j \le q_{j-2} + \ldots + q_{j-n} = q_{j-2} + \ldots + q_{n+3} + q_{n+2} + 1$. (7.4) implies that $q_j \le \mathfrak{F}_{n-j-1}$. Note that for $b = 2^{n-1} + 1$, we have $b'_i = 0$, $1 \le i < n$. Thus, $q_j = q_{j-2} + \ldots + q_{n+3} + q_{n+2} + 1$, $n + 3 < j \le 2n$, so $q_j = \mathfrak{F}_{n-j-1}$.

2. Now let $j > 2n$. In this case, $q_j \le q_{j-2} + \ldots + q_{j-n} < q_{j-2} + \ldots + q_{j-n} + q_{j-n-1} + \ldots + q_{n+3} + q_{n+2} + 1 = \mathfrak{F}_{j-n-1}$.

□

Lemma 7.4.5 implies that for $j > n + 1$

$$p_j \le \frac{1}{\sqrt{5}} \left(\frac{1 + \sqrt{5}}{4} \right)^{j-n-1},$$

so the sequence p_j decreases exponentially as j increases.

Finally, we determine the tail probabilities $r_j = \Pr[X \le j]$ in a similar way as the probability distribution p_j, $j \ge 0$.

Theorem 7.4.6 *Let $2^{n-1} < b \leq 2^n$ and let b'_i be the i-th least significant bit of $b-1$, $i = 1, \ldots, n$. Then $r_0 = \ldots = r_{n-1} = 0$, $r_n = p$,*

$$r_j = p + \sum_{i=2}^{n} \frac{(1 - b'_{n-i+1}) r_{j-i}}{2^i}, \; j > n,$$

where $p = b/2^n$.

Proof: For $j \leq n$ the result is obvious. Assume that $j > n$. Recall that f_i denotes the event that s starts with a failing run of length i, $0 \leq i \leq n$. Then,

$$r_j = \sum_{i=0}^{n} \Pr[X \leq j | f_i] \Pr(f_i).$$

Clearly, $\Pr[X \leq j | f_0] = 1$. Note that $\Pr[X \leq j | f_i] = r_{j-i}$, for $1 \leq i \leq n$. On the other hand, $\Pr(f_0) = p$, $\Pr(f_1) = 0$, $\Pr(f_i) = (1 - b'_{n-i+1})/2^i$, $2 \leq i \leq n$, from which the result follows. □

Let $b = 2^{n-1} + 1$. Then $r_j = 2^{n-j} s_j p$, where $s_0 = \ldots = s_{n-1} = 0$, $s_n = 1$, $s_j = s_{j-2} + \ldots + s_{j-n} + 2^{j-n}$, $j > n$. Theorem 7.4.6 implies that if, say, $n = 1000$ the probability that the algorithm needs more than 1020 bits is less than 0.01. In general, it follows that $1 - r_j$ decreases exponentially as a function of j (with rate between $1/2$ and $(1 + \sqrt{5})/4$).

7.5 Oblivious algorithms

Note that the execution paths of Algorithms 7.2.1, 7.2.2, and 7.2.3 do not depend on the value of the output m. In other words, the execution paths of these algorithms do not reveal any information about m. For instance, the execution path of the generate-and-compare algorithm only reveals how many n-bit strings have been used to generate m. In this sense, Algorithms 7.2.1, 7.2.2, and 7.2.3 are *oblivious*.

On the other hand, Algorithm 7.3.1 is *not* oblivious since the execution path of this algorithm reveals some information on m. For example, if the case $r = 1$ applies in the if-then-else statement it follows that $m \neq b - 1$.

To construct an oblivious version of Algorithm 7.3.1, it suffices to hide the value of r. The value of index i need not be hidden as the algorithm will always end in a successful run, with i starting at $i = n$ and ending at $i = 0$. What happens during the failing runs does not matter because after each failing run the algorithm returns to its initial state ($i = n$, $r = 0$).

An oblivious version of Algorithm 7.3.1 is Algorithm 7.5.1. To update the value of i at the end of each iteration we need to know the value of $(1 - r)\ell_i(m)(1 - b'_i)$. Since we do not need to hide the value of i, it follows that we do not need to hide this value either.

Algorithm 7.5.1 Oblivious bit-by-bit algorithm

```
Input: Bound $b > 0$, source of random bits
Output: Uniformly distributed random number between 0 and $b - 1$
  $n \leftarrow \lceil \log_2 b \rceil$
  $r \leftarrow 0$; $i \leftarrow n$
  while $i > 0$ do
    $\ell_i(m) \in_R \{0, 1\}$
    $r \leftarrow r + (1 - r)(1 - \ell_i(m))b'_i$
    if $(1 - r)\ell_i(m)(1 - b'_i) = 1$ then
      $i \leftarrow n$
    else
      $i \leftarrow i - 1$
    end if
  end while
  return $m$
```

7.6 Generating shared random numbers

Consider the following problem. Several participants aim to generate a shared random number uniformly distributed in an interval $\{0, 1, \ldots, b-1\}$, $b > 0$. Our goal is to design protocols for solving this problem and to analyze their computational complexity and round complexity.

Below, we analyze the protocols for secure multiparty generation of bounded integers assuming a setting based on verifiable secret sharing (see [GMW87] and subsequent work). A value (e.g., from a finite field) is said to be in "shared form" when each participant holds a share of the value (e.g., as in Shamir's threshold scheme). Addition of shared values is often for free due to the linearity of the underlying secret scheme. For multiplication and other more involved operations, special protocols exist that take one or more shared values as input and produce one or more (new) shared values as output.

However, our analysis does not critically rely on the use of verifiable secret sharing. The closely related setting of secure multiparty computation based on threshold homomorphic cryptosystems (see [FH96; CDN01] for details) may be used as well. In that case, a value is said to be in "shared form" when the value is encrypted under the public key of a threshold homomorphic cryptosystem. The corresponding private key is shared among the parties, of which a certain fraction needs to cooperate for successful decryption. The homomorphic property of the cryptosystem ensures that addition of shared values is for free, whereas special protocols take care of multiplication and further operations on shared values.

7.6.1 The protocols

An obvious way to generate a shared random number is as follows. Each participant generates a random number between 0 and $b-1$, then the sum of the numbers is computed in shared form, and finally the sum is reduced modulo b. We refer to this protocol as the *naive approach*. In Section 7.6.2, we show that the naive approach is quite inefficient in terms of computational complexity and round complexity.

An alternative to the naive approach is to use the protocols induced by Algorithms 7.2.1, 7.2.2, 7.2.3, and 7.5.1. To transform the above-mentioned oblivious algorithms into protocols for generating shared random numbers, the random bits have to be generated securely, in shared form, using a multiparty protocol. Besides that, the value of the output m, its intermediate values, and the value of r in Algorithm 7.5.1 have to be computed securely, in shared form.

Throughout our analysis, we do not count the complexity of generating shared random bits. We only count the additional complexity. The shared random bits can be generated, for instance, using the protocol RAN_2 proposed by Damgård et al. [DFK$^+$06] which uses 2 rounds and 2 secure multiplication gates. However, RAN_2 may have limited applicability.

7.6.2 Complexity of the protocols

In this section, we analyze the computational complexity and the round complexity of the protocols for generating shared random numbers. The basic building blocks of these protocols are protocols for addition, subtraction, comparison, and modular reduction of bitwise-shared integers (by a bitwise-shared integer we mean that each bit of the integer is shared).

Without loss of generality, we let $b = 2^{n-1} + 1$ for some $n > 1$. For simplicity, b is assumed to be publicly known.

Let t be the number of participants who wish to generate a shared random number. In the *naive approach*, the sum of t n-bit numbers is computed in shared form and then the sum is securely reduced modulo an n-bit number. In our complexity analysis, we assume that the sum is computed using a tree structure of depth $\log_2 t$.

The *generate-and-compare protocol* requires on average about 2 secure comparisons of bitwise-shared integers.

The bottleneck of the *generate-and-reduce protocol* is secure reduction of an $(n + k)$-bit integer modulo an n-bit integer.

Knuth and Yao [KY76] prove that before the random number m is generated, the inner loop of Algorithm 7.2.3 is repeated on average $b\nu(1/b) \simeq n + 1$ times, for $i = 1, 2, \ldots, n+1$. Observe that for $i = 1, \ldots, n-1$, we have $h < b$ so comparison of m and b is done on average 2 times (for $i = n$ and $i = n+1$). Subtraction $m \leftarrow m-b$ is done on average only once. Therefore *Knuth-Yao's protocol* requires on average 2 comparisons and 1 subtraction of bitwise-shared integers.

Each step of the *bit-by-bit protocol* requires a secure multiplication of r and $\ell_i(m)$. Besides that, at each step the bit $(1 - r)\ell_i(m)(1 - b'_i)$ has to be revealed. Since the total number of steps of the protocol is on average $n + 3$, the protocol costs $n + 3$ secure multiplications and $n + 3$ reveals.

The computational complexity and the round complexity of the above protocols depends on how addition, comparison, and modular reduction of the bitwise-shared integers are implemented (subtraction is usually implemented in the same way as addition). There exist several protocols for these tasks (e.g., [ST06; GSV07; DFK$^+$06]). Some of them have relatively low computational complexity but linear in n round complexity [ST06]. Others have logarithmic or even constant round complexity but relatively high computational complexity [GSV07; DFK$^+$06]. We first focus on protocols with low computational complexity. Then we consider protocols with low round complexity.

Computational complexity

Assume that addition and comparison of bitwise-shared integers are implemented using the protocol described by Schoenmakers and Tuyls [ST06, Section 2.3] which uses l rounds and $2l$ secure multiplication gates, where l is the bit length of the integers. To the best of our knowledge, this protocol is the most efficient in terms of computational complexity. Assume also that secure modular reduction is implemented using the long division method. Then, the complexity of the naive approach is about $n \log_2 t + 2n \log_2 t$ rounds and $2nt + 4n \log_2 t$ secure multiplication gates. Table 7.2 shows that the naive approach along with the generate-and-reduce protocol have the worst computational complexity.

The bit-by-bit protocol has essentially minimal computational complexity, which is only about 1 secure multiplication and 1 reveal per bit (at each step of the protocol, the product of $(1-r)$ and $\ell_i(m)$ is computed and then the value of $(1-r)\ell_i(m)(1-b'_i)$ is revealed).

Protocol	Computational complexity	Round complexity, rounds
Naive approach	$2n(t + 2\log_2 t)$ mult's	$3n \log_2 t$
Generate-and-compare	$4n$ mult's	$2n$
Generate-and-reduce	$4nk$ mult's	$2nk$
Knuth-Yao's	$6n$ mult's	$3n$
Bit-by-bit	$(n+3)$ mult's + $(n+3)$ reveals	$n + 3$

Table 7.2: The lowest possible computational complexity and the corresponding round complexity of the protocols for generating shared random numbers (excluding the complexity of generating shared random bits). The security parameter of the generate-and-reduce protocol is denoted by k; the number of participants is denoted by t.

Round complexity

Constant-round protocols for addition, comparison, and modular reduction of bitwise-shared integers are proposed by Damgård et al. [DFK$^+$06]. As reported by Nishide and Ohta [NO07], the complexity of the addition protocol is 15 rounds and $47l \log_2 l$ secure multiplication gates, where l is the bit length of the integers. The complexity of the comparison protocol is 7 rounds and $17l$ secure multiplication gates. Reducing an l_1-bit integer modulo an l_2-bit integer implies l_1 subtractions and l_1 comparisons of bitwise-shared integers (the subtractions and the comparisons can be done in parallel, cf. [DFK$^+$06]). Applying these protocols makes the round complexity of the generate-and-compare protocol, generate-and-reduce protocol, and Knuth-Yao's protocol constant (see Table 7.3).

In some cases, logarithmic-round protocols for addition and comparison of bitwise-shared integers are more efficient than the constant-round protocols. For instance, Garay et al. [GSV07] propose a comparison protocol that uses $\log_2 l$ rounds and $3l$ secure multiplication gates. A logarithmic-round addition protocol can be derived, for example, from the Brent-Kung circuit (see, e.g., [Par00, Chapter 6]). The complexity of the latter protocol is about $2 \log_2 l$ rounds and $4l$ secure multiplication gates.

Note that the bit-by-bit protocol does not imply neither addition nor comparisons of bitwise-shared integers. Therefore, the complexity of the bit-by bit protocol in Table 7.3 is the same as the one in Table 7.2.

Protocol	Computational complexity	Round complexity, rounds
Naive approach	$47n \log_2 n(t+n) + 17n^2$ mult's	$15 \log t + 22$
	$n(3t+7n)$ mult's	$\log_2 n(2 \log_2 t + 3)$
Generate-and-compare	$34n$ mult's	14
	$6n$ mult's	$2 \log_2 n$
Generate-and-reduce	$(47n \log_2 n + 17n)(n+k)$ mult's	22
	$7n(n+k)$ mult's	$3 \log_2 n$
Knuth-Yao's	$47n \log_2 n + 34n$ mult's	29
	$10n$ mult's	$4 \log_2 n$
Bit-by-bit	$(n+3)$ mult's + $(n+3)$ reveals	$n+3$

Table 7.3: Versions of the protocols for generating shared random numbers with reduced round complexity (the complexity of generating shared random bits is not included). As before, the security parameter of the generate-and-reduce protocol is denoted by k; the number of participants is denoted by t.

7.7 Conclusion

The choice of the protocol for generating random numbers in the setting of secure multiparty computation depends on the designer's goal. If one aims at minimizing the round complexity, the generate-and-compare protocol is a good choice. In turn, the bit-by-bit protocol has essentially minimal computational complexity. Since communication complexity is usually strongly related to computational complexity the bit-by-bit protocol is very efficient also in terms of communication complexity.

Glossary

Notation	Meaning
$\mathbb{Z}$	set of all integers
$\mathbb{Z}_N = \{0, 1, \ldots, N-1\}$	ring of integers modulo N
$[x]_N = x \bmod N$	the smallest nonnegative residue of x modulo N
$\mathbb{Z}_N^*$	group of integers from $\mathbb{Z}_N$ relatively prime to N
QR_N	set of quadratic residues modulo N
$[x]_N = x \bmod N$	the smallest nonnegative residue of x modulo N
ϕ	Euler's totient function
$\mathbb{F}_p$	finite field with p elements
$E(\mathbb{F}_p)$	elliptic curve over $\mathbb{F}_p$
$\#E(\mathbb{F}_p)$	number of points on the curve $E(\mathbb{F}_p)$
$\mathrm{x}(P) \in \mathbb{Z}$	x-coordinate of $P \in E(\mathbb{F}_p)$ interpreted as an integer
$\Pr[X = x]$	probability that random variable X takes on value x
$\mathsf{E}(X)$	expected value of random variable X
$\mathsf{Var}(X)$	variance of random variable X, $\mathsf{Var}(X) = \mathsf{E}(X - \mathsf{E}(X))^2$
$\Delta(X, Y)$	statistical distance between random variables X and Y
$s \in_R S$	element s is chosen uniformly at random from set S
U_N	random variable uniformly distributed on $\mathbb{Z}_N$
$\ell_j(x)$	the j-th least significant bit of x
$L_j(x) = x \bmod 2^j$	integer consisting of j least significant bits of x
$L(n)$	running time of the Number Field Sieve

Abbreviation	Meaning
NFS	Number Field Sieve
DLNFS	discrete logarithm variant of the NFS
DL problem	discrete logarithm problem
ECDL problem	elliptic curve discrete logarithm problem
CDH problem	computational Diffie-Hellman problem
DDH problem	decisional Diffie-Hellman problem
DLSE problem	discrete logarithm with short exponents problem
DEC generator	Dual Elliptic Curve generator
DSA	Digital Signature Algorithm

References

[ACGS88] W. Alexi, B. Chor, O. Goldeich, and C. P. Schnorr, *RSA and Rabin functions: Certain parts are as hard as the whole*, SIAM Journal on Computing (1988), 194–209.

[BBS86] L. Blum, M. Blum, and M. Shub, *A simple unpredictable pseudo-random number generator*, SIAM Journal on Computing (1986), 364–383.

[BCP03] E. Bresson, D. Catalano, and D. Pointcheval, *A simple public-key cryptosystem with a double trapdoor decryption mechanism and its applications*, 2003, Lecture Notes in Computer Science, vol. 2894, Springer-Verlag, 2003, pp. 37–54.

[BF01] D. Boneh and M. K. Franklin, *Identity-based encryption from the Weil pairing*, Advances in Cryptology—Crypto 01, Lecture Notes in Computer Science, vol. 2139, Springer-Verlag, 2001, pp. 213–229.

[BG07] D. Brown and K. Gjøsteen, *A security analysis of the NIST SP 800-90 elliptic curve random number generator*, Cryptology ePrint Archive, Report 2007/048, 2007, `http://eprint.iacr.org/`. To appear in the proceedings of Crypto 2007.

[BGP06] C. Berbain, H. Gilbert, and J. Patarin, *QUAD: A practical stream cipher with provable security*, Advances in Cryptology—Eurocrypt 2006, Lecture Notes in Computer Science, 2006, pp. 109–128.

[BHHG01] D. Boneh, S. Halevi, and N. Howgrave-Graham, *The modular inversion hidden number problem*, Advances in Cryptology—Asiacrypt 2001, Lecture Notes in Computer Science, vol. 2248, Springer-Verlag, 2001, pp. 36–51.

[BK05] E. Barker and J. Kelsey, *Recommendation for random number generation using deterministic random bit generators*, December 2005, NIST Special Publication (SP) 800-90.

[Ble02] D. Bleichenbacher, *On the generation of DSA one-time keys*, 6th Workshop on Elliptic Curve Cryptography, 2002.

[BM82] M. Blum and S. Micali, *How to generate cryptographically strong sequences of pseudo random bits*, Symposium on Foundations of Computer Science, 1982, pp. 112–117.

[BOCS83] M. Ben-Or, B. Chor, and A. Shamir, *On the cryptographic security of single RSA bits*, ACM Symposium on Theory of Computing, 1983, pp. 421–430.

[BP97] N. Barić and B. Pfitzmann, *Collision-free accumulators and fail-stop signature schemes without trees*, Advances in Cryptology—Eurocrypt 1997, Lecture Notes in Computer Science, vol. 1233, 1997, pp. 480–494.

[BR96] M. Bellare and P. Rogaway, *The exact security of digital signatures – how to sign with RSA and Rabin*, Advances in Cryptology—Eurocrypt 1996, Lecture Notes in Computer Science, vol. 1070, Springer-Verlag, 1996, pp. 399–416.

[BR06] M. Bellare and P. Rogaway, *The security of triple encryption and a framework for code-based game-playing proofs*, Advances in Cryptology—Eurocrypt 2006, Lecture Notes in Computer Science, vol. 4004, Springer-Verlag, 2006, pp. 409–426.

[Bro06] D. Brown, *Conjectured security of the ANSI-NIST elliptic curve RNG*, Cryptology ePrint Archive, Report 2006/117, 2006, `http://eprint.iacr.org/`.

[BV98] D. Boneh and R. Venkatesan, *Breaking RSA may not be equivalent to factoring*, Advances in Cryptology—Eurocrypt 1998, Lecture Notes in Computer Science, vol. 1403, Springer-Verlag, 1998, pp. 59–71.

[Cam06] M. Campagna, *Security bounds for the NIST codebook-based deterministic random bit generator*, Cryptology ePrint Archive, Report 2006/379, 2006, `http://eprint.iacr.org/`.

[Can01] R. Canetti, *Universally composable security: A new paradigm for cryptographic protocols*, Symposium on Foundations of Computer Science, 2001, pp. 136–145.

[CDN01] R. Cramer, I. Damgård, and J. Nielsen, *Multiparty computation from threshold homomorphic encryption*, Advances in Cryptology—Eurocrypt 2001, Lecture Notes in Computer Science, vol. 2045, Springer-Verlag, 2001, pp. 280–300.

[Cor02] J. Coron, *Optimal security proofs for PSS and other signature schemes*, Advances in Cryptology—Eurocrypt 2002, Lecture Notes in Computer Science, vol. 2332, Springer-Verlag, 2002, pp. 272–287.

[CS98] R. Cramer and V. Shoup, *A practical public key cryptosystem provably secure against adaptive chosen ciphertext attack*, Advances in Cryptology—Crypto 1998, Lecture Notes in Computer Science, vol. 1462, Springer, 1998, pp. 13–25.

[CS00] R. Cramer and V. Shoup, *Signature schemes based on the strong RSA assumption*, ACM Transactions on Information and System Security **3** (2000), no. 3, 161–185.

[CS04] R. Cramer and V. Shoup, *Design and analysis of practical public-key encryption schemes secure against adaptive chosen ciphertext attack*, SIAM Journal on Computing **33** (2004), no. 1, 167–226.

[DFK$^+$06] I. Damgaard, M. Fitzi, E. Kiltz, J. Nielsen, and T. Toft, *Unconditionally secure constant-rounds multi-party computation for equality, comparison, bits and exponentiation*, Third Theory of Cryptography Conference, TCC 2006 (Berlin), Lecture Notes in Computer Science, vol. 3876, Springer-Verlag, 2006, pp. 285–304.

[DH76] W. Diffie and M. E. Hellman, *New directions in cryptography*, IEEE Transactions on Information Theory **22** (1976), no. 6, 644–654.

[DSS00] *Digital Signature Standard FIPS PUB 186-2*, January 2000, National Institute of Standards and Technology, `http://csrc.nist.gov/publications/fips/fips186-2/fips186-2-change1.pdf`.

[eBA] *eBATS: ECRYPT Benchmarking of Asymmetric Systems*, `http://www.ecrypt.eu.org/ebats/`.

[ECS] D. Eastlake, S. Crocker, and J. Schiller, *RFC 1750 – Randomness recommendations for security*, Available from http://www.ietf.org/rfc/rfc1750.txt.

[FH96] M. Franklin and S. Haber, *Joint encryption and message-efficient secure computation*, Journal of Cryptology **9** (1996), no. 4, 217–232.

[FPS07] R. R. Farashahi, R. Pellikaan, and A. Sidorenko, *Extractors for binary elliptic curves*, 2007, To appear in the proceedings of the International Workshop on Coding and Cryptography.

[FPSZ06] P. Fouque, D. Pointcheval, J. Stern, and S. Zimmer, *Hardness of distinguishing the MSB or LSB of secret keys in Diffie-Hellman schemes*, International Colloquium on Automata, Languages and Programming, 2006, pp. 240–251.

[FS00] R. Fischlin and C. P. Schnorr, *Stronger security proofs for RSA and Rabin bits*, Journal of Cryptology **13** (2000), no. 2, 221–244.

[FS03] N. Ferguson and B. Schneier, *Practical cryptography*, Wiley Publishing, USA, 2003.

[FSS07] R. R. Farashahi, B. Schoenmakers, and A. Sidorenko, *Efficient pseudorandom generators based on the DDH assumption*, Public Key Cryptography—PKC 07, Lecture Notes in Computer Science, vol. 4450, Springer-Verlag, 2007, pp. 426–441.

[Gam85] T. E. Gamal, *A public key cryptosystem and a signature scheme based on discrete logarithms*, IEEE Transactions on Information Theory **31** (1985), no. 4, 469–472.

[Gen05] R. Gennaro, *An improved pseudo-random generator based on the discrete logarithm problem*, Journal of Cryptology **18** (2005), no. 2, 91–110.

[Gjø06] K. Gjøsteen, *Comments on Dual-EC-DRBG/NIST SP 800-90, Draft December 2005*, March 2006, `http://www.math.ntnu.no/~kristiag/drafts/dual-ec-drbg-comments.pdf`.

[GKR04] R. Gennaro, H. Krawczyk, and T. Rabin, *Secure hashed Diffie-Hellman over non-DDH groups*, Cryptology ePrint Archive, Report 2004/099, 2004, `http://eprint.iacr.org/`.

[GM84] S. Goldwasser and S. Micali, *Probabilistic encryption*, Special issue of Journal of Computer and Systems Sciences **28** (1984), no. 2, 270–299.

[GMR88] S. Goldwasser, S. Micali, and R. Rivest, *A digital signature scheme secure against adaptive chosen-message attacks*, SIAM Journal on Computing **17** (1988), no. 2, 281–308.

[GMT82] S. Goldwasser, S. Micali, and P. Tong, *Why and how to establish a private code on a public network*, Symposium on Foundations of Computer Science, 1982, pp. 134–144.

[GMW87] O. Goldreich, S. Micali, and A. Wigderson, *How to play any mental game or a completeness theorem for protocols with honest majority*, ACM Symposium on Theory of Computing, 1987, pp. 218–229.

[GNU] GNU Open Source Community, *Gnu mp, the gnu multiple precision arithmetic library*, `http://gmplib.org/`.

[Gol95] O. Goldreich, *Three XOR-lemmas – an exposition*, Electronic Colloquium on Computational Complexity **2** (1995), no. 56.

[Gol01] O. Goldreich, *Foundations of cryptography*, Cambridge University Press, Cambridge, UK, 2001.

[GPR06] Z. Gutterman, B. Pinkas, and T. Reinman, *Analysis of the Linux random number generator*, Cryptology ePrint Archive, Report 2006/086, 2006, `http://eprint.iacr.org/`.

[GSV07] J. Garay, B. Schoenmakers, and J. Villegas, *Practical and secure solutions for integer comparison*, Public Key Cryptography—PKC 2007 (Berlin), Lecture Notes in Computer Science, vol. 4450, Springer-Verlag, 2007, pp. 330–342.

[GW96] I. Goldberg and E. Wagner, *Randomness and the Netscape browser*, Dr. Dobb's Journal (1996), 66–70.

[HHR06] I. Haitner, D. Harnik, and O. Reingold, *On the power of the randomized iterate*, Advances in Cryptology—Crypto 2006, Lecture Notes in Computer Science, vol. 4117, Springer-Verlag, 2006, pp. 22–40.

[HILL99] J. Håstad, R. Impagliazzo, L. A. Levin, and M. Luby, *Construction of a pseudo-random generator from any one-way function*, SIAM Journal on Computing **28** (1999), 1364–1396.

[HN99] J. Håstad and M. Näslund, *The security of all RSA and discrete log bits*, Electronic Colloquium on Computational Complexity (ECCC) **6** (1999), no. 37.

[Hoe63] W. Hoeffding, *Probability in equations for sums of bounded random variables*, Journal of the American Statistical Association **56** (1963), 13–30.

[IN89] R. Impagliazzo and M. Naor, *Efficient cryptographic schemes provably as secure as subset sum*, Symposium on Foundations of Computer Science, 1989, pp. 236–241.

[Jia06] S. Jiang, *Efficient primitives from exponentiation in $\mathbb{Z}_p$*, Australasian Conference on Information Security and Privacy, Lecture Notes in Computer Science, vol. 4058, Springer-Verlag, 2006, pp. 259–270.

[JJSH00] A. Juels, M. Jakobsson, E. Shriver, and B. K. Hillyer, *How to turn loaded dice into fair coins*, IEEE Transactions on Information Theory **46** (2000), no. 3, 911–921.

[JN03] A. Joux and K. Nguyen, *Separating decision Diffie-Hellman from computational Diffie-Hellman in cryptographic groups*, Journal of Cryptology **16** (2003), no. 4, 239–247.

[Kal88] B. S. Kaliski, *Elliptic curves and cryptography: A pseudorandom bit generator and other tools*, Ph.D. thesis, MIT, Cambridge, MA, USA, 1988.

[KM06] N. Koblitz and A. Menezes, *Another look at "provable security". II*, Progress in Cryptology—Indocrypt 2006, vol. 4329, Springer-Verlag, 2006, pp. 148–175.

[KSF99] J. Kelsey, B. Schneier, and N. Ferguson, *Yarrow-160: Notes on the design and analysis of the Yarrow cryptographic pseudorandom number generator*, Selected Areas in Cryptography, 1999, pp. 13–33.

[KW03] J. Katz and N. Wang, *Efficiency improvements for signature schemes with tight security reductions*, ACM Conference on Computer and Communications Security, 2003, pp. 155–164.

[KY76] D. Knuth and A. Yao, *The complexity of nonuniform random number generation*, Algorithms and Complexity: New Directions and Recent Results (1976), 357–428.

[Len04] A. K. Lenstra, *Key lengths*, Contribution to The Handbook of Information Security, 2004, `http://cm.bell-labs.com/who/akl/key_lengths.pdf`.

[Lub94] M. Luby, *Pseudorandomness and cryptographic applications*, Princeton University Press, Princeton, NJ, USA, 1994.

[LV00] A. K. Lenstra and E. R. Verheul, *The XTR public key system*, Advances in Cryptology—Crypto 2000, Lecture Notes in Computer Science, vol. 1880, Springer-Verlag, 2000, pp. 1–19.

[LV01] A. K. Lenstra and E. R. Verheul, *Selecting cryptographic key sizes*, Journal of Cryptology **14** (2001), no. 4, 255–293.

[LW83] D. L. Long and A. Wigderson, *How discreet is the discrete log*, ACM Symposium on Theory of Computing, 1983, pp. 413–420.

[Mar95] G. Marsaglia, *The Marsaglia random number CDROM including the Diehard battery of tests of randomness*, 1995, `http://stat.fsu.edu/pub/diehard/`.

[Mau94] U. M. Maurer, *Towards the equivalence of breaking the Diffie-Hellman protocol and computing discrete algorithms*, Advances in Cryptology—Crypto 1994, Lecture Notes in Computer Science, vol. 839, Springer-Verlag, 1994, pp. 271–281.

[MOV93] A. Menezes, T. Okamoto, and S. A. Vanstone, *Reducing elliptic curve logarithms to logarithms in a finite field*, IEEE Transactions on Information Theory **39** (1993), no. 5, 1639–1646.

[MvOV00] A. J. Menezes, P. C. van Oorschot, and S. A. Vanstone, *Handbook of applied cryptography*, CRC Press series on discrete mathematics and its applications, 2000.

[MW96] U. M. Maurer and S. Wolf, *Diffie-Hellman oracles*, Advances in Cryptology—Crypto 1996, Lecture Notes in Computer Science, vol. 1109, Springer-Verlag, 1996, pp. 268–282.

[Nao89] M. Naor, *Bit commitment using pseudo-randomness*, Advances in Cryptology—Crypto 1989, Lecture Notes in Computer Science, vol. 435, Springer-Verlag, 1989, pp. 128–136.

[NO07] T. Nishide and K. Ohta, *Multiparty computation for interval, equality, and comparison without bit-decomposition protocol*, Public Key Cryptography—PKC 2007, Lecture Notes in Computer Science, vol. 4450, Springer-Verlag, 2007, pp. 343–360.

[NR04] M. Naor and O. Reingold, *Number-theoretic constructions of efficient pseudo-random functions*, Journal of the ACM **51** (2004), no. 2, 231–262.

[NY90] M. Naor and M. Yung, *Public-key cryptosystems provably secure against chosen ciphertext attacks*, ACM Symposium on Theory of Computing, 1990, pp. 427–437.

[Par00] B. Parhami, *Computer arithmetic. Algorithms and hardware designs*, Oxford University Peters, 2000.

[Per92] R. Peralta, *On the distribution of quadratic residues and non-residues modulo a prime number*, Mathematics of Computation **58** (1992).

[Pol00] J. M. Pollard, *Kangaroos, monopoly and discrete logarithms*, Journal of Cryptology **13** (2000), no. 4, 437–447.

[Pre01] P. Preuss, *Are the digits of Pi random? Lab researchers may hold the key*, 2001, `http://www.lbl.gov/Science-Articles/Archive/pi-random.html`.

[PS98] S. Patel and G. Sundaram, *An efficient discrete log pseudo random generator*, Advances in Cryptology—Crypto 1998, Lecture Notes in Computer Science, vol. 1462, Springer-Verlag, 1998, pp. 304–317.

[PW00] B. Pfitzmann and M. Waidner, *Composition and integrity preservation of secure reactive systems*, ACM Conference on Computer and Communications Security, 2000, pp. 245–254.

[RS92] C. Rackoff and D. Simon, *Noninteractive zero-knowledge proof of knowledge and chosen ciphertext attack*, Advances in Cryptology—Crypto 1991, Lecture Notes in Computer Science, vol. 576, Springer-Verlag, 1992, pp. 433–444.

[RSA78] R. Rivest, A. Shamir, and L. Adleman, *A method for obtaining digital signatures and public-key cryptosystems*, Communications of the ACM **21** (1978), no. 2, 120–126.

[Sha02] R. Shaltiel, *Recent developments in explicit constructions of extractors*, Bulletin of the EATCS **77** (2002), 67–95.

[Sho05] V. Shoup, *A computational introduction to number theory and algebra*, Cambridge University Press, Cambridge, UK, 2005.

[SKW$^+$99] B. Schneier, J. Kelsey, D. Whiting, D. Wagner, C. Hall, and N. Ferguson, *Performance comparison of the AES submissions*, 1999, `http://www.schneier.com/paper-aes-performance.pdf`.

[SPW06] R. Steinfeld, J. Pieprzyk, and H. Wang, *On the provable security of an efficient RSA-based pseudorandom generator*, Advances in Cryptology—Asiacrypt 2006, vol. 4284, 2006, pp. 194–209.

[SS05] A. Sidorenko and B. Schoenmakers, *Concrete security of the Blum-Blum-Shub pseudorandom generator*, 10th IMA International Conference on Cryptography and Coding, Lecture Notes in Computer Science, vol. 3796, Springer-Verlag, 2005, pp. 355–375.

[SS06] B. Schoenmakers and A. Sidorenko, *Cryptanalysis of the Dual Elliptic Curve pseudorandom generator*, Cryptology ePrint Archive, Report 2006/190, 2006, `http://eprint.iacr.org/`.

[ST06] B. Schoenmakers and P. Tuyls, *Efficient binary conversion for Paillier encrypted values*, Advances in Cryptology—Eurocrypt 2006, Lecture Notes in Computer Science, vol. 4004, Springer-Verlag, 2006, pp. 522–537.

[vOW96] P. van Oorschot and M. Wiener, *On Diffie-Hellman key agreement with short exponents*, Advances in Cryptology—Eurocrypt 1996, Lecture Notes in Computer Science, vol. 1070, Springer-Verlag, 1996, pp. 332–343.

[VV84] U. V. Vazirani and V. V. Vazirani, *Efficient and secure pseudo-random number generation*, Symposium on Foundations of Computer Science, 1984, pp. 458–463.

[Wol99] S. Wolf, *Information-theoretically and computationally secure key agreement in cryptography*, Ph.D. thesis, ETH Zurich, 1999.

[Yao82] A. C. Yao, *Theory and application of trapdoor functions*, Symposium on Foundations of Computer Science, 1982, pp. 80–91.

[YCBC07] B. Yang, O. Chen, D. Bernstein, and J. Chen, *Analysis of QUAD*, To appear in the proceedings of Fast Software Encryption 2007, 2007.

[YY04] A. Young and M. Yung, *Malicious cryptography: Exposing cryptovirology*, Wiley, 2004.

Index

Acknowledgements

First of all, I would like to express my sincere appreciation to my supervisors Henk van Tilborg and Berry Schoenmakers. I am very grateful to Henk for inviting me to do my Ph.D. in Eindhoven and also for his support during these years. Berry was always ready to discuss various issues concerning my research and to answer my questions. I owe him a great deal for his continuous guidance, advice, and encouragement.

Some results included in this thesis are obtained in collaboration with Reza Rezaeian Farashahi and David Galindo. I thank them for their creative ideas and critical insight.

I am grateful to my reading committee of Henk van Tilborg, Berry Schoenmakers, Arjen Lenstra, Claus-Peter Schnorr, Tanja Lange, and David Galindo. Their comments and suggestions helped to improve the thesis significantly.

I thank Arjen Lenstra, Alfred Menezes, and Benne de Weger for interesting discussions and insightful comments. Mehmet Kiraz and José Villegas gave me a lot of feedback on individual chapters. I appreciate it a lot.

Furthermore, I would like to express my deep gratitude to my former supervisor Ernst M. Gabidulin for introducing me to the field of cryptography. Without his initial support, this thesis would not have been started.

I am indebted to all members of the Coding Theory and Cryptography group as well as the members of the Discrete Algebra and Geometry group for the friendly and creative atmosphere, which made my stay at TU Eindhoven enjoyable and unforgettable. In particular, I thank Tim Mussche, who I had the pleasure to share the office with.

I thank my father Vladimir, my brother Yury, and my grandmother Genrietta Andreevna, for their support, understanding, and encouragement. Last but not least, I thank my wife Natasha for her love.

Summary: Design and analysis of provably secure pseudorandom generators

Random numbers are used in a lot of applications. In particular, random numbers are essential for most of the cryptographic systems. The security of a cryptographic system may depend on the “quality” of the random numbers utilized by the system. Therefore, the random numbers used for cryptographic purposes should be generated with utmost care.

Pseudorandom generator is a mechanism for producing random numbers on a deterministic computer. A pseudorandom generator is said to be cryptographically secure if its output cannot be distinguished from uniformly random by any computationally bounded adversary. In some cases, it is possible to prove that a pseudorandom generator is cryptographically secure under a computational assumption, e.g., the intractability of the RSA problem. Such pseudorandom generators are called provably secure.

In this thesis we pursue several goals. First, we analyze the security of several existing pseudorandom generators such as the classical RSA generator and the recently developed Dual Elliptic Curve generator. Second, we design a new family of provably secure pseudorandom generators. Third, we analyse the concrete security of complex cryptographic systems that use a pseudorandom generator as a building block. Finally, we consider the problem of converting random bits into random numbers.

Chapter 3 is devoted to the analysis of the Dual Elliptic Curve generator proposed in NIST SP800-90. The authors claim that the security of the generator relies on the intractability of the elliptic curve discrete logarithm problem but they provide no reduction. We show that in fact the generator is insecure. A simple and efficient distinguisher is presented. Our attack demonstrates that schemes with heuristic security analysis are unreliable, in contrast with provably secure schemes.

The Dual Elliptic Curve generator is modified and generalized in Chapter 4. Although the original Dual Elliptic Curve generator is shown to be insecure, the modified version is provably secure under the decisional Diffie-Hellman assumption. Besides that, the new construction is very efficient. Two specific instances of the new construction are proposed. The techniques used to design the specific instances, for example, a new randomness extractor, are of independent interest for other applications.

Chapter 5 is about the RSA generator. Combining techniques of Fischlin and Schnorr with ideas of Vazirani and Vazirani we construct a new reduction for this generator in the case that more than one bit is output on each iteration. The new reduction is more efficient than all previously known reductions. As a result, it guarantees security of the generator for relatively low seed lengths.

In Chapter 6, we consider a pseudorandom generator as a component of a complex cryptographic system. In particular, we analyze the concrete security of probabilistic digital signature schemes and public key encryption schemes that use a pseudorandom generator as a source of randomness.

Chapter 7 concerns the problem of converting random bits to random numbers. We revisit this problem in the setting of secure multiparty computation. We propose a new algorithm for solving this problem and show that the computational complexity of this algorithm is less than the computational complexity of the existing methods.

Curriculum Vitae

Andrey Sidorenko was born on December 26, 1980 in Moscow, Russia. Between 1987 and 1997, he was a high school student. In 1995, Andrey won the third place in the mathematics olympiad of the Krasnoyarsk region.

In 1997, he entered Moscow Institute of Physics and Technology, where he took a wide variety of courses in mathematics, physics, and computer science. After attending very interesting lectures on cryptography held by prof. dr. Ernst M. Gabidulin, Andrey decided to study this discipline in more detail. Under the supervision of this professor, Andrey wrote his Master's thesis on analysis of the Hidden Field Equations public key cryptosystem.

During the Master's program, Andrey worked as an assistant at the Department of Radio Engineering. At that time, Andrey also worked as a software engineer at the Institute for Information Transmission Problems. He implemented list decoding algorithms for the Reed-Solomon codes.

In 2003, Andrey received his Master's degree with honors. At the same year, he joined the Coding Theory and Cryptography group at Technische Universiteit Eindhoven. Until 2007, he was a Ph.D. student under the supervision of prof. dr. Henk van Tilborg and dr. Berry Schoenmakers.