Efficient Pseudorandom Generators Based on the DDH Assumption

Reza Rezaeian Farashahi, Berry Schoenmakers, and Andrey Sidorenko

Eindhoven University of Technology,
P.O. Box 513, 5600MB Eindhoven, The Netherlands
r.rezaeian@tue.nl, berry@win.tue.nl, a.sidorenko@tue.nl

Abstract. A family of pseudorandom generators based on the decisional Diffie-Hellman assumption is proposed. The new construction is a modified and generalized version of the Dual Elliptic Curve generator proposed by Barker and Kelsey. Although the original Dual Elliptic Curve generator is shown to be insecure, the modified version is provably secure and very efficient in comparison with the other pseudorandom generators based on discrete log assumptions.
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.

1 Introduction

A pseudorandom generator is a deterministic algorithm that converts a short sequence of uniformly distributed random bits into a longer sequence of bits that cannot be distinguished from uniformly random by a computationally bounded algorithm. It is known that a pseudorandom generator can be constructed from any one-way function [27]. Thus, intractability of the discrete logarithm problem suffices to construct a pseudorandom generator. Such a construction was first proposed by Blum and Micali [2]. 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 paper, 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.

Using strong assumptions in order to improve performance of cryptographic schemes is a common practice nowadays. In particular, several pseudorandom generators based on strong number theoretic assumptions have been proposed during the last decade. For instance, Patel and Sundaram [22], Gennaro [8] show that efficient pseudorandom generators can be built if one assumes that computing discrete logarithms with short exponents is a hard problem. Steinfeld et al. [28] propose an improved version of the well-known RSA generator assuming the intractability of a strong variant of the RSA problem.

In comparison with many other assumptions, the DDH assumption is thoroughly studied (for more details about the security of the DDH problem, refer e.g. to [21]) and has become a basis for a wide variety of cryptographic schemes. To the best of our knowledge, the construction proposed in this paper is the first pseudorandom generator based on the DDH assumption. Security of the new construction is tightly related to the intractability of the DDH problem.

1.1 Related Work

Our work is inspired by the publication of Barker and Kelsey [1], in which the so-called Dual Elliptic Curve generator is proposed. 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 Dual Elliptic Curve generator produces two sequences of points s_iP and s_iQ such that s_i is set to be the x-coordinate of $s_{i-1}P$, $i = 1, 2, \ldots, k$. 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 are intractable in $\mathrm{E}(\mathbb{F}_p)$ [3]. However, the binary sequence produced by the generator turns out to be distinguishable from uniform. The reason is that points of the elliptic curve are transformed into random bits in an improper way [10, 24].

Some ideas of the Dual Elliptic Curve generator are present in the earlier work by Naor and Reingold [21]. 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 [21] 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 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}. Accordingly, the goal of Naor and Reingold [21] is to construct not a pseudorandom generator but a pseudorandom function, for which function G turns out to be a suitable building block.

1.2 Our Contributions

We modify and generalize the Dual Elliptic Curve generator such that the modified version is provably secure under the DDH assumption. In comparison with the original Dual Elliptic Curve 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 all known 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 elegant idea of Cramer and Shoup [5] 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 into a uniformly random number in $\mathbb{Z}_q$, which in turn can be easily converted into a string of *uniformly random bits* using, for instance, algorithm Q_2 from [12] (for an overview of probabilistic randomness extractors, refer to [25]). Note that all (probabilistic and deterministic) extractors known so far can only convert random elements of the subgroup into *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 problem in the corresponding groups.

2 Preliminaries

In this section, we introduce some conventions and recall basic definitions.

2.1 Notation

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

$$\Delta(x, y) = \frac{1}{2} \sum_{\alpha \in S} |\Pr[x = \alpha] - \Pr[y = \alpha]|.$$

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

$$|\Pr[\mathcal{D}(x) = 1] - \Pr[\mathcal{D}(y) = 1]| \geq \epsilon.$$

If the statistical distance between x and y is less than ϵ then no algorithm distinguishes x and y with advantage ϵ (see, e.g., Exercise 22 of [17]).

Let U_m denote a random variable uniformly distributed on $\mathbb{Z}_m$. We say that an algorithm is T-time if it halts in time at most T.

2.2 Pseudorandom Generators

Consider a deterministic algorithm $\mathsf{PRG} : \{0,1\}^n \mapsto \{0,1\}^M$, where $M > n$. Loosely speaking, PRG is called a pseudorandom generator if it maps uniformly distributed input into an output which is computationally indistinguishable from uniform. The input is called the seed and the output is called the pseudorandom sequence. The precise definition is given below.

A T-time algorithm $\mathcal{D} : \{0,1\}^M \mapsto \{0,1\}$ is said to be a (T, ϵ)-distinguisher for PRG if

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

Definition 1 (Pseudorandom generator). *Algorithm* PRG *is called a* (T, ϵ)*-secure pseudorandom generator if no* (T, ϵ)*-distinguisher exists for* PRG.

An important question is what level of security (T, ϵ) suffices for practical applications of pseudorandom generators. Unfortunately, the level of security is often chosen arbitrarily. Knuth ([14], p. 176) sets $\epsilon = 0.01$ and consider several values for T up to $53.5 \cdot 10^{12}$ Mips-Years[1]. In [6], the security level is set to $T = 1$ Mips-Year and $\epsilon = 0.01$. In [8], $T = 3.5 \cdot 10^{10}$ Mips-Years, $\epsilon = 0.01$.

The fact that a pseudorandom generator is (T, ϵ)-secure does not automatically mean that the generator is (T', ϵ')-secure for all T' and ϵ' such that $T'/\epsilon' \leq T/\epsilon$. For instance, if a pseudorandom generator is $(T, 0.01)$-secure it does not necessarily mean that the generator is $(T', 0.009)$-secure even if $T \gg T'$. The reason is that a $(T', 0.009)$-distinguisher cannot always be transformed into a $(T, 0.01)$-distinguisher. Indeed, the only way to improve the success probability of the distinguisher is to run it several times on the same input. However, the latter does not always help since there might be "bad" inputs, that is, inputs for which the success probability of the distinguisher is very low or equals 0.

It is reasonable to require that a pseudorandom generator is secure for *all* pairs (T, ϵ) such that the *time-success ratio* T/ϵ is below a certain bound that is set to be 2^{80} time units throughout this paper (the time unit is defined in Section 2.4). Time-success ratio is a standard way to define security of cryptographic schemes [17, 27].

[1] A Mips-Year is defined as the amount of computation that can be performed in one year by a single DEC VAX 11/780 (see also [16]).

2.3 Decisional Diffie-Hellman Problem

Let $\mathbb{G}$ be a multiplicative group of prime order q. For $x, y \in \mathbb{G}$ 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 $s = \log_x y$. The discrete logarithm (DL) problem is to find s given x and y.

Definition 2 (DDH problem). *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 $Y_{DDH} \in_R \mathbb{G}^4$. Algorithm $\mathcal{D}$ is said to solve the decisional Diffie-Hellman (DDH) problem in $\mathbb{G}$ with advantage ϵ if it distinguishes the random variables X_{DDH} and Y_{DDH} with advantage ϵ, that is,*

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

Related to the decisional Diffie-Hellman problem is the computational Diffie-Hellman (CDH) problem (given x, y and x^s, compute y^s).

Clearly, the DL problem is at least as hard to solve as the CDH problem. The CDH problem is proved to be equivalent to the DL problem under certain conditions [18, 19]. Moreover, no groups are known such that the CDH problem is strictly easier to solve than the DL problem. The common practice is to assume that these two problems are 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 easy. The latter groups are referred to as the non-DDH groups [9]. Furthermore, Wolf [29] 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 non-DDH groups, there is no way to solve the DDH problem rather than to solve the CDH problem.

We do not use non-DDH groups in this paper. To compute security parameters for the pseudorandom generators, we assume that the DDH problem and the DL problem are equally hard, in agreement with common practice. We formalize this as follows.

Let T_{DL} be the running time of the best known algorithm for solving a random instance of the DL problem in a group $\mathbb{G}$. Of course, T_{DL} depends on the group $\mathbb{G}$, that is, $T_{DL} = T_{DL}(\mathbb{G})$. For instance, in the case of finite fields, T_{DL} corresponds to the running time of the discrete logarithm variant of the Number Field Sieve, while for most of the ordinary elliptic curves the best known algorithms are the exponential square root attacks.

Assumption 1 *Unless $\mathbb{G}$ is a non-DDH group, no T-time algorithm solves the DDH problem in $\mathbb{G}$ with probability ϵ if $T/\epsilon \leq T_{DL}(\mathbb{G})$.*

2.4 Conventions

Time Units. A unit of time has to be set to measure the running time of the algorithms. Throughout this paper, the unit of time is one DES encryption. According to the data from [16], a software implementation of DES is estimated to take about 360 Pentium clock cycles. Therefore, we assign

$$1 \text{ time unit} = 360 \text{ Pentium clock cycles.}$$

Security level. The table by Lenstra and Verheul [16] implies that 2^{80} DES encryptions will be infeasible for classical computers until the year 2013. Therefore, we set 2^{80} time units as the security level to be reached.

Modular multiplication cost. In [16], it is reported that multiplication modulo p takes about $(\log_2 p)^2/24$ Pentium clock cycles, that is,

$$(\log_2 p)^2/(24 \cdot 360) \text{ time units.}$$

Complexity of discrete logarithm variant of the NFS. The discrete logarithm variant of the Number Field Sieve (NFS) algorithm solves the discrete logarithm problem in a n-bit prime field in expected time $L(n) = A\exp((1.9229+o(1))(n\ln 2)^{1/3}(\ln(n\ln 2))^{2/3})$, where A is a constant. Following [16], we assume that the $o(1)$-term is zero and estimate the constant A from experimental data. Unfortunately, practical experience with the discrete logarithm variant of the NFS is limited. On the other hand, there are several data points for the Number Field Sieve factoring algorithm. For instance, factoring a 512-bit integer is reported to take about $3\cdot 10^{17}$ Pentium clock cycles [16]. Since computing discrete logarithms in n-bit fields takes about the same amount of time as factoring n-bit integers for any n in the current range of interest (cf. [16]), this suggests that $A \approx 4.7\cdot 10^{-5}$ and thus

$$L(n) = 4.7\cdot 10^{-5}\exp(1.9229(n\ln 2)^{1/3}(\ln(n\ln 2))^{2/3}) \text{ time units.}$$

It is believed that the discrete logarithm problem in the extension field is as hard as the discrete logarithm problem in the prime field of similar size (cf. [15]).

3 DDH Generator

In this section, our main result 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. In contrast with the Dual Elliptic Curve generator [1], the DDH 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 enum that "enumerates" the elements of the group).

3.1 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.

Let $x, y \in_R \mathbb{G}$. The seed of the DDH generator (Algorithm 3.1) is $s_0 \in_R \mathbb{Z}_q$ and $\mathsf{randp}_0, \mathsf{randq}_0 \in_R \mathbb{Z}_l$. The DDH generator transforms the seed into the sequence of k pseudorandom numbers from $\mathbb{Z}_q$.

Algorithm 3.1 DDH generator

Input: $s_0\in\mathbb{Z}_q$, $\mathsf{randp}_0\in\mathbb{Z}_l$, $\mathsf{randq}_0\in\mathbb{Z}_l$, $k>0$
Output: kn pseudorandom bits
  for $i=1$ to k **do**
    Set $(s_i, \mathsf{randq}_i) \leftarrow \mathsf{enum}_2(x^{s_{i-1}}, \mathsf{randq}_{i-1})$
    Set $(\mathsf{output}_i, \mathsf{randp}_i) \leftarrow \mathsf{enum}_2(y^{s_{i-1}}, \mathsf{randp}_{i-1})$
  end for
  Return $\mathsf{output}_1, \ldots, \mathsf{output}_k$

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

3.2 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 2. *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. 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,

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

where $U = (u_1, \ldots, u_k)$, $u_i \in_R \mathbb{Z}_q$, $i = 1, \ldots, k$. Let $j \in_R \{1, 2, \ldots, k\}$. Due to the classical hybrid argument (see, e.g., [11, Section 3.2.3]),

$$|\Pr[\mathcal{D}(Z_j) = 1] - \Pr[\mathcal{D}(Z_{j+1}) = 1]| \geq \epsilon/k,$$

where

$$Z_j = (u_1, \ldots, u_{j-1}, \mathsf{output}_1, \ldots, \mathsf{output}_{k-j+1}),$$
$$Z_{j+1} = (u_1, \ldots, u_{j-1}, u_j, \mathsf{output}_1, \ldots, \mathsf{output}_{k-j}),$$

the probability is taken not only over internal coin flips of $\mathcal{D}$ but also over the choice of j. 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 $j \in_R \{1, 2, \ldots, k\}$
Select $r_1, \ldots, r_{j-1} \in_R \mathbb{Z}_q$, $\mathsf{randp}_0 \in_R Z_l$, $\mathsf{randq}_0 \in_R Z_l$
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 $i = 2$ to $k - j$ **do**
  Set $(s_i, \mathsf{randp}_i) \leftarrow \mathsf{enum}(x^{s_{i-1}}, \mathsf{randp}_{i-1})$
  Set $(r_{i+j-1}, \mathsf{randq}_i) \leftarrow \mathsf{enum}(y^{s_{i-1}}, \mathsf{randq}_{i-1})$
end for
Set $Z = (r_1, \ldots, r_k)$
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 terms of Definition 1. It outputs numbers in $\mathbb{Z}_q$ rather than bits. However, converting random numbers into random bits is a relatively easy problem. For instance, one can use Algorithm Q_2 from [12] that is shown to produce on average $n - 2$ bits given a uniformly distributed random number U_q where n

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

For the sake of simplicity, throughout this paper, 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 (the proof can be found, for instance, in [4]).

Lemma 1. *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*.*

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 1. *Let* $0 \leq (2^n - q)/2^n \leq \delta$*. Suppose the DDH generator is not* (T, ϵ)*-secure. Then 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 $\mathcal{D} : \{0,1\}^{kn} \mapsto \{0,1\}$ that runs in time at most T and

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

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

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

Now, the statement follows from Theorem 2.

4 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.

4.1 Group of Quadratic Residues Modulo Safe Prime

To construct the first instance of the DDH generator, we use an elegant idea of Cramer and Shoup [5] who show that there exists a simple deterministic function that enumerates elements of the group of quadratic residues modulo safe prime p.

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 $\mathsf{enum}_1 : \mathbb{G}_1 \mapsto \mathbb{Z}_q$,

$$\mathsf{enum}_1(x) = \begin{cases} x, & \text{if } x \leq q; \\ p - x, & \text{if } x > q. \end{cases}$$

It is shown in [5] that function enum_1 is a bijection. Moreover, enum_1 does not require any additional input, so in terms of Section 3.1 $l = 1$.

Let $x, y \in \mathbb{G}_1$. Let $s_0 \in_R \mathbb{Z}_q$ be the seed. Generator PRG_1 (Algorithm 4.1) is a deterministic algorithm that transforms the seed into the sequence of kn pseudorandom bits.

The next statement follows from Corollary 1.

Algorithm 4.1 Generator PRG_1

Input: $s_0 \in \mathbb{Z}_q$, $k > 0$
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 $\mathsf{output}_1, \ldots, \mathsf{output}_k$

Proposition 1. *Suppose pseudorandom generator* PRG_1 *is not* (T, ϵ)*-secure. Then there exists an algorithm that solves the DDH problem in* $\mathbb{G}_1$ *in time at most* T *with advantage* $\epsilon/k - \delta$.

The seed length n plays the role of the 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 (T, ϵ)-secure for all T, ϵ such that $T/\epsilon < 2^{80}$ time units.

For $\delta = \epsilon/(2k)$, the generator is (T, ϵ)-secure if

$$2kT/\epsilon < T_{DL}(\mathbb{G}_1), \tag{2}$$

where $T_{DL}(\mathbb{G}_1)$ is the running time of the fastest known method for solving the discrete logarithm problem in $\mathbb{G}_1$. According to the current state of the art, we set $T_{DL}(\mathbb{G}_1)$ to be the running time of the discrete logarithm variant of the Number Field Sieve $L(n)$ (see Section 2.4). Note that $k = M/n$. Then, (2) holds if $2MT/(n\epsilon) < L(n)$. For $M = 2^{20}$ and $T/\epsilon = 2^{80}$, the smallest parameter n that satisfies the above inequality is $n \approx 1600$.

Recall that q satisfies $0 \le (2^n - q)/2^n \le \delta$. We have assumed that $\delta = \epsilon/(2k)$. For $M = 2^{20}$, $n = 1600$, and $\epsilon = 2^{-80}$, this condition implies that $0 < 2^n - q < 2^{1500}$. There are plenty of safe primes $p = 2q + 1$ such that $0 < 2^{1600} - q < 2^{1500}$.

4.2 Arbitrary Prime Order Subgroup of $\mathbb{Z}_p^*$

In this section, we show that the DDH generator can be based not only 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 $p-1$, $p-1 = lq$, $l \ge 2$, such that $\gcd(l, q) = 1$. If p is a safe prime then $l = 2$. Denote by $\mathbb{G}_2$ a subgroup of $\mathbb{Z}_p^*$ of order q. Throughout this section, multiplication of integers is done modulo p.

Let $\mathsf{split}_2 : \mathbb{Z}_p^* \mapsto \mathbb{Z}_q \times \mathbb{Z}_l$ denote a bijection that splits an element of $\mathbb{Z}_p^*$ into two smaller numbers. An example of split_2 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:

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

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 2. *Function* enum_2 *defined above is a bijection.*

Proof. Let $f : \mathbb{G}_2 \times \mathbb{Z}_l \mapsto \mathbb{Z}_p^*$ be defined as $f(x, \mathsf{rand}) = xt^{\mathsf{rand}} \bmod p$ for $x \in \mathbb{G}_2$ 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 $\gcd(q, l) = 1$, it 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 1.

Proposition 2. *Suppose pseudorandom generator* PRG_2 *is not* (T, ϵ)*-secure. Then 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. The above pseudorandom generator outputs n bits per two modular exponentiations with n-bit exponent, which implies linear in n number of modular multiplications. Therefore, the computational effort per output bit does *not* depend on n. On the other hand, the computational effort is proportional to m^2. Our goal is now to determine parameters m and n that minimize the computational effort under the condition that the generator is (T, ϵ)-secure for all T, ϵ satisfying $T/\epsilon < 2^{80}$.

For $\delta = \epsilon/(2k)$, generator PRG_2 is (T, ϵ)-secure if

$$2kT/\epsilon < T_{DL}(\mathbb{G}_2), \tag{3}$$

where $T_{DL}(\mathbb{G}_2)$ is the running time of the fastest known method for solving the discrete logarithm problem in $\mathbb{G}_2$. The best algorithms for solving the discrete logarithm problem in $\mathbb{G}_2$ are Pollard's rho method in $\mathbb{G}_2$ and the discrete logarithm variant of the Number Field Sieve in the full multiplicative group $\mathbb{Z}_p^*$. The running time of Pollard's rho method is estimated to be $0.88\sqrt{q}$ group operations (cf. [16]). Since $k = M/n$, condition (3) implies that

$$2MT/(n\epsilon) < \min[L(m),\ 0.88 \cdot 2^{n/2} m^2/(24 \cdot 360)].$$

For $M = 2^{20}$, $T/\epsilon = 2^{80}$, the optimal parameters n and m are $m \approx 1600$, $160 \lessapprox n \leq m$.

In comparison with PRG_1, the seed of PRG_2 is somewhat longer, although if $m \approx n$ it is roughly of the same size. Moreover, PRG_2 is less efficient than PRG_1 in terms of computational effort since 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 the Pollard's rho attack.

4.3 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 [25]). Provided with some extra randomness, it converts a uniformly random element of a subgroup of $\mathbb{Z}_p^*$ of order q into a uniformly random number in $\mathbb{Z}_q$, which in turn can be easily converted into a string of *uniformly random bits* using, for instance, algorithm Q_2 from [12]. Note that all (probabilistic and deterministic) extractors known so far can only convert random elements of the subgroup into *bits that are statistically close to uniform.*

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 into the random binary string.

If the size of the subgroup q is sufficiently large, our extractor is more efficient than the general purpose probabilistic randomness extractors (e.g., the universal hash functions [27]) in terms of the number of extra random bits required. For instance, if the statistical distance to be reached is 2^{-80} our extractor requires less extra randomness than universal hash functions if the size of the subgroup is at least $p/2^{160}$. If the size of the subgroup is close to the size of the group p, our extractor requires just few extra random bits.

The recently proposed deterministic extractor by Fouque et al. [7] 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.*

5 Generator PRG_1 versus the Gennaro generator

To the best of our knowledge, the pseudorandom generator proposed by Gennaro [8] is the best known pseudorandom generator based on a discrete log like problem. In this section, we show that generator PRG_1 is more efficient than the Gennaro generator [8].

Security of the Gennaro generator is based on a variant of the discrete logarithm problem, that is, the discrete logarithm with short exponent (DLSE) problem. Let x, y be elements of a multiplicative group $\mathbb{G}$. The c-DLSE problem is to find s, $0 \leq s < 2^c$, such that $y = x^s$ given x, y and the parameter c. Clearly, the DLSE problem is not harder to solve than the original discrete logarithm problem.

Now, we recall the basic results of [8].

Let g be a generator of $\mathbb{Z}_p^*$, where p is an n-bit safe prime. For a nonnegative integer x let $\ell_j(x) \in \{0, 1\}$ denote the j-th least significant bit of x:

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

Let $x_1 \in_R \mathbb{Z}_{p-1}$ be the seed. The Gennaro generator (Algorithm 5.1) transforms the seed into the pseudorandom sequence of length $k(n - c - 1)$.

Algorithm 5.1 The Gennaro pseudorandom generator

Input: $x_1 \in \mathbb{Z}_{p-1}$, $k > 0$
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 $\mathsf{output}_1, \ldots, \mathsf{output}_k$

The following statement is the concrete version of Theorem 2 of [8].

Theorem 3 (Gennaro). *Suppose the Gennaro pseudorandom generator is not (T, ϵ)-secure. Then there exists an algorithm that solves the c-DLSE in $\mathbb{Z}_p^*$ in time $16c(\ln c)(k/\epsilon)^3 T$ with probability $1/2$.*

The Gennaro 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. [15]). Then, the average computational effort is $1.3cn^2/(24 \cdot 360(n - c - 1))$ time units per output bit. Our goal is now to determine n and c that minimize the computational effort under the condition that the generator is (T, ϵ)-secure for all T, ϵ satisfying $T/\epsilon < 2^{80}$ with a natural limitation $T \geq 1$ time unit.

Theorem 3 implies that the Gennaro generator is (T, ϵ)-secure if

$$32c(\ln c)(k/\epsilon)^3 T < T_{DLSE}(\mathbb{Z}_p^*),$$

where $T_{DLSE}(\mathbb{Z}_p^*)$ is the running time of the fastest algorithm for solving the c-DLSE problem in $\mathbb{Z}_p^*$. The fastest algorithms for solving the DLSE problem are the discrete logarithm variant of the NFS and the Pollard's lambda method. The complexity of the latter is close to $2 \cdot 2^{c/2}$ multiplications in $\mathbb{Z}_p^*$, that is, $2^{c/2+1} n^2/(24 \cdot 360)$ time units (cf. [23]). Note that $k = M/(n-c-1)$, where M is the total number of pseudorandom bits produced by the generator. Thus, the Gennaro generator is (T, ϵ)-secure if

$$\frac{32c(\ln c)M^3 T}{\epsilon^3 (n-c-1)^3} < \min[L(n),\ 2^{c/2+1} n^2/(24 \cdot 360)].$$

For $M = 2^{20}$, $T/\epsilon < 2^{80}$ with a natural limitation $T \geq 1$ the optimal parameters are $n \approx 18000, c \approx 520$.

The secure length of the modulus turns out to be quite large. Recall that generator PRG_1 is provably secure for much smaller parameter n, namely, $n \approx 1600$. The reason is that the reduction in Theorem 3 is *not tight* in the sense that a distinguisher for the Gennaro generator is transformed into the far less efficient solver for the DLSE problem (note that ϵ is raised to the power of 3 in the statement of Theorem 3). On the contrary, the reduction in Theorem 2 is much tighter.

To compare the Gennaro generator with generator PRG_1, we determine the computational effort for both generators.

1. The average computational effort of the Gennaro generator is $1.3cn^2/(24 \cdot 360(n-c-1))$ time units per output bit. For $n = 18000$, $c = 520$, we get about 1500 time units per output bit.
2. The generator PRG_1 outputs n bits at the cost of 2 modular exponentiations with n-bit exponent. The average computational effort for $n = 1600$ is $2.6n^2/(24 \cdot 360) \approx 770$ time units per output bit.

Thus, for $M = 2^{20}$ bits to be produced and for the level of security of 2^{80} time units, generator PRG_1 is about 2 times faster than the Gennaro generator. Furthermore, the seed length of generator PRG_1 is more than 10 times shorter (1600 bits versus 18000 bits).

We draw the attention of the reader to the way the comparison is done. At first sight, it seems that the Gennaro generator is more efficient than generator PRG_1 since the Gennaro generator outputs almost n bits per modular exponentiation with a short c-bit exponent, while generator PRG_1 outputs n bits per 2 exponentiations with a full-size exponent. However, *it should not be neglected* that the n's in these two cases are different. Due to the tighter reduction, generator PRG_1 is provably secure for much smaller n. This is the main reason why generator PRG_1 turns out to be more efficient for the same level of security.

References

1. E. Barker and J. Kelsey, *Recommendation for random number generation using deterministic random bit generators*, December 2005, NIST Special Publication (SP) 800-90.
2. M. Blum and S. Micali, *How to generate cryptographically strong sequences of pseudo-random bits*, SIAM Journal on Computing **13** (1984), no. 4, 850–864.
3. D. Brown, *Conjectured security of the ANSI-NIST Elliptic Curve RNG*, Cryptology ePrint Archive, Report 2006/117, 2006, `http://eprint.iacr.org/`.
4. O. Chevassut, P. Fouque, P. Gaudry, and D. Pointcheval, *Key derivation and randomness extraction*, Cryptology ePrint Archive, Report 2005/061, 2005, `http://eprint.iacr.org/`.
5. R. Cramer and V. Shoup, *Design and analysis of practical public-key encryption schemes secure against adaptive chosen ciphertext attack*, SIAM Journal on Computing (2003), 167–226.

6. R. Fischlin and C. P. Schnorr, *Stronger security proofs for RSA and Rabin bits*, Journal of Cryptology **13** (2000), no. 2, 221–244.
7. P. Fouque, D. Pointcheval, J. Stern, and S. Zimmer, *Hardness of distinguishing the MSB or LSB of secret keys in Diffie-Hellman schemes*, ICALP (2), 2006, pp. 240–251.
8. R. Gennaro, *An improved pseudo-random generator based on the discrete logarithm problem*, Journal of Cryptology **18** (2005), no. 2, 91–110.
9. 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/`.
10. 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`.
11. O. Goldreich, *Foundations of cryptography*, Cambridge University Press, Cambridge, UK, 2001.
12. 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.
13. B. S. Kaliski, *Elliptic curves and cryptography: A pseudorandom bit generator and other tools*, Ph.D. thesis, MIT, Cambridge, MA, USA, 1988.
14. D. E. Knuth, *Seminumerical algorithms*, third ed., vol. 3, Addison-Wesley, Reading, MA, USA, 1997.
15. 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.
16. ———, *Selecting cryptographic key sizes*, Journal of Cryptology **14** (2001), no. 4, 255–293.
17. M. Luby, *Pseudorandomness and cryptographic applications*, Princeton University Press, Princeton, NJ, USA, 1994.
18. U. M. Maurer, *Towards the equivalence of breaking the Diffie-Hellman protocol and computing discrete algorithms*, CRYPTO, 1994, pp. 271–281.
19. U. M. Maurer and S. Wolf, *Diffie-Hellman oracles.*, CRYPTO, 1996, pp. 268–282.
20. 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.
21. M. Naor and O. Reingold, *Number-theoretic constructions of efficient pseudo-random functions*, Journal of the ACM **51** (2004), no. 2, 231–262.
22. S. Patel and G. S. Sundaram, *An efficient discrete log pseudo random generator.*, CRYPTO, 1998, pp. 304–317.
23. J. M. Pollard, *Kangaroos, monopoly and discrete logarithms*, Journal of Cryptology **13** (2000), no. 4, 437–447.
24. B. Schoenmakers and A. Sidorenko, *Cryptanalysis of the Dual Elliptic Curve pseudorandom generator*, Cryptology ePrint Archive, Report 2006/190, 2006, `http://eprint.iacr.org/`.
25. R. Shaltiel, *Recent developments in explicit constructions of extractors.*, Bulletin of the EATCS **77** (2002), 67–95.
26. J. H. Silverman, *The arithmetic of elliptic curves*, vol. 106, Springer-Verlag, Berlin, 1986.
27. 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.
28. R. Steinfeld, J. Pieprzyk, and H. Wang, *On the provable security of an efficient rsa-based pseudorandom generator*, Cryptology ePrint Archive, Report 2006/206, 2006, `http://eprint.iacr.org/`.
29. S. Wolf, *Information-theoretically and computationally secure key agreement in cryptography*, Ph.D. thesis, ETH Zurich, 1999.