CS 355 – Topics in Cryptography April 30, 2018

Lecture 9: Lattice Cryptography and the SIS Problem

Instructors: Henry Corrigan-Gibbs, Sam Kim, David J. Wu

1 Introduction

In the next few lectures, we are going to discuss lattice-based cryptography. Why might people find lattice cryptography interesting?

1. Lattice problems allow cryptographic constructions to be based on *worst-case* problems as opposed to *average-case* problems.
2. Many very advanced cryptographic primitives such as *fully homomorphic encryption* can be based on lattice assumptions.
3. Lattice problems are conjectured to be secure against quantum computers.

Average vs. Worst-case problems. One of the key features that lattice cryptography provides is that it allows cryptography to be based on *worst-case* problems. So far, all of the problems that we have seen so far such as CDH, DDH, and Factor are all *average-case* problems. In order for an adversary to break these problems, it must only succeed in solving the problem for some *random* instance. Lattice cryptography can be based on *worst-case* problems such as GapSVP. In order for an adversary to break these problems, it must succeed in solving the problem on *all* instance of the problem.

Let's consider the problem of factoring for example.

- **Average-case factoring assumption**: For all PPT algorithm $\mathcal{A}$, we have

$$\Pr\left[\mathcal{A}(N) \to (p,q) \;\middle|\; p,q \xleftarrow{R} \mathsf{Primes}_\lambda, N = pq\right] = \mathrm{negl}(\lambda).$$

  Here, the randomness is over the random coins of $\mathcal{A}$, and the random coins to sample p, q.

- **Worst-case factoring assumption**: For all PPT algorithm $\mathcal{A}$, there exists some $N = pq$ for $p, q \in \mathsf{Primes}_\lambda$ such that

$$\Pr\left[\mathcal{A}(N) \to (p,q)\right] = \mathrm{negl}(\lambda).$$

  Here, the randomness is just over the random coins of $\mathcal{A}$.

There are many problems that are hard to solve in the worst-case, but easy to solve on average. The fact that lattice cryptography can be based on worst-case problems gives strong evidence to its security. In fact, even without any security implications, just the fact that cryptography can be based on some worst-case hardness assumption alone is also quite interesting from a complexity-theoretic perspective.

Lattice Problems. In order to construct crypto schemes, people do not actually use worst-case problems such as GapSVP directly. Instead, people use average-case problems that have reductions from GapSVP.[1] There are two main average-case problems that people use. The first problem is called the *short integers solutions* (SIS) problem, which was introduced by Ajtai [1]. It gives rise to one-way functions, collision-resistant hash functions, digital signatures, and other "minicrypt" primitives. The second problem is called the *learning with errors* (LWE) problem, which was introduced by Regev [3]. It gives rise to public-key encryption, fully homomorphic encryption, identity-based encryption, and beyond.

The existence of an adversary that can break SIS or LWE can be directly translated to breaking the GapSVP problem. Due to time constraints, in lecture, we will just focus on the average-case problems of SIS and LWE. However, we provide a bit more detail on GapSVP and lattices in Section 4.

2 Short Integers Solutions Problem

Today, we will focus on the *short integers solutions* (SIS) problem.[2]

> $\mathsf{SIS}(n, m, q, B)$: Let $n, m, q, B \in \mathbb{N}$ be positive integers. For a given adversary $\mathcal{A}$, we define the following experiment:
>
> - The challenger samples $\mathbf{A} \xleftarrow{R} \mathbb{Z}_q^{n \times m}$, and gives $\mathbf{A}$ to the adversary $\mathcal{A}$.
> - The adversary $\mathcal{A}$ outputs some *non-zero* vector $\mathbf{x} \in \mathbb{Z}_q^m$.
>
> We define $\mathcal{A}$'s advantage in solving the SIS problem for the set of parameters n, m, q, B, denoted $\mathsf{SISAdv}_{n,m,q,B}[\mathcal{A}]$, to be the probability that $\mathbf{A} \cdot \mathbf{x} = \mathbf{0} \pmod q$ and $\|\mathbf{x}\|_\infty \leq B$.

So the SIS problem is parameterized by the matrix dimensions $n, m \in \mathbb{N}$, modulus q, and a norm bound B on the solution. At first, it might be difficult to keep track of all these parameters.

- One should think of n as the security parameter λ that defines the hardness of the problem. The bigger the n is, the harder the problem becomes.
- The parameter m is set depending on the specific applications, but generally $m >> n$.
- The modulus q can be set to be any $q = \text{poly}(n)$, but concretely, just think of $q = O(n^2)$.
- The norm bound $B << q$ should also be set depending on the specific applications.

It is conjectured that for any sufficiently large $n \in \mathbb{N}$ (this is the security parameter), for any $m, q, B \in \mathbb{N}$, satisfying $q > B \cdot \text{poly}(n)$ (for any polynomial poly), the $\mathsf{SIS}(n, m, q, B)$ is hard.

The $\mathsf{SIS}(n, m, q, B)$ problem can be used to construct a collision resistant hash function.

Defining CRHF. Let's first define collision resistant hash functions.

Definition 2.1 (CRHF)**.** An efficiently computable (deterministic) function $H : \mathcal{K} \times \mathcal{X} \to \mathcal{Y}$ is a collision resistant hash function if it satisfies the following properties:

[1] In fact, these problems average-case problems can be viewed as the average-case versions of GapSVP.

[2] In lecture, we just set $B = 1$. We will talk more about this in the next lecture.

- **Compressing**: $|\mathcal{Y}| < |\mathcal{X}|$
- **Collision Resistance**: For any efficient adversary $\mathcal{A}$, we have

$$\Pr\left[\mathcal{A}(\mathsf{pk}) \to (x, x') \wedge x \neq x' \wedge H(\mathsf{pk}, x) = H(\mathsf{pk}, x') \;\middle|\; \mathsf{pk} \xleftarrow{R} \mathcal{K}\right] = \text{negl}.$$

CRHF from SIS. For a fixed n, let's set $q = n^2$ and set m to be any positive integer satisfying $m > n \log q$. Then, define the key space, input space, and output space as follows:

- $\mathcal{K} = \mathbb{Z}_q^{n \times m}$
- $\mathcal{X} = \{0,1\}^m$
- $\mathcal{Y} = \mathbb{Z}_q^n$

Then, define a hash function $H : \mathcal{K} \times \mathcal{X} \to \mathcal{Y}$ as follows:

$$H(\mathbf{A}, \mathbf{x}) = \mathbf{A} \cdot \mathbf{x} \mod q$$

where $\mathbf{A} \in \mathbb{Z}_q^{n \times m}$, and $\mathbf{x} \in \{0,1\}^m$.

Let's see why this function is a collision-resistant hash function.

- **Compressing**: We can just calculate how many elements are contained in the input space and output space. By simple calculation, we have $|\mathcal{X}| = 2^m$ and $|\mathcal{Y}| = q^n = (2^{\log q})^n = 2^{n \log q}$. Now, since $m > n \log q$, we have $|\mathcal{X}| > |\mathcal{Y}|$.
- **Collision Resistance**: If there exists an adversary $\mathcal{A}$ that can find collisions of H, then we can construct an algorithm $\mathcal{B}$ that uses $\mathcal{A}$ to break $\mathsf{SIS}(n, m, q, 1)$ (here, the bound $B = 1$). The reduction is very simple. $\mathcal{B}$ works as follows:
  1. $\mathcal{B}$ receives a challenge $\mathbf{A} \in \mathbb{Z}_q^{n \times m}$ from the $\mathsf{SIS}(n, m, q, 1)$ challenger. It forwards $\mathbf{A}$ to $\mathcal{A}$.
  2. $\mathcal{A}$ finds two elements $\mathbf{x}, \mathbf{x}' \in \{0,1\}^m$ such that $\mathbf{x} \neq \mathbf{x}'$ and $\mathbf{A} \cdot \mathbf{x} = \mathbf{A} \cdot \mathbf{x}'$.
  3. $\mathcal{B}$ submits $\mathbf{x} - \mathbf{x}' \in \{-1, 0, 1\}^m$ as the solution to the $\mathsf{SIS}(n, m, q, 1)$ challenger. Clearly, the vector $\mathbf{x} - \mathbf{x}'$ is non-zero and satisfies $\|\mathbf{x} - \mathbf{x}'\|_\infty \leq 1$. Furthermore, by linearity of the hash function, we have

$$\mathbf{A} \cdot \mathbf{x} = \mathbf{A} \cdot \mathbf{x}' \Rightarrow \mathbf{A} \cdot (\mathbf{x} - \mathbf{x}') = 0.$$

Let's discuss some useful properties that this SIS hash function satisfies.

Updatability. A useful property of the SIS hash function is that it is efficiently *updatable*. Let's consider a simple example of software distribution. Say you have a large software $P_{0.0}$ that you want to distribute over a public channel. To prevent an attacker from tampering with the software, you maintain a hash digest of the software $\mathbf{y} = H_{\mathsf{pk}}(P_{0.0})$ in some read-only public space. Then, any user that downloads the software P' can verify whether it received the correct software by checking if $H_{\mathsf{pk}}(P') = \mathbf{y}$.

At a later point in time, let's say you want to update the software $P_{0.0}$ slightly to $P_{0.1}$. Then, in order for users to check integrity of $P_{1.0}$, you must update the hash digest in the read-only public space. Hence, you must recompute the has $\mathbf{y} = H_{\mathsf{pk}}(P_{1.0})$. If the software $P_{1.0}$ is very large, then this can be costly.

However, let's say you originally used the SIS hash function $H_{\mathbf{A}}(\cdot)$ to hash $\mathbf{y} = H_{\mathbf{A}}(P_{0.0})$. Then, to update the digest $\mathbf{y}$, you do not need to recompute the hash $H_{\mathbf{A}}(P_{1.0})$, but simply add or subtract the corresponding column vectors of $\mathbf{A}$. As a simple example, let's say $P_{0.0}$ and $P_{1.0}$ only differ in the first bit. Then, to update the $\mathbf{y}$, you just need to subtract $\mathbf{y} - \mathbf{a}_1$ where $\mathbf{a}_1$ is the first column vector of $\mathbf{A}$:

$$\mathbf{A} = \begin{pmatrix} | & | & & | \\ \mathbf{a}_1 & \mathbf{a}_2 & \cdots & \mathbf{a}_m \\ | & | & & | \end{pmatrix}.$$

Universality. The SIS hash function is also a good randomness extractor.

Definition 2.2 (Universal Hash Function)**.** A hash function $H : \mathcal{K} \times \mathcal{X} \to \mathcal{Y}$ is *universal* if for every two distinct elements $\mathbf{x}, \mathbf{x}' \in \mathcal{X}$, we have

$$\Pr_{\mathsf{pk} \xleftarrow{R} \mathcal{K}} \left[H(\mathsf{pk}, \mathbf{x}) = H(\mathsf{pk}, \mathbf{x}') \right] = \frac{1}{|\mathcal{Y}|}.$$

The SIS hash function is, in fact, a universal hash function. Showing this is a simple exercise.

Definition 2.3 (Guessing Probability)**.** Let $\mathcal{X}$ be a finite set and let $D_{\mathcal{X}}$ be some distribution on $\mathcal{X}$. Then, we define the guessing probability of $D_{\mathcal{X}}$ to be $\gamma = \max_{\mathbf{x}^* \in \mathcal{X}} \Pr_{\mathbf{x} \leftarrow D_{\mathcal{X}}}[\mathbf{x}^* = \mathbf{x}]$.

Theorem 2.4 (Simplified Leftover Hash Lemma)**.** *Let $\mathcal{X}$ and $\mathcal{Y}$ be two finite sets, and let $H : \mathcal{K} \times \mathcal{X} \to \mathcal{Y}$ be a universal family of hash functions. Let $D_{\mathcal{X}}$ be some distribution on $\mathcal{X}$ with guessing probability at most γ. Then, for any (unbounded) adversary $\mathcal{A}$, the following distributions are indistinguishable:*

$$\left| \Pr\left[\mathcal{A}(\mathsf{pk}, H(\mathsf{pk}, \mathbf{x})) = 1 \;\middle|\; \mathsf{pk} \xleftarrow{R} \mathcal{K}, \mathbf{x} \leftarrow D_{\mathcal{X}} \right] - \Pr\left[\mathcal{A}(\mathsf{pk}, \mathbf{y}) = 1 \;\middle|\; \mathsf{pk} \xleftarrow{R} \mathcal{K}, \mathbf{y} \xleftarrow{R} \mathcal{Y} \right] \right| = \gamma \cdot |\mathcal{Y}|.$$

We will show how to use the leftover hash lemma next lecture.

3 Bonus Material: Worst-Case Lattice Problems

An n-dimensional lattice $\mathcal{L}$ is any subset of $\mathbb{R}^n$ that is both:

1. an *additive subgroup*: $\mathbf{0} \in \mathcal{L}$, and $-\mathbf{x}, \mathbf{x} + \mathbf{y} \in \mathcal{L}$ for every $\mathbf{x}, \mathbf{y} \in \mathcal{L}$
2. *discrete*: every $\mathbf{x} \in \mathcal{L}$ has a neighborhood in $\mathbb{R}^n$ in which $\mathbf{x}$ is the only lattice point.

They are generally defined with respect to some independent basis vectors $\mathbf{b}_1, \ldots, \mathbf{b}_n$. Then, a lattice is defined to be all integer combinations of these vectors

$$\mathcal{L}(\mathbf{b}_1, \ldots, \mathbf{b}_n) = \left\{ \sum_{i \in [n]} c_i \mathbf{b}_i \;\middle|\; c_i \in \mathbb{Z} \right\}.$$

The *minimum distance* of a lattice $\mathcal{L}$ is the length of a shortest nonzero lattice vector.

Definition 3.1. The *minimum distance* of a lattice $\mathcal{L}$ is the length of a shortest nonzero lattice vector:

$$\lambda_1() = \min_{\mathbf{x} \in \mathcal{L} \setminus \{\mathbf{0}\}} \|\mathbf{x}\|.$$

Lattice Problems. The most well-known computational problems on lattices is the *shortest vector problem*:

Definition 3.2 (Shortest Vector Problem (SVP))**.** Given an arbitrary basis $\mathbf{b}_1, \ldots, \mathbf{b}_n$ of some lattice $\mathcal{L} = \mathcal{L}(\mathbf{b}_1, \ldots, \mathbf{b}_n)$, find the shortest nonzero lattice vector, i.e., a $\mathbf{x} \in \mathcal{L}$ for which $\|\mathbf{x}\| = \lambda_1(\mathcal{L})$.

We can also define the *approximate* version of the shortest vector problem. For a function of the dimension n, $\gamma = \gamma(n)$, define the approximate shortest vector problem as follows:

Definition 3.3 (Approximate Shortest Vector Problem (SVP_γ))**.** Given a basis $\mathbf{b}_1, \ldots, \mathbf{b}_n$ of an n-dimensional lattice $\mathcal{L} = \mathcal{L}(\mathbf{b}_1, \ldots, \mathbf{b}_n)$, find a nonzero vector $\mathbf{x} \in \mathcal{L}$ for which $\|\mathbf{x}\| \leq \gamma(n) \cdot \lambda_1(\mathcal{L})$.

Finally we can define the *decision* version of the approximate shortest vector problem, which we denote by GapSVP_γ.

Definition 3.4 (Decisional Approximate SVP (GapSVP_γ))**.** Given a basis $\mathbf{B}$ of an n-dimensional lattice $\mathcal{L} = \mathcal{L}(\mathbf{B})$ where either $\lambda_1(\mathcal{L}) \leq 1$ or $\lambda_1(\mathcal{L}) > \gamma(n)$, determine which is the case.

When $\gamma(n)$ is any polynomial or even subexponential, the GapSVP_γ problem is conjectured to be hard. If you want to learn more about the background on lattice-based cryptography, there is a really nice reference [2].

References

[1] Miklós Ajtai. Generating hard instances of lattice problems. In *STOC*, 1996.

[2] Chris Peikert et al. A decade of lattice cryptography. *Foundations and Trends® in Theoretical Computer Science*, 10(4):283–424, 2016.

[3] Oded Regev. On lattices, learning with errors, random linear codes, and cryptography. In *STOC*, 2005.