

# PRG Cryptanalysis

*Instructor: Daniele Micciancio*

UCSD CSE

As a motivating application for the study of lattice algorithms in cryptography we consider the cryptanalysis of (insecure) pseudorandom generators. We recall that a pseudorandom generator is a program  $G(x)$  (computable in deterministic polynomial time) that maps bitstrings  $x \in \{0, 1\}^n$  to longer strings  $G(x) \in \{0, 1\}^m$  such that, if  $x$  is chosen uniformly at random and kept secret, then the output  $G(x)$  “looks” random to any efficient observer or adversary. We will formally define secure pseudorandom generators later on. But, for now, we will use the minimal security requirement that given  $G(x)$ , it should be computationally hard to recover the secret seed  $x$ .

We consider two popular types of generators: subset-sum generators and linear congruential generators. For simplicity, in both cases, we consider a generalized definition of generator where the input  $x$  and output  $G(x)$  are not necessarily bitstrings, but elements of some arbitrary set. We will assume that the secret seed  $x$  is chosen uniformly at random from a set of size approximately  $N = 2^n$ , so that mounting an exhaustive search attack on the seed would take exponential time. The task of the generator is to stretch this relatively short ( $n$ -bits) random seed into a polynomially longer ( $2n$  or  $n^2$ -bits) string, e.g., an element from a set of size roughly  $N^2 = 2^{2n}$  or  $N^n = 2^{n^2}$ .

**Subset-Sum Generator** Let  $p$  be a prime of size  $\log_2 p \approx n^2$ , and  $a_1, \dots, a_n \in \mathbb{Z}_p$  be chosen uniformly at random. The parameters  $p, a_1, \dots, a_n$  are made public, and known to everybody. You can think of these parameters as part of the program that computes  $G$ .

- The input to the generator is a sequence of  $n$  bits  $(x_1, \dots, x_n) \in \{0, 1\}^n$ .
- The output is  $G(\mathbf{x}) = \sum_i a_i \cdot x_i \pmod{p}$ .

Notice that the output is a number with  $\log_2 p \approx n^2$  bits. So, the generator stretches  $n$ -bits to roughly  $n^2$  bits.

You can think of the output  $G(\mathbf{x})$  as the sum (modulo  $p$ ) of a random subset  $I = \{i: x_i = 1\}$  of the values  $a_1, \dots, a_n$ , hence the name “subset-sum”. The subset-sum problem asks, given  $a_1, \dots, a_n$  and target value  $b$ , to recover a subset  $I$  (or its indicator vector  $\mathbf{x}$ ) that adds up to  $G(\mathbf{x}) = b$ . Subset-sum is a famous NP-hard problem, so one may hope that recovering  $\mathbf{x}$  from  $b$  is indeed computationally hard, and cannot be solved much faster than an exponential time exhaustive search.

**Truncated Linear Congruential Generator (LCG)** Here we consider another popular type of generators based on linear recurrence relations. The popularity of the generator is due to its simplicity, and variants of this generator were used for some time as the standard pseudorandom generator for the Unix operating system.

As before, we first choose random public parameters. Let  $p$  be a prime with  $\log_2 p \approx n$  bits, and choose  $a, b \in \mathbb{Z}_p$  uniformly at random. The values  $p, a, b$  are made public and hardwired in the program computing the generator. The input to the generator is a value  $x \in \mathbb{Z}_p$ , representable with  $\log_2 p \approx n$  bits. This value defines a sequence according to the recurrence relation

1.  $x_0 = x$
2.  $x_{i+1} = ax_i + b \pmod{p}$  for  $i = 1, 2, \dots$

Of course, producing  $x_0, \dots, x_{k-1}$  as the output of  $G$  would be insecure, because it reveals the secret  $x_0$ . Omitting  $x_0$ , and starting the output with  $x_1$  would hardly make any difference because one can think of  $x_1$  as being the seed, instead of  $x_0$ . Moreover, the original seed can still be recovered as  $x_0 = (x_1 - b) / a \pmod{p}$ .

In order to protect the seed  $x$ , the generator  $G$  will output only a portion of the bits from each  $x_i$ . For example,  $G$  may output only the  $\delta n$  (for some  $\delta > 0$ , e.g.,  $\delta = 0.1$ ) least significant bits  $y_i = x_i \bmod 2^{\delta n}$ . (Here we are implicitly assuming  $\delta n$  is an integer. E.g., for  $\delta = 0.1$ , assume  $n$  is a multiple of 10.) In order to stretch the  $n$  input bits  $x$  to a sequence of  $n^2$  bits, the generator computes a sequence of length  $k = n / \delta$ , and outputs  $G(x) = (y_0, \dots, y_{k-1})$ . Since each  $y_i$  is  $\delta n$  bits, the output of  $G$  has length  $\delta n (n / \delta) = n^2$ .

In summary, for some parameter  $\delta \in [0, 1]$ :

- The input to the generator is a value  $x \in \mathbb{Z}_p$ , representable with  $\log_2 p \approx n$  bits.
- The output of the generator is  $G(\mathbf{x}) = (y_0, \dots, y_{k-1})$  where  $y_i = x_i \bmod 2^{\lceil \delta n \rceil}$  is obtained by truncating the values produced by the linear congruential recurrence  $x_0 = x, x_{i+1} = ax_i + b \bmod p$ .

## 1 Lattices

Neither generator (LCG or subsetsum) is trivially insecure. In fact, both types of generators, for some values of the parameters, have been (or may still be) used in practice. However, we will see that both generators can be completely broken using lattices. This may be surprising at first because neither generator makes explicit use of lattices, and demonstrates how lattices are powerful combinatorial objects that can be used to attack a variety of seemingly unrelated problems.

So, the first question to answer is: where is the hidden lattice, and what lattice problem is relevant to the cryptanalysis problem? We will see that in both cases the relevant lattice problem is the Closest Vector Problem.

**Definition 1.** *The closest vector problem (CVP), given a lattice basis  $\mathbf{B}$  and a target vector  $\mathbf{t}$ , asks to find the lattice point  $\mathbf{Bx}$  closest to the target, i.e., find an integer vector  $\mathbf{x}$  such that  $\|\mathbf{Bx} - \mathbf{t}\|$  is minimized.*

In general, as we will see, CVP is an NP-hard problem. So, we have little hope to efficiently solve this problem in its full generality. However, in practice one often needs to solve only a special variant of this problem, where the target is known to be close to the lattice.

**Definition 2.** *The Bounded Distance Decoding problem ( $\alpha$ -BDD), given a lattice basis  $\mathbf{B}$  and a target vector  $\mathbf{t}$  such that  $\text{dist}(\mathbf{t}, \mathcal{L}(\mathbf{B})) \leq \alpha \lambda_1(\mathcal{L}(\mathbf{B}))$ , asks to find a lattice point  $\mathbf{Bx}$  such that  $\text{dist}(\mathbf{t}, \mathbf{Bx}) \leq \alpha \lambda_1(\mathcal{L}(\mathbf{B}))$ .*

In the above definition  $\text{dist}(\mathbf{t}, \mathcal{L}(\mathbf{B})) = \min \{\|\mathbf{t} - \mathbf{Bx}\| : \mathbf{x} \in \mathbb{Z}^n\}$  is the distance between the target and the closest lattice point, and  $\lambda_1(\mathcal{L}(\mathbf{B})) = \min \{\|\mathbf{Bx}\| : \mathbf{x} \in \mathbb{Z}^n \setminus \{\mathbf{0}\}\}$  is the minimum distance of the lattice. Typically,  $\alpha > 0$  is very small, making the target very close to the lattice, compared to the distance between lattice points. Notice that as soon as  $\alpha < 1/2$ , the solution to BDD is necessarily unique, as for any two solutions  $\mathbf{x}, \mathbf{y}$ , their difference  $\mathbf{x} - \mathbf{y}$  is a lattice vector of length at most

$$\|\mathbf{x} - \mathbf{y}\| \leq \|\mathbf{x} - \mathbf{t}\| + \|\mathbf{t} - \mathbf{y}\| \leq 2\alpha \lambda_1 < \lambda_1.$$

So, by definition of  $\lambda_1$  it must be  $\mathbf{x} - \mathbf{y} = \mathbf{0}$ , i.e.,  $\mathbf{x} = \mathbf{y}$  are the same solution.

The LLL basis reduction algorithm allows to solve  $\alpha$ -BDD in polynomial time when  $\alpha = 2^{-(n-1)/2}$  is exponentially small in the lattice dimension  $n$ . So, for each of the PRG constructions, the questions to be answered are:

- How can the cryptanalysis problem be formulated as an instance of BDD?
- Is the factor  $\alpha$  achieved by LLL sufficient to solve the BDD instance?

## 2 Cryptanalysis of Truncated LCG

Consider a truncated LCG with publicly known parameters  $p, a, b, u$ . Recall that  $p$  is a prime with  $n = \log_2 p$  bits, and  $u = \delta n$  is the number of bits output at each iteration, for some (small) constant  $0 < \delta < 1$ , e.g.,  $\delta = 0.1$ . Given a secret seed  $x$ , the LCG is iterated  $k$  to define the sequence  $x_0, \dots, x_{k-1}$  where

$$\begin{aligned} x_0 &= x \\ x_i &= ax_{i-1} + b \pmod{p} \quad \text{for } i = 1, \dots, k-1. \end{aligned}$$

For concreteness, we set the length of the sequence to  $k=2/\delta$ , so that the generator outputs a string of length  $ku=2\delta n/\delta=2n$  about twice as long as the secret input seed. But you can let  $k$  be an arbitrary parameter. At the end of the proof, we discuss for what values of  $k$  the attack recovers the seed with high probability.

By solving the recurrence we see that

$$x_i = a^i x + b_i \pmod{p} \quad \text{where} \quad b_i = b \sum_{j < i} a^j \pmod{p}.$$

The exact expression for  $b_i$  is not really relevant. All that matters is that each  $b_i$  can be easily computed from the public parameters  $a, b, p$ . Next, write each  $x_i$  as

$$x_i = z_i 2^u + y_i,$$

where  $y_i = x_i \bmod 2^u$  is the block of  $u$  bits output by the generator at iteration  $i$ , and  $z_i = \lfloor x_i / 2^u \rfloor < p / 2^u$  are the (secret) high order bits of the state variable  $x_i$ . We assume the adversary gets to see the output  $\mathbf{y} = (y_0, \dots, y_{k-1})$  of the LCG, and the goal is to recover the seed  $x$ .

First thing to do is to set up an appropriate lattice. All the information above can be compactly described in vector notation as

$$\mathbf{a}x + \mathbf{b} = 2^u \mathbf{z} + \mathbf{y} \pmod{p}$$

where  $\mathbf{a} = (1, a, a^2, \dots, a^{k-1}) \bmod p$ ,  $\mathbf{b} = (b_0, \dots, b_{k-1})$ , and  $\mathbf{y}$  are known, and  $\mathbf{z}$  is an *unknown* vector with relatively *small* entries  $z_i \in [0, p/2^u)$ . Dividing the equation by  $2^u$  (modulo  $p$ ), rearranging the terms and writing the reduction modulo  $p$  explicitly, this can be formulated as a CVP instance with integer lattice  $\Lambda_a = \mathcal{L}(\mathbf{B})$  and target  $\mathbf{t}$  defined as

$$\mathbf{B} = \begin{bmatrix} 1 & & & & \\ a & p & & & \\ a^2 & & p & & \\ \vdots & & & \ddots & \\ a^{k-1} & & & & p \end{bmatrix} \quad \mathbf{t} = 2^{-u} (\mathbf{y} - \mathbf{b}) \pmod{p}$$

The first column of  $\mathbf{B}$  corresponds to the term  $\mathbf{a}$ , which can be used any integer number of times, while the other columns can be used for modular reduction. (Note that the column  $(p, 0, \dots, 0)$  is not included in the matrix. Can you tell why?) We remark that in the definition of  $\mathbf{t}$ , the value  $2^{-u}$  should be interpreted as the inverse of  $2^u$  in  $\mathbb{Z}_p$ , so that the target vector has integer coordinates. The dimension of the lattice is  $k$ , the number of iterations of the LCG.

First, we evaluate the distance of the target  $\mathbf{t}$  to the lattice. If we multiply the first basis vector by  $2^{-u}x(\bmod p)$  and use the other basis vectors for reduction modulo  $p$  we get

$$\mathbf{a}(2^{-u}x) = 2^{-u}(2^u\mathbf{z} + (\mathbf{y} - \mathbf{b})) = \mathbf{z} + \mathbf{t}(\bmod p)$$

which is at distance  $\|\mathbf{z}\| \leq \sqrt{k}(p/2^u) = \mu$  from the target  $\mathbf{t}$ .

Next, we would like to show that the minimum distance of the lattice  $\Lambda_a$  is much larger than  $\mu$ , making this an instance of the BDD problem which we can solve using lattice basis reduction algorithms. We can get a first estimate of the minimum distance of  $\Lambda_a$  by invoking Minkowski's theorem. Since the basis matrix  $\mathbf{B}$  is triangular, the determinant of  $\Lambda_a$  is easily computed as the product of the diagonal elements:  $\det(\Lambda_a) = p^{k-1}$ . So, by Minkowski's theorem, the minimum distance is *at most*  $\lambda \leq \sqrt{k} \det(\Lambda_a)^{1/k} = \sqrt{k} p^{1-1/k}$ . Heuristically, we can expect the bound to be close to the actual value of  $\lambda$ . (This is just a heuristic assumption, we will give a rigorous proof soon.) If  $\lambda \approx \sqrt{k} p^{1-1/k}$ , then  $\lambda$  is larger than  $\mu \leq \sqrt{k} p/2^u$  by a factor  $\gamma = \lambda/\mu = 2^u p^{-1/k} = p^{\delta-1/k}$ , making this an  $\alpha$ -BDD instance for  $\alpha = 1/\gamma$ . So, as long as  $\delta k > 1$  (as needed, for  $G$  to stretch a seed of length  $n$  into a longer string of length  $uk = (\delta k)n$ ), we have  $\gamma > 1$ . For example, if  $G$  stretches the seed by a factor 2, then  $k = 2/\delta$ , and we expect  $\gamma \approx p^{\delta-\delta/2} = p^{\delta/2}$ . Recall that  $p \approx 2^n$  is exponential in the security parameter  $n$ . Since  $\delta$  is a constant, the factor  $\gamma \approx p^{\delta/2} \approx 2^{(\delta/2)n}$  is also exponential in  $n$ , well within the reach of LLL or other practical lattice reduction algorithms.

However, what we described just a heuristic argument, not a proof. This is because Minkowski's theorem gives an *upper bound* on  $\lambda$ , while in order to prove that  $(\mathbf{B}, \mathbf{t})$  is a BDD instance we need a *lower bound* on  $\lambda$ . While Minkowski's bound often provides a good heuristic, it can also be very far from the actual length of  $\lambda$ . In order to prove a lower bound on  $\lambda$  we need to use specific information about the lattice  $\Lambda_a$ . We will show that if  $a$  is chosen uniformly at random (modulo  $p$ ), then  $\lambda(\Lambda_a)$  is much larger than  $\mu$  (say, by a factor  $\gamma \gg 1$ ) with very high probability. This allows to provably break the LCG  $G$  with high probability by solving an instance of the  $\alpha$ -BDD problem, with  $\alpha = 1/\gamma$ .

All vectors in  $\Lambda_a$  equal  $c\mathbf{a}(\bmod p)$  for some integer coefficient  $c$ . So, the shortest nonzero vector in  $\Lambda_a$  can be obtained by picking an integer  $c \neq 0(\bmod p)$ , and then reducing all coordinates of  $c\mathbf{a}$  modulo  $p$  to the range  $(-p/2, p/2)$ . Let  $A$  be the set of all  $a$ 's such that there is a  $c \in \mathbb{Z}_p^*$  for which  $|c \cdot a^i(\bmod p)| \leq \gamma\mu$  for all  $i = 0, \dots, k-1$ . (When taking the absolute value of a number modulo  $p$ , we always assume that the number has been reduced to the interval  $(-p/2, p/2)$ .) Think of  $A$  as the set of *bad* values, for which  $\Lambda_a$  may have a short vector (shorter than the bound  $\gamma\mu$ ). Note that if  $a \notin A$ , then, no matter which  $c$  you pick, we will have  $|c \cdot a^i(\bmod p)| > \gamma\mu$  for some  $i$ , and the length of the shortest nonzero lattice vector will be greater than  $\gamma\mu$ . Since  $a$  is chosen uniformly at random, the probability that  $a \in A$  is  $|A|/p$ . In order to show that this probability is small, we need to bound the size of  $A$ , and show that it is much smaller than  $p$ . To this end, we claim that all  $a \in A$  are the root of a nonzero polynomial  $m(a) = 0(\bmod p)$  with small coefficients. But the number of such polynomials cannot be very large because their coefficients are small, and each polynomial has less than  $k$  roots. So,  $A$  cannot be too large.

**Lemma 3.** *For any  $a \in A$  there is a nonzero polynomial  $q \in \mathbb{Z}_p[x]$  of degree less than  $k$  with coefficients  $|q_i| \leq \beta = (\gamma k \mu)^{1/(k-1)}$  such that  $q(a) = 0$ .*

**Proof.** The intuition is the following. Assume for simplicity that  $c = 1$ , and  $|a^i \pmod p| \leq \gamma \mu$  for all  $i$ . (Below, we adapt the argument to arbitrary  $c$ .) If  $m(x)$  is a polynomial with small coefficients, then  $m(a) = \sum_i m_i a^i \pmod p$  is also small, i.e., it falls in a small range, of size  $\ll p$ . But there are many more polynomials than that, because even if each coefficient is small, there are many coefficients. So two of these polynomials must take the same value on  $a$ . Taking their difference, we see that  $a$  is the zero of a polynomial with small coefficients.

Fix an  $a \in A$  and let  $c \not\equiv 0 \pmod p$  be an integer such that  $|c \cdot a^i \pmod p| \leq \gamma \mu$  for all  $i < k$ . Let  $M$  be the set of all polynomials  $m(x) \in \mathbb{Z}_p[x]$  with coefficients  $|m_i| \leq \beta/2$ . Notice that for any  $m(x) \in M$  we have

$$|c m(a) \pmod p| = \left| \sum_i c m_i a^i \pmod p \right| \leq \sum_i |m_i| |c a^i \pmod p| \leq \frac{\beta}{2} k \gamma \mu.$$

So,  $c m(a) \pmod p$  ranges over a set  $\{-\beta k \gamma \mu / 2, \dots, \beta k \gamma \mu / 2\}$  of size at most  $\beta k \gamma \mu + 1$ . Since  $M$  has size  $(\beta + 1)^k > \beta^k + 1 = (\gamma k \mu)^{k/(k-1)} + 1 = \beta k \gamma \mu + 1$ , by the pigeonhole principle, there must be two distinct polynomials  $m(x), \hat{m}(x) \in M$  such that  $c m(a) = c \hat{m}(a) \pmod p$ . Dividing by  $c$  and subtracting gives a nonzero polynomial  $q(x) = (m - \hat{m})(x)$  with coefficients  $|q_i| \leq \beta$ , such that  $(m - \hat{m})(a) = 0 \pmod p$ .  $\square$

Note that for the proof to work, the set of polynomials with small coefficients should satisfy two competing requirements: it should be large enough so that we can apply the pigeonhole principle in the proof of Lemma 3. But it should also be small enough to bound the size of  $A$ . So, let's see for what values of  $k$  we can balance the two requirements, and successfully break  $G$ .

The number of nonzero polynomials of degree  $< k$  with coefficients bounded by  $\beta$  is  $(2\beta + 1)^k - 1$ , and each one of them has at most  $k - 1$  roots. Therefore, the set  $A$  has size at most

$$|A| \leq (k - 1) (2\beta + 1)^k \approx k 2^k \beta^k.$$

So, using  $\beta = (k \gamma \mu)^{1/(k-1)}$ ,  $\mu = \sqrt{k} p^{1-\delta}$ , and  $k = 2/\delta$  we see that the probability that  $\Lambda_a$  has a nonzero vector of length bounded by  $\gamma \mu$  is at most

$$\Pr_a \{ \lambda(\Lambda_a) \leq \gamma \mu \} \leq \frac{|A|}{p} = C_k \cdot p^{(1-\delta k)/(k-1)} = \frac{C_k}{k^{-1/\delta}}$$

for some constant  $C_k = k^{(2.5k-1)/(k-1)} 2^k (1/\alpha)^{k/(k-1)}$ . Notice that the attack works with a constant number of samples  $k = 2/\delta \approx 20$ , and it requires the solution of a lattice problem in constant dimension  $k$ . So, the factor  $\alpha = 1/\gamma$  achieved by LLL is also constant. Since  $p \approx 2^n$  is exponential in the security parameter  $n$ , the failure probability of the attack (over the choice of  $a, b$ ) is exponentially small.

The number of samples used is also close to optimal: each sample reveals  $\delta n$  bits of information. So, in order to collect enough information to uniquely determine the  $n$ -bit seed  $x \in \mathbb{Z}_p$ , one needs at least  $1/\delta$  samples. The attack we described only uses twice as many, and the same analysis can be used to show that one could have chosen an even smaller  $k \approx \delta$ . In fact, using more samples than necessary does not necessarily lead to a better attack, because it increases the dimension of the lattice  $k$ . This increases the running time of lattice reduction algorithms, and lowers the quality of their output. This is a common feature of many lattice based attacks, where tuning the lattice dimension can be used to optimize the run time performance and success probability of the attacks.

### 3 Cryptanalysis of the Subset-Sum generator

Let  $p$  be a prime of length  $\log_2 p = n^2$ , and  $\mathbf{a} = (a_1, \dots, a_n) \in \mathbb{Z}_p^n$  be chosen uniformly at random. These values are public and known to the attacker. The Subset-Sum generator, on input a uniformly random string  $\mathbf{x} = (x_1, \dots, x_n) \in \{0, 1\}^n$ , outputs  $b = G(\mathbf{x}) = \sum_i a_i x_i \pmod{p} \in \mathbb{Z}_p$ .

Given the public parameters  $p, \mathbf{a}$  and output  $b$  we set up the following lattice and target vector:

$$\mathbf{B} = \begin{bmatrix} Cp & Ca_1 & \dots & Ca_n \\ & 2 & & \\ & & \ddots & \\ & & & 2 \end{bmatrix} \quad \mathbf{t} = \begin{bmatrix} Cb \\ 1 \\ \vdots \\ 1 \end{bmatrix}$$

where  $C$  is a large constant to be determined.

Clearly, there is a lattice vector  $\mathbf{B}(c, \mathbf{x}) = (Cb, 2\mathbf{x})$  within distance  $\sqrt{n}$  from the target. We will prove that the minimum distance of the lattice  $\Lambda_{\mathbf{a}}$  generated by  $\mathbf{B}$  is much larger than that (say, by an exponential factor  $\gamma = 2^{n/2}$ ), so, this is an instance of the BDD problem, and it can be easily solved using lattice reduction algorithms.

As before, we could first give a heuristic argument using Minkoski's bound, and the fact that  $\mathbf{B}$  has determinant  $Cp$ . But the formal proof that (for random  $a_i$ ) the lattice has no short vectors is easy enough that we can jump directly into it.

We set  $C = \gamma\sqrt{n}$ , so that all lattice vectors of length bounded by  $C = \gamma\sqrt{n}$  must have the form  $(0, 2\mathbf{z})$  for some  $2\|\mathbf{z}\| \leq C$ . We consider the set  $Z$  of all nonzero integer vectors  $\mathbf{z}$  of length  $\|\mathbf{z}\| \leq C/2$ . For each  $\mathbf{z} \in Z$ ,  $(0, 2\mathbf{z})$  is a lattice vector if and only if  $\sum_i a_i z_i = 0 \pmod{p}$ . But, when the  $a_i$  are chosen uniformly at random and  $\mathbf{z} \neq 0$ , the value  $\sum_i a_i z_i$  is uniformly distributed in  $\mathbb{Z}_p$ . So,  $(0, 2\mathbf{z})$  is a lattice vector with probability  $1/p$ . By union bound, the probability (over the choice of  $\mathbf{a}$ ) that there is a short vector is

$$\Pr_{\mathbf{a}}\{\lambda(\Lambda_{\mathbf{a}}) \leq C\} \leq \frac{|Z|}{p} \leq \frac{(2C+1)^n}{p} \approx \frac{1}{2^{n/2}}.$$

So, the failure probability is exponentially small.

## 4 Conclusion

We have seen how the cryptanalysis of different types of pseudorandom generators reduces to the bounded distance decoding (BDD) problem, a variant of the closest vector problem (CVP) on point lattices. In the case of the truncated LCG generators, we have seen that even stretching the seed by a small constant factor allows to recover the seed by solving a lattice problem in constant dimension  $k = O(1/\delta)$ . In the case of subset-sum, we used an  $n$ -dimensional lattice, where  $n$  is the security parameter of the pseudorandom generator. The attack works when the generator outputs more than  $n^2$  bits. Performing a similar attack on subset-sum generators with smaller output (say,  $O(n)$  bits), is closely related to finding better solutions to lattice problems. In fact, we will see that subset-sum can be quite a fine problem to build cryptographic functions if used properly.

Both attacks illustrate a common feature of lattice algorithms when applied in cryptography: both for the linear congruential generator, and the subset-sum generator, the attack we described is not guaranteed to work for any input. Rather, it works with high probability when run on a randomly selected problem instance. This makes perfect sense in cryptography, where keys are chosen at random, and define a probability distribution on problem instances. Knowing that a problem is easy or hard in the average, tells you if your randomly chosen key can or cannot be cracked. Notice how both cryptanalysis problems defined a certain (natural) distribution on lattices, and we proved that the relevant lattice problem could be solved with high probability when the lattice is chosen according to that distribution. Average case complexity is relevant not only in cryptanalysis, but also in most practical setting, when the input distribution is known. For this reason, we will be primarily interested in the average case complexity of lattice problems.