CSE 206A: Lattice Algorithms and Applications Spring 2014

Lattice Cryptography

Instructor: *Daniele Micciancio* UCSD CSE

Lattice cryptography studies the construction of cryptographic functions whose security is based on the conjectured intractability of computationally hard lattice problems, like (variants of) the approximate closest vector problem (CVP_γ) and approximate shortest vector problem (SVP_γ).

Most cryptographic schemes are typically described by *functions families*, i.e., collections of functions $f_k\colon X \to Y$ indexed by a set of keys $k \in K$. In cryptographic applications, these are usually required to be *one-way functions*, i.e., easy to compute (given the function key k and input x), but computationally hard to invert (given the key k and target value $y = f(x)$).

Definition 1 *A function family is a collection $\mathcal{F} = \{f_k\colon X \to Y\}_{k\in K}$ of functions with common domain X and range Y, indexed by a set of keys K, together with a (typically uniform) probability distribution over the keys $k \leftarrow K$.*

Often, the domain X is also endowed with a probability distribution $x \leftarrow X$. In order to formalize the notion of efficient computation in the asymptotic setting, one needs to consider sequences of function families $\mathcal{F}_n$ indexed by a security parameter n, where the domain X_n, codomain Y_n and key space K_n depend on n. Efficient computation is then identified with computations that can be carried out in time polynomial in n, and infeasible computations are those taking superpolynomial or exponential time in n. For simplicity, in what follow we fix the value of the security parameter n, and consider a single function family $\mathcal{F} = \{f_k\colon X \to Y\}_{k\in K}$, leaving the dependency on the security parameter n implicit.

We always require function families $\mathcal{F}$ to be efficiently computable, in the sense that the key distribution $k \leftarrow K$ is efficiently samplable, and there is an efficient evaluation algorithm that on input $k \in K$ and $x \in X$, outputs $f_k(x)$. We also assume that testing membership in the domain X, and sampling input values $x \leftarrow X$ can be done efficiently.

One can defined several security properties for function families. Two of the most fundamental properties are collision resistance and one-wayness.

Definition 2 *A function family $\mathcal{F} = \{f_k\colon X \to Y\}$ is collision resistant if for any probabilistic polynomial time algorithm $\mathcal{A}$, the following experiment has success probability $\Pr\{X\} = \epsilon$ for some negligible[1] ϵ:*

1. *Sample $k \leftarrow K$*
2. *Run $(x_1, x_2) \leftarrow \mathcal{A}(k)$*

[1] A function $\epsilon(n)$ is negligible if $\epsilon = n^{-\omega}$. Typically, in cryptography, $\epsilon = 2^{-\Omega(n)}$ is exponentially small in the security parameter n.

3. *X is the event that $x_1, x_2 \in X$, $x_1 \neq x_2$ and $f_k(x_1) = f_k(x_2)$*

Definition 3 *A function family $\mathcal{F} = \{f_k : X \to Y\}$ is one-way with respect to (efficiently samplable) input distribution X if for any probabilistic polynomial time algorithm $\mathcal{A}$, the following experiment has success probability $\Pr\{X\} = \epsilon$ for some negligible function ϵ:*

1. *Sample $k \leftarrow K$, $x \leftarrow X$ and let $y = f_k(x)$*
2. *Run $x' \leftarrow \mathcal{A}(k, y)$*
3. *X is the event that $x' \in X$ and $f_k(x') = y$*

Notice that in the definition of one-wayness the value x' output by the adversary $\mathcal{A}$ is not required to be the same as the one x chosen to compute y. This is to avoid trivial functions (e.g., the constant function $f(x) = 0$) to be considered one-way. Of course, when the functions f_k are injective over X, then the adversary $\mathcal{A}$ succeeds only if it recovers the original $x' = x$.

One last important security definition, that comes up when studying cryptographic primitives providing secrecy, is pseudorandomness.

Definition 4 *A function family $\mathcal{F} = \{f_k : X \to Y\}$ is a pseudorandom generator, with input distribution X, if the two distributions $D_0 = \{(k, f_k(x)) \mid k \leftarrow K, x \leftarrow X\}$ and $D_1 = \{(k, y) \mid k \leftarrow K, y \leftarrow Y\}$ are ϵ-indistinguishable for some negligible ϵ, i.e., for any probabilistic polynomial time algorithm $\mathcal{A}$,*

$$|\Pr\{\mathcal{A}(x) \mid x \leftarrow D_0\} - \Pr\{\mathcal{A}(x) \mid x \leftarrow D_1\}| \leq \epsilon$$

1 Function families from lattices

In lattice cryptography, functions may be indexed by full dimensional lattices (e.g., represented by a basis), with the lattice dimension serving as the security parameter. Given a lattice basis $\mathbf{B} \in \mathbb{R}^{n \times n}$, one can define the function

$$f_{\mathbf{B}}(\mathbf{x}) = \mathbf{x} \bmod \mathbf{B} = \mathbf{B}(\mathbf{B}^{-1}\lfloor \mathbf{x} \rfloor).$$

In other words, $f_{\mathbf{B}}(\mathbf{x})$ rounds $\mathbf{x}$ to the corner of the fundamental parallelepiped $f_{\mathbf{B}}(\mathbf{x}) + \mathcal{P}(\mathbf{B})$ containing $\mathbf{x}$. As defined, $f_{\mathbf{B}}$ is a function with domain $X \subseteq \mathbb{R}^n$ and codomain $\mathcal{P}(\mathbf{B})$. Notice that without any restriction on the domain X, the function $f_{\mathbf{B}}(\mathbf{x}) = \mathbf{y}$ is easy to invert because $f_{\mathbf{B}}(\mathbf{y}) = \mathbf{y}$, so $\mathbf{y}$ itself is a valid preimage of $\mathbf{y}$. However, if we let the domain X of the function be a set of small vectors (say, the set $X = \mathcal{B}(r)$ of all vectors of length at most r), then inverting $\mathbf{y} = f_{\mathbf{B}}(\mathbf{x})$ corresponds to finding a lattice point $\mathbf{v} \in \mathcal{L}(\mathbf{B})$ within distance r from $\mathbf{y}$, i.e., solving some version of the closest vector problem:

- In one direction, if there is an adversary $\mathcal{A}(\mathbf{B}, \mathbf{y}) = \mathbf{x}'$ that outputs a valid preimage of $\mathbf{y}$ (i.e. $\mathbf{x}' \in X$ and $f_{\mathbf{B}}(\mathbf{x}') = \mathbf{y}$), then we can easily compute a lattice vector $\mathbf{v} = \mathbf{y} - \mathbf{x}' \in \mathcal{L}(\mathbf{B})$ within distance $\|\mathbf{y} - \mathbf{v}\| = \|\mathbf{x}'\| \leq r$ from $\mathbf{y}$.

- Conversely, if we can find a lattice vector $\mathbf{v} \in \mathcal{L}(\mathbf{B})$ within distance r from $\mathbf{y}$, then we also have a valid preimage $\mathbf{x}' = \mathbf{y} - \mathbf{v}$ because $\|\mathbf{x}'\| = \|\mathbf{y} - \mathbf{v}\| \leq r$ and $f_{\mathbf{B}}(\mathbf{x}') = (\mathbf{y} - \mathbf{v}) \bmod \mathbf{B} = \mathbf{y}$.

We can think of $\mathbf{y} = f_{\mathbf{B}}(\mathbf{x})$ as the target from the coset $\mathbf{x}+\mathcal{L}(\mathbf{B})$ that makes the inversion problem computationally hardest. This is because given an arbitrary $\mathbf{y}' \in \mathbf{x}+\mathcal{L}(\mathbf{B})$, one can efficiently recover $\mathbf{y}$ by computing $\mathbf{y} = f_{\mathbf{B}}(\mathbf{y}') = f_{\mathbf{B}}(\mathbf{y}) = f_{\mathbf{B}}(\mathbf{x})$. So, if one can efficiently recover $\mathbf{x}$ given $\mathbf{y}$, then one can also recover it given $\mathbf{y}'$.

Notice also that while the definition of $f_{\mathbf{B}}$ depends on a basis $\mathbf{B}$, we can think of $f_{\mathbf{B}}(\mathbf{x})$ as computing a standard representative of the coset $\mathbf{x} + \Lambda$ under a known (but arbitrary) representation of a lattice $\Lambda = \mathcal{L}(\mathbf{B})$. This is so because given any $\mathbf{y} = f_{\Lambda}(\mathbf{x}) \in \mathbf{x} + \Lambda$, one can efficiently compute $f_{\mathbf{B}}(\mathbf{x}) = f_{\mathbf{B}}(\mathbf{y})$ for any specific basis $\mathbf{B}$, as long as $\mathbf{B}$ is known.

So, more abstractly, we can think of our function family as being defined as $f_{\Lambda}(\mathbf{x}) = \mathbf{x}+\Lambda$, where the key Λ is an n-dimensional lattice. Other representations are possible. For example, an equivalent way to define a concrete representation of these functions is to represent the lattice Λ by a dual basis $\mathcal{L}(\mathbf{D}) = \hat{\Lambda}$, and let $f_{\mathbf{D}}(\mathbf{x}) = \mathbf{D}^T\mathbf{x} \pmod 1 \in [0,1)^n$.

Representational issues aside, in order to properly define a cryptographic function family, we also need to select a probability distribution over the keys, i.e., the n-dimensional lattices Λ indexing the functions. Some definitions (e.g., one-wayness and pseudorandomness) also require a probability distribution over the input vectors $\mathbf{x} \in X$. We will discuss these average-case complexity in the next section. But before doing that, we give a closer look at the CVP instances corresponding to inverting f_{Λ} as a worst-case computational problem, i.e., for an arbitrary lattice Λ and input $\mathbf{x} \in \mathcal{B}(r)$. When explaining the relation between inverting $\mathbf{y} = f_{\Lambda}(\mathbf{x})$ and finding a lattice point close to $\mathbf{y}$, we said that this is "some version of" the closest vector problem. There are two ways in which this inversion problem may differ from a generic instance of CVP, depending on the size r of the domain:

- If $r < mu$ is smaller than the covering radius of the lattice, then not every target $\mathbf{y} \in \mathbb{R}^n$ is allowed: the function f_{Λ} is not surjective, and inverting the function f_{Λ} can be done by solving a restricted version of CVP where the target $\mathbf{y}$ is guaranteed to be relatively close to the lattice.

- If $r \geq \lambda/2$ is bigger than the unique decoding radius, then there may be multiple lattice points within distance r from $\mathbf{y}$, and the inversion problem does not require to find the point closest to the target: any lattice point withint distance r from the target is considered a valid solution.

Of special interest are settings of r for which only one of the two restrictions above applies:

- $r < \lambda(\Lambda)/2$: this setting guarantees that there is at most one point within distance r from the target, making the function $f_{\Lambda}\colon \mathcal{B}(r) \to \mathbb{R}^n/\Lambda$ injective. In this regime of parameters, any point within distance r is necessarily the lattice point closest to the target. This is a restricted version of CVP called the Bounded Distance Decoding

(BDD) problem: given a lattice Λ and target $\mathbf{y}$ within distance $r < \lambda/2$ from Λ, find the lattice point closest to $\mathbf{y}$.

- $r > \mu(\Lambda)$: this setting guarantees that there is always at least one lattice point within distance r from the target, making the function $f_\Lambda : \mathcal{B}(r) \to \mathbb{R}^n/\Lambda$ surjective. This is also an important version of CVP, called the Absolute Distance Decoding (ADD) problem: given a lattice Λ, target $\mathbf{y}$ and bound $r \geq mu(\Lambda)$, find a (not necessarily closest) lattice point within distance r from $\mathbf{y}$.

Different parameter settings ($r < \lambda/2$ for injective functions, or $r > \mu$ for surjective functions) are used in different cryptographic applications. For example, collision resistant hash functions are usually expected to produce uniformly random outputs, and therefore require surjective functions. On the other hand, encryption is based on injective functions in order to enable decryption. But before we can study any cryptographic application, we need to define appropriate probability distributions on lattices to be used as keys by the functions.

2 Random Lattices

The two most common distributions over n-dimensional lattices encountered in cryptography are defined as follows. Fix positive integers $k \leq n \leq q$, where k (and/or $n - k$) serves as the main security parameter. Typically n is a small multiple of k (e.g., $n = O(k)$ or $n = O(k \log k)$) and q is a small prime with $O(\log k)$ bits. Notice that q is very small, not at all like the large primes (with $O(k)$ bits) used in number theoretic cryptography.[2] For any matrix $\mathbf{A} \in \mathbb{Z}_q^{k \times n}$ define

$$\Lambda_q^\perp(\mathbf{A}) = \{\mathbf{x} \in \mathbb{Z}^n : \mathbf{A}\mathbf{x} = \mathbf{0} \bmod q\}$$

$$\Lambda_q(\mathbf{A}) = \{\mathbf{x} \in \mathbb{Z}^n : \mathbf{x} = \mathbf{A}^T\mathbf{s} \bmod q \text{ for some } \mathbf{s} \in \mathbb{Z}_q^k\}.$$

Intuitively, $\Lambda_q(\mathbf{A})$ is the lattice generated by the rows of $\mathbf{A}$ modulo q, while $\Lambda_q^\perp(\mathbf{A})$ is the set of solutions of the system of k linear equations modulo q defined by the rows of $\mathbf{A}$. It is easy to check that $\Lambda_q(\mathbf{A})$ and $\Lambda_q^\perp(\mathbf{A})$ are subgroups of $\mathbb{Z}^n$, and therefore they are lattices. It is also apparent from the definition that these lattices are q-ary, i.e., they are periodic modulo q: one can take the finite set Q of lattice points with coordinates in $\{0, \ldots, q-1\}$, and recover the whole lattice by tiling the space with copies $Q + q\mathbb{Z}^n$. The matrix $\mathbf{A}$ used to represent them is not a lattice basis. A lattice basis $\mathbf{B}$ for the corresponding lattices can be efficiently computed from $\mathbf{A}$ using linear algebra, but it is typically not needed: cryptographic operations are usually expressed and implemented directly in terms of $\mathbf{A}$. A random lattice is obtained by picking $\mathbf{A} \in \mathbb{Z}_q^{k \times n}$ uniformly at random. The corresponding distributions are denoted $\Lambda_q(n, k)$ and $\Lambda_q^\perp(n, k)$.

[2] In fact, q is not even required to be prime, but for simplicity in these notes we will assume q is a small prime number.

Regarding the error vector distribution, one possible way to choose $\mathbf{x}$ may be to select it uniformly at random among all integer vectors of bounded norm, but for technical reasons a different distribution is often more convenient. In lattice cryptography, perturbation vectors are typically chosen according to the Gaussian distribution D_α which picks each $\mathbf{x} \in \mathbb{Z}^n$ with probability (roughly) proportional to $\exp(-\pi\|\mathbf{x}/\alpha\|^2)$. The Gaussian distribution has the analytical advantage that the probability of a point $\mathbf{x}$ depends only on its norm $\|\mathbf{x}\|$, and still the coordinates of $\mathbf{x}$ can be chosen *independently* (namely, each with probability proportional to $\exp(-\pi|x_i/\alpha|^2)$). It can be shown that when $\mathbf{x}$ is chosen according to this distribution (over $\mathbb{Z}^n$), $\Pr\{\|\mathbf{x}\| > \sqrt{n}\alpha\}$ is exponentially small. So, by truncating a negligibly small tail, D_α can be regarded as a probability distribution over the integer vectors of norm bounded by $\alpha\sqrt{n}$.

Before providing a theoretical justification for using these distributions in cryptography, let us try to get a better understanding of the lattices. We begin by observing that $\Lambda_q(\mathbf{A})$ and $\Lambda_q^\perp(\mathbf{A})$ are dual to each other, up to a scaling factor q.

Exercise 1 *Show that* $\Lambda_q(\mathbf{A}) = q\cdot\widehat{\Lambda_q^\perp(\mathbf{A})}$ *and* $\Lambda_q^\perp(\mathbf{A}) = q\cdot\widehat{\Lambda_q(\mathbf{A})}$. *In particular,* $\det(\Lambda_q(\mathbf{A}))\cdot\det(\Lambda_q^\perp(\mathbf{A})) = q^n$. *Moreover, for any* $\mathbf{A} \in \mathbb{Z}_q^{k\times n}$, *we have* $\det(\Lambda_q^\perp(\mathbf{A})) \le q^k$ *and* $\det(\Lambda_q(\mathbf{A})) \ge q^{n-k}$.

To better understand the relation between $\Lambda_q(n,k)$ and $\Lambda_q^\perp(n,k)$, it is convenient to define two auxiliary distributions. Let $\tilde{\Lambda}_q^\perp(n,k)$ the conditional distribution of a lattice chosen according to distributions $\Lambda_q^\perp(n,k)$, given that the lattice has determinant exactly q^k. Similarly, let $\tilde{\Lambda}_q(n,k)$ be the conditional distribution of a lattice chosen according to $\Lambda_q(n,n-k)$, given that the determinant of the lattice is q^{n-k}. In both cases, when q is a prime, the condition is equivalent to requiring that the rows of $\mathbf{A}$ are linearly independent modulo q. An equivalent condition (valid for any q) is that the columns of $\mathbf{A}$ generate the whole space $\mathbf{A}\mathbb{Z}_q^n = \mathbb{Z}_q^k$. How much do these conditional distributions differ from the original ones? Not much.

Exercise 2 *Prove that for any* q *and* $\mathbf{A} \in \mathbb{Z}_q^{k\times n}$, *the following conditions are equivalent:*

1. $\det(\Lambda_q^\perp(\mathbf{A})) = q^k$
2. $\det(\Lambda_q(\mathbf{A})) = q^{n-k}$
3. $\mathbf{A}\mathbb{Z}_q^n = \mathbb{Z}_q^k$

Exercise 3 *Show that if* $\mathbf{A} \in \mathbb{Z}_q^{k\times n}$ *is chosen uniformly at random, then* $\Pr\{\det(\Lambda_q^\perp(\mathbf{A})) = q^k\} = \Pr\{\det(\Lambda_q(\mathbf{A})) = q^{n-k}\} = \Pr\{\mathbf{A}\mathbb{Z}_q^n = \mathbb{Z}_q^k\} \ge 1 - 1/q^{n-k}$. *Moreover, the conditional distributions* $\tilde{\Lambda}_q^\perp(n,k) = \tilde{\Lambda}_q(n,n-k)$ *are identical.*

So, for typical settings of the parameters (e.g., $n \ge 2k$), lattices chosen according to $\Lambda_q^\perp(n,k)$ or $\Lambda_q(n,n-k)$ have determinant q^k except with negligible probability $\epsilon \le q^{-k}$, and the distributions $\Lambda_q^\perp(n,k)$ and $\Lambda_q(n,n-k)$ are almost identical because they are both statistically close to $\tilde{\Lambda}_q^\perp(n,k) = \tilde{\Lambda}_q(n,n-k)$.

We now move to estimating the parameters of lattices chosen according to these distributions. Clearly, we always have $\lambda_1 \leq \lambda_n \leq q$ and $\mu \leq \sqrt{n}q$ because $q\mathbf{I}$ gives a set of n linearly independent vectors of length q. In fact, from Minkowski's Theorem and Exercise 1, we know that $\lambda(\Lambda) \leq \sqrt{n}q^{k/n}$ for any $\Lambda \in \Lambda_q^{\perp}(n,k)$. This upper bound is essentially tight.

Exercise 4 *Prove that there is a constant $\delta > 0$ such that if Λ is chosen according to $\Lambda_q^{\perp}(n,k)$, then $\Pr\{\lambda(\Lambda) < \delta\sqrt{n}q^{k/n}\} \leq 1/2^n$.* [Hint: consider all integer vectors of norm at most $\delta\sqrt{n}q^{k/n}$ and use a union bound.]

What about the other parameters λ_n, μ? Also these parameters are very close to Minkowski's upper bound with high probability.

Exercise 5 *Prove that if Λ is chosen according to $\Lambda_q^{\perp}(n,k)$, then $\Pr\{\frac{1}{\delta}\cdot\sqrt{n}\cdot q^{k/n} \leq 2\mu(\Lambda)\} \leq 1/2^n$.*[Hint: Prove the bound for $\Lambda_q(n, n-k) \approx \Lambda_q^{\perp}(n,k)$ instead, and use duality and the transference theorems.]

In summary, when a lattice is chosen according to $\Lambda_q^{\perp}(n,k) \approx \tilde{\Lambda}_q^{\perp}(n,k) = \tilde{\Lambda}_q(n, n-k) \approx \Lambda_q(n, n-k)$, all the parameters $\lambda_1, \ldots, \lambda_n, \mu$ are within a constant factor from Minkowski's bound $\sqrt{n}q^{k/n}$ with overwhelming probability. This provides very useful information about what it means to solve *ADD* or *BDD* on these random lattices. Let Λ be a lattice chosen according to $\Lambda_q^{\perp}(n,k)$, and let $\mathbf{t} = \mathbf{v} + \mathbf{x}$ be a lattice point $\mathbf{v} \in \Lambda$ perturbed by a noise vector $\mathbf{x}$ chosen according to distribution $D_{cq^{k/n}}$ over $\mathcal{B}(c\sqrt{n}q^{k/n})$. Finding a lattice point within distance $c\sqrt{n}q^{k/n}$ from $\mathbf{t}$ is an *ADD* problem when $c > \delta$ and it is a *BDD* problem when $c < 1/\delta$.

3 One way functions

We now show that solving *ADD* and *BDD* on random lattices can be formulated as the problem of inverting the function

$$f_{\mathbf{A}}(\mathbf{x}) = \mathbf{A}\mathbf{x} \pmod{q}. \tag{1}$$

when the matrix $\mathbf{A} \in \mathbb{Z}_q^{k \times n}$ is chosen uniformly at random and the input is chosen according to distribution $D_{cq^{k/n}}$, for some $c > 0$. Of course, given a matrix $\mathbf{A}$ and a value $\mathbf{b} = f_{\mathbf{A}}(\mathbf{x})$, recovering a possible preimage of $\mathbf{y}$ under $f_{\mathbf{A}}$ is just a matter of performing some linear algebra, and it can be efficiently accomplished in a variety of ways. In order to get a hard-to-invert function from (1), one needs to regard $D_{cq^{k/n}}$ as a probability distribution over $\mathcal{B}(c\sqrt{n}q^{k/n})$, and consider $f_{\mathbf{A}}$ as a function with this domain. The relation between inverting $f_{\mathbf{A}}(\mathbf{x}) = \mathbf{b}$ (i.e., finding small solutions to the inhomogeneous system $\mathbf{A}\mathbf{x} = \mathbf{b} \pmod{q}$) and lattice problems is easily explained, and corresponds to the syndrone decoding formulation of CVP. Using linear algebra, one can efficiently find an arbitrary solution $\mathbf{t} \in \mathbb{Z}_q^n$ to the system, but this solution will generally have large entries and not belong to the domain of $f_{\mathbf{A}}$. Linear algebra gives us a little more than an arbitrary solution. It tells us that any solution to the inhomogeneous system $\mathbf{A}\mathbf{x} = \mathbf{b}$ can be expressed as the sum of any fixed

specific solution $\mathbf{t}$ to $\mathbf{At} = \mathbf{b} \pmod q$ and a solution $\mathbf{z}$ to the homogeneous system $\mathbf{Az} = \mathbf{0} \pmod q$. But the set of solutions to the homogeneous system is precisely the lattice $\Lambda_q^\perp(\mathbf{A})$. So, finding a small $\mathbf{x} = \mathbf{t} + \mathbf{z}$ is equivalent to finding a lattice point $-\mathbf{z} \in \Lambda_q^\perp(\mathbf{A})$ within distance $\|\mathbf{t} - (-\mathbf{z})\| = \|\mathbf{x}\|$ from the target $\mathbf{t}$. In summary, inverting $f_\mathbf{A}$ is equivalent to the problem of finding lattice vectors in $\Lambda_q^\perp(\mathbf{A})$ within distance $c\sqrt{n}q^{k/n}$ from the target. If $\mathbf{A}$ is chosen uniformly at random, this corresponds to selecting the lattice according to distribution $\Lambda_q^\perp(n,k)$ and error vector distribution D_α.

Depending of the value of c, this is the ADD_c problem (for $c > \delta$) or the $BDD_{1/c}$ problem (for $c < 1/\delta$). These two different ranges for c also correspond to very different statistical properties of the function $f_\mathbf{A}$. Namely, when $c < 1/\delta$, the function $f_\mathbf{A}$ is injective with high probability (which corresponds to BDD having at most one solution). Similarly, when $c > \delta$, the function $f_\mathbf{A}$ is surjective with high probability. In fact, one can say even more: when $c > \delta$ all the output values in $\mathbb{Z}_q^n$ have almost the same probability under $f_\mathbf{A}(D_{cq^{k/n}})$.

However, while the statistical properties of $f_\mathbf{A}$ are quite different for $c < 1/\delta$ and $c > \delta$, it turns out that if $f_\mathbf{A}$ is a one-way function, then it will look like pretty much like a bijection to any computationally bounded adversary regardless of the value of c:

1. No polynomial time adversary can distinguish the output distribution $f_\mathbf{A}(D_{cq^{k/n}})$ from the uniform distribution over $\mathbb{Z}_q^n$, even when $c < 1/\delta$ and the function is far from producing statistically randon outputs.

2. No polynomial time adversary can find two distinct input values $\mathbf{x} \neq \mathbf{y}$ such that $f_\mathbf{A}(\mathbf{x}) = f_\mathbf{A}(\mathbf{y})$, even when $c > \delta$ and such collisions certainly exist.

The above properties can be formally proves based on the assumption that $f_\mathbf{A}$ is hard to invert. I.e., any efficient algorithm that finds collisions $f_\mathbf{A}(\mathbf{x}) = f_\mathbf{A}(\mathbf{y})$, or tells the two distributions $f_\mathbf{A}(D_{cq^{k/n}})$ and $\mathbb{Z}_q^n$ apart (with nonnegligible advantage over the choice of $\mathbf{A}$), can be turned into an efficient algorithm to invert $f_\mathbf{A}$ (again, for randomly chosen $\mathbf{A}$). We will formally prove some of these properties later on. For now, we observe that these properties immediately gives simple cryptographic applications like collision resistant hash functions and pseudorandom generators, and move on to describe more complex applications like public key encryption.

Lattice duality can be used to give a different (but essentially equivalent) construction of one-way functions. Assume the lattice is chosen according to the (almost identical) distribution $\Lambda_q(n, n-k)$, i.e., $\Lambda = \Lambda_q(\mathbf{A})$ for a random $\mathbf{A} \in \mathbb{Z}_q^{n\times(n-k)}$. Then, the ADD/BDD instance corresponding to lattice Λ and error vector $\mathbf{x} \leftarrow D_\alpha$ can also be formulated by picking a "random" lattice point $\mathbf{v} \in \Lambda$, and perturbing it by $\mathbf{x}$ to obtain the target $\mathbf{t} = \mathbf{v} + \mathbf{x}$. Since the lattice is periodic modulo q, all vectors can be reduced modulo q, and the lattice point can be chosen uniformly at random from the finite set $\Lambda \bmod q$. The random lattice point can be chosen as $\mathbf{v} = \mathbf{As} \bmod q$ for $\mathbf{s} \leftarrow \mathbb{Z}_q^{n-k}$, and the resulting one-way function is:

$$g_\mathbf{A}(\mathbf{s}, \mathbf{x}) = \mathbf{A}^T\mathbf{s} + \mathbf{x} \pmod q.$$

Notice that the input to this function consists of two parts: a uniformly random $\mathbf{s} \in \mathbb{Z}_q^{n-k}$ and a short (typically gaussian) error vector $\mathbf{x}$ which corresponds the input of the original

function $f_{\mathbf{A}'}$. We remark that the two matrices $\mathbf{A}$ and $\mathbf{A}'$ associated to the two functions have different (complementary) dimensions. In particular, they are not the same, rather they correspond to the primal and dual definition of the same q-ary lattice $\Lambda_q(\mathbf{A}') = \Lambda_q^{\perp}(\mathbf{A})$. But, syntactic and representational issues aside, $f_{\mathbf{A}}$ and $g_{\mathbf{A}'}$ are essentially the same one-way function. Both formulations are convenient to use in different contexts, and we will use the two formulations interchangeably. A more important issue the length of error vector $\mathbf{x}$. This vector is drawn from a (truncated) distribution D_α that produces vectors of length at most $\alpha\sqrt{n}$. When α is sufficiently small, both functions are one-to-one (and produce pseudorandom output), while when α is sufficiently large, the output is statistically close to uniform and the function is collision resistant.

The hardness of inverting function $g_{\mathbf{A}}$ is often referred to as the *Learning With Errors* (LWE) problem. More specifically, LWE corresponds to the setting where $\mathbf{A} \in \mathbb{Z}_q^{n \times k}$ for fixed security parameter k and arbitrarily large lattice dimension $n = k^{O(1)}$, and the coordinates of $\mathbf{x}$ chosen independently at random. The LWE terminology comes from the interpretation of inverting g as a learning problem, where the task is to learn a secret k-dimensional vector $\mathbf{s} \in \mathbb{Z}_q^k$ given a polynomial numbers of random independent "noisy" samples of the form $(\mathbf{a}_i, b_i = \mathbf{a}_i\mathbf{s} + x_i)$ where $\mathbf{a}_i$ is chosen uniformly at random and x_i is the (gaussian) noise added to each sample.

4 Public Key Encryption

A public key encryption scheme consists of three algorithms:

1. A key generation algorithm KeyGen that on input a security parameter, outputs a pair of secret and public keys (sk, pk)
2. An encryption algorithm $\text{Enc}(pk, m; r)$ that on input the public key pk, a message m and some randomness r, output a ciphertext c
3. A decryption algorithm $\text{Dec}(sk, c)$ that on input the secret key sk and the ciphertext c, recovers the original message m.

To simplify the presentation and security definitions, we will consider encryption schemes where the message $m \in 0, 1$ consists of a single bit. Longer messages can be encrypted one bit at a time. In practice, it is easy to extend the schemes described here to more efficient systems that allow to encrypt longer messages in one shot.

A bit encryption scheme is secure if no polynomial time adversary $\mathcal{A}$ can distringuish the encryption of 0 from the encryption of 1. More precisely, consider the following experiment:

1. $(sk, pk) \leftarrow KeyGen(k)$
2. $b \leftarrow 0, 1$
3. $b' \leftarrow \mathcal{A}(pk, Enc(pk, b))$

The adversary is successful in the attack if $b = b'$. Clearly, an adversary can always success with probability $\frac{1}{2}$ by guessing at random. The ecryption scheme is secure if the advantage $|\Pr b = b' - \frac{1}{2}| \leq \epsilon$, for an appropriately small $\epsilon > 0$.[3]

How can we encrypt with random lattices? Let $\mathbf{A} \in \mathbb{Z}_q^{k \times n}$ be a randomly chosen matrix. We can think of $\mathbf{A}$ as a global parameter shared by all users, or let the key generation algorithm choose it at random and include it as part of the public key. As already discussed, $\mathbf{A}$ defines two lattices $\Lambda_q(\mathbf{A})$ and $\Lambda_q^\perp(\mathbf{A})$. Public key generation and encryption work as follows.

- Key Generation: The secret key sk is chosen simply as a random short vector $\mathbf{x} \leftarrow D_\alpha^n$. This error vector and lattice $\Lambda_q(\mathbf{A})$ define a hard CVP instance $(\Lambda_q(\mathbf{A}), \mathbf{t})$ where $\mathbf{t} = g_{\mathbf{A}}(\mathbf{s}, \mathbf{x}) = \mathbf{A}^T\mathbf{s} + \mathbf{x} \pmod q$, with $\mathbf{s} \in \mathbb{Z}_q^{n-k}$ chosen uniformly at random. The vector $\mathbf{t}$ is used as a public key pk.

The public key generation can be interpreted as the process as taking a random lattice $\Lambda_q(\mathbf{A})$ and augmenting it by planting in it a very short vector $\mathbf{x}$ (the secret key) to obtain $\Lambda' = \Lambda_q(\mathbf{x}^T, \mathbf{A}) = \Lambda_q(\mathbf{A}') \supseteq \Lambda$ where $\mathbf{A}' = (\mathbf{t}, \mathbf{A})$.[4] Since the output of $g_{\mathbf{A}}$ is pseudorandom, the public key vector $\mathbf{t}$ is computationally indistinguishable from a uniformly random vector in $\mathbb{Z}_q^n$, and the lattice $\Lambda' = \Lambda_q(\mathbf{A}')$ follows a distribution that is computationally indistinguishable from $\Lambda(n, k+1)$.

How can we use lattice Λ' to encrypt? How can the short vector $\mathbf{x} \in \Lambda'$ be used to decrypt? This is more easily explained using real numbers. We will show later that everything can be adapted to use only integer vectors (modulo q).

- Encryption: the randomness used for encryption is also a short vector $\mathbf{r} \in \mathbb{R}^n$, which defines a CVP instance in the dual lattice $\widehat{\Lambda}'$. The randomness $\mathbf{r}$ is used to generate a hard CVP instance $\mathbf{v} + \mathbf{r}$ where $\mathbf{v} \in \widehat{\Lambda}'$ is chosen at random. (Recall that $q\widehat{\Lambda}'$ is a q-ary lattice, and therefore we can choose $\mathbf{v}$ uniformly in $\widehat{\Lambda}' \cap [0,1)^n$.) The message is encrypted by first encoding it into a vector $\textsc{encode}(m) \in \mathbb{R}^n$ and then output the ciphertest $\mathbf{c} = (\mathbf{v} + \mathbf{r}) + \textsc{encode}(m)$.

The intuition (formalized later in an actual proof) is that since $(\mathbf{v} + \mathbf{r})$ is indistinguishable from random, it can be used as a one-time pad. If $\mathbf{c}$ were really random, then it would completely hide the message m, regardless of how the encoding function is defined. In fact, the main purpose of encoding the message m is not security, rather it is to enable decryption. Before giving a formal security analysis, let's examine the decryption process. Recall that any lattice vector $\mathbf{x}$ partitions the dual lattice $\widehat{\Lambda}'$ into hyperplanes $H_i = \{\mathbf{y} : \langle \mathbf{y}, \mathbf{x} \rangle = i\}$. Moreover, the vector $\mathbf{x}$ can be used to determine the distance of any target $\mathbf{y}$ from these hyperplanes by computing the scalar product $\langle \mathbf{y}, \mathbf{x} \rangle$. We will encode $m = 0$ as a point close to the hyperplanes (e.g., the origin $\mathbf{0}$), and the message $m = 1$ as a point far from the hyperplanes. This can be done as follows. Let $\mathbf{m} \in \mathbb{Z}^n/q$ be a vector orthogonal to the rows

[3] The security of cryptographic functions is usually measured in "bits", where k bits of security means that an adversary running in time at most 2^k has advantage at most $\epsilon \leq 2^{-k}$

[4] We use notation $(\mathbf{x}^T, \mathbf{A})$ for a matrix with $\mathbf{x}^T$ as its first row, and the other rows given by $\mathbf{A}$.

of $\mathbf{A}$ and such that $\langle \mathbf{m}, \mathbf{t} \rangle = \langle \mathbf{m}, \mathbf{x} \rangle = 1$. This vector can be easily computed from the public key. Notice that $\mathbf{m}$ is on the hyperplane H_1, so $\mathbf{m}/2$ is halfway between two hyperplanes. The distance between the hyperplanes is $1/\|\mathbf{x}\|$. Since $\mathbf{x}$ is short, the hyperplanes are far apart. Provided $\|\mathbf{r}\| < 1/(4\|\mathbf{x}\|)$, we can decrypt by computing the distance of $\mathbf{c}$ from the hyperplanes, and checking if it is less than $1/(4\|\mathbf{x}\|)$. Specifically:

1. If $m = 0$, then $\mathbf{c}$ is within distance $1/(4\|\mathbf{x}\|)$ from the hyperplanes
2. If $m = 1$, then $\mathbf{c}$ is at distance at least $1/(2\|\mathbf{x}\|) - \|\mathbf{r}\| > 1/(4\|\mathbf{x}\|)$ from the hyperplanes.

So, we can decrypt by computing $\lceil 2\langle \mathbf{x}, \mathbf{c} \rangle \rfloor \pmod 2 = m + \lceil 2\langle \mathbf{x}, \mathbf{r} \rangle \rfloor \pmod 2 = m$.

Everything we have done so far can be readily adapted to using only integer lattices, simply by scaling everything by a factor q. The scaled dual lattice is $q\widehat{\Lambda'} = \Lambda_q^{\perp}(\mathbf{A}')$. The error vector $\mathbf{r}$ and encoding $\mathbf{m}$ are also scaled up. The error $\mathbf{r}$ is chosen according to D_β^n, so that the corresponding random CVP instance is hard on average, and indistinguishable from random. The only part that requires some attention is $\mathbf{m}$, because $q\mathbf{m}/2$ is not integer when q is odd. The encryption process can be slighly simplified by representing the CVP instance using function f rather than function g. Notice that the ciphertext $\mathbf{c}$ is only used to compute the scalar product with a lattice vector $\mathbf{x}$, reduced modulo 1 (or modulo q in the scaled lattice.) This product only depends on the coset of $\mathbf{c}$, and in fact the vector $\mathbf{v}$ cancels out during decryption. So, rather than picking a random lattice point $\mathbf{v}$ and perturbing it by $\mathbf{r}$, one may simply compute $f_{\mathbf{A}'}(\mathbf{r})$ and use that as a one time pad. Similarly, the encoding $\mathbf{m}$ is also processed accordingly to yield $f(q\mathbf{m}) = (q, 0, \ldots, 0)$. Using function f the encryption becomes $\mathbf{c} = f_{\mathbf{A}'}(\mathbf{r}) + (\lceil q/2 \rfloor \cdot m, 0 \ldots, 0)$, where we have rounded $q/2$ to the closest integer so that we can work with only integer vectors. The decryption algorithm can also be easily modified to work with this ciphertext representation. Specifically, we express the secret vector $\mathbf{x}$ with respect to the public matrix $\mathbf{A}'$, to yield $(1, -\mathbf{s})$ and decrypt by $\lceil 2\langle (1, -\mathbf{s}), \mathbf{c} \rangle / q \rfloor \pmod 2 = m$.

We now prove security, using the version of the encryption scheme over the integers. Assume there is an adversary $\mathcal{A}$ that breaks the security of the scheme. Security is defined by the following experiment:

1. A matrix $\mathbf{A}$ and vectors $\mathbf{s}, \mathbf{x}$ are chosen with the appropriate probability distribution. Let $\mathbf{A}' = (g_{\mathbf{A}}^T(\mathbf{s}, \mathbf{x}), \mathbf{A})$.
2. Pick a random $\mathbf{r}$ and compute $\mathbf{y} = f_{\mathbf{A}'}(\mathbf{r})$
3. Choose a message bit $m \in 0, 1$ at random and compute the ciphertext $c_m = \mathbf{y} + \textsc{encode}(m)$
4. Run the adversary on input $\mathcal{A}(\mathbf{A}', \mathbf{c})$. The adversary wins if it outputs m.

Let the probability that $\mathcal{A}$ correctly guesses m in the experiment above be δ_1. We want to prove that the advantage $|\delta - \frac{1}{2}|$ is small. Consider a modified experiment where the first step is replaced by choosing $\mathbf{A}' \in \mathbb{Z}_q^{(k+1)\times n}$ uniformly at random, and let δ_2 be the success probability of $\mathcal{A}$ in this modified experiement. Notice that this is the same as

choosing $\mathbf{A}$ as before, and replacing $g^T_{\mathbf{A}}(\mathbf{s}, \mathbf{x})$ with a uniformly random vector. Since $g^T_{\mathbf{A}}(\mathbf{s}, \mathbf{x})$ is computationally indistinguishable from a uniformly random vector, it must be $\delta_2 \approx \delta_1$.

Now consider a further modification of the experiment, where, beside replacing step 1 with a randomly chosen $\mathbf{A}'$,we also replace step 2 by choosing $\mathbf{y} \in \mathbb{Z}_q^{k+1}$ uniformly at random. Let δ_3 be the success probability of the adversary in this third experiment. Again, since $f_{\mathbf{A}'}(\mathbf{r})$ is pseudorandom, it must be $\delta_3 \approx \delta_2$.

Finally, we observe that $\delta_3 = \frac{1}{2}$ because in the input to the adversary $\mathcal{A}$ is a uniformly random matrix, statistically independent of the message m. So, $\delta_1 \approx \delta_2 \approx \delta_3 = \frac{1}{2}$, proving the security of the scheme.