Lattices, Learning with Errors and Post-Quantum Cryptography

Lecture Notes

Contents

Preface

These notes are based on lectures in the course CS294: Lattices, Learning with Errors and Post-Quantum Cryptography at UC Berkeley in Spring 2020.

CHAPTER 1

The Learning with Errors Problem: Introduction and Basic Cryptography

The learning with errors (LWE) problem was introduced in its current form in a seminal work of Oded Regev for which he won the Gödel prize in 2018. In its typical form, the LWE problem asks to solve a system of noisy linear equations. That is, it asks to find $\mathbf{s} \in \mathbb{Z}_q^n$ given

$$\left\{(\mathbf{a}_i, \langle \mathbf{a}_i, \mathbf{s}\rangle + e_i) : \ \mathbf{s} \leftarrow \mathbb{Z}_q^n, \mathbf{a}_i \leftarrow \mathbb{Z}_q^n, e_i \leftarrow \chi\right\}_{i=1}^m \tag{1.1}$$

where:

- $\mathbb{Z}_q = \mathbb{Z}/q\mathbb{Z}$ denotes the finite ring of integers modulo q, $\mathbb{Z}_q^n$ denotes the vector space of dimension n over $\mathbb{Z}_q$;
- χ is a probability distribution over $\mathbb{Z}$ which typically outputs "small" numbers, an example being the uniform distribution over an interval $[-B, \ldots, B]$ where $B \ll q/2$; and
- $a \leftarrow \mathcal{D}$ denotes that a is chosen according to the finite probability distribution $\mathcal{D}$, $a \leftarrow S$ denotes that a is chosen uniformly at random from the (finite) set S.

In this first lecture, we will present various perspectives on the LWE (and the closely related "short integer solutions" or SIS) problem, basic theorems regarding the different variants of these problems and their basic cryptographic applications.

We will shortly derive LWE in a different way, "from first principles", starting from a different view, that of finding special solutions to systems of linear equations.

1.1 Solving Systems of Linear Equations

Consider the problem of solving a system of linear equations

$$\mathbf{A}\mathbf{e} = \mathbf{b} \bmod q \tag{1.2}$$

given $\mathbf{A} \in \mathbb{Z}_q^{n\times m}$ and $\mathbf{b} \in \mathbb{Z}_q^n$. This can be accomplished in polynomial time with Gaussian elimination. However, slight variations of this problem become hard for Gaussian elimination and indeed, we believe, for all polynomial-time algorithms. This course is concerned with two such problems, very related to each other, called the SIS problem and the LWE problem.

The "Total" Regime and SIS

Assume that we now ask for solutions to equation 1.2 where $\mathbf{e}$ lies in some subset $S \subseteq \mathbb{Z}_q^m$. Typically we will think of subsets S that are defined geometrically, for example:

- $S = \{0, 1\}$, which is the classical subset sum problem modulo q. More generally, $S = [-B \dots B]^m$ is the set of all solutions where each coordinate can only take a bounded value (absolute value bounded by some number $B \ll q/2$). This will be the primary setting of interest.

- $S = \mathsf{Ball}_R^2$, the Euclidean ball of (small) radius R.

In all cases, we are asking for *short* solutions to systems of linear equations and hence this is called the SIS (short integer solutions) problem.

The SIS problem $\mathsf{SIS}(n, m, q, B)$ as we will study is parameterized by the number of variables m, the number of equations n, the ambient finite field $\mathbb{Z}_q$, and the bound on the absolute value of the solutions B. Namely, we require that each coordinate $e_i \in [-B, -B+1, \dots, B-1, B]$.

To define an average-case problem, we need to specify the probability distributions for $\mathbf{A}$ and $\mathbf{b}$. We will, for the most part of this course, take $\mathbf{A}$ to be uniformly random in $\mathbb{Z}_q^{n \times m}$. There are two distinct ways to define $\mathbf{b}$. The first is in the "total" regime where we simply choose $\mathbf{b}$ from the uniform distribution over $\mathbb{Z}_q^n$.

What does "total" mean? Total problems in NP are ones for which each problem instance has a solution that can be verified given a witness, but the solution may be hard to find. An example is the factoring problem where you are given a positive integer N and you are asked for its prime factorization. A non-example is the 3-coloring problem where you are given a graph G and you are asked for a 3-coloring; although this problem is in NP, it is not total as not every graph is 3-colorable.

Totality of SIS on the Average. Here, using a simple probabilistic argument, one can show that (B-bounded) solutions are very likely to exist if $(2B+1)^m \gg q^n$, or $m = \Omega(\frac{n \log q}{\log B})$. *We call this regime of parameters the total regime or the SIS regime.* Thus, roughly speaking, in the SIS regime, m is large enough that we are guaranteed solutions (even exponentially many of them) when $\mathbf{A}$ and $\mathbf{b}$ are chosen to be uniformly random. The problem then is to actually find a solution.

A Variant: homogenous SIS. The homogenous version of SIS asks for a *non-zero* solution to equation 1.1 with the right hand side being $\mathbf{0}$, that is, $\mathbf{A}\mathbf{e} = \mathbf{0} \pmod q$. This variant is *worst-case* total as long as $(B+1)^m > q^n$. That is, for every instance $\mathbf{A}$ is guaranteed to have a solution. We leave the proof to the reader (Hint: Pigeonhole). SIS and hSIS are equivalent on the average-case. We again leave the simple proof to the reader.

The Planted Regime and LWE

When $m \ll \frac{n \log q}{\log B}$, one can show again that there are likely to be no B-bounded solutions for a uniformly random $\mathbf{b}$ and thus, we have to find a different, sensible, way to state this problem. To do this, we first pick a B-bounded vector $\mathbf{e}$ and compute $\mathbf{b}$ as $\mathbf{Ae} \bmod q$. In a sense, we *plant* the solution $\mathbf{e}$ inside $\mathbf{b}$. The goal now is to recover $\mathbf{e}$ (which is very likely to be unique) given $\mathbf{A}$ and $\mathbf{b}$. *We call this the planted regime or the LWE regime.*

But why is this LWE when it looks so different from Equation 1.1?

This is because the SIS problem in the planted regime is simply LWE in disguise. For, given an LWE instance $(\mathbf{A}, \mathbf{y}^T = \mathbf{s}^T\mathbf{A} + \mathbf{e}^T)$, let $\mathbf{A}^\perp \in \mathbb{Z}_q^{(m-n)\times m}$ be a full-rank set of vectors in the right-kernel of $\mathbf{A}$. That is,

$$\mathbf{A}^\perp \cdot \mathbf{A}^t = 0 \bmod q$$

Then,

$$\mathbf{b} := \mathbf{A}^\perp \cdot \mathbf{y} = \mathbf{A}^\perp \cdot (\mathbf{A}^t\mathbf{s} + \mathbf{e}) = \mathbf{A}^\perp \cdot \mathbf{e} \bmod q$$

so $(\mathbf{A}^\perp, \mathbf{b})$ is an SIS instance $\mathsf{SIS}(m-n, m, q, B)$ whose solution is the LWE error vector. Furthermore, this is in the planted regime since one can show with an easy probabilistic argument that the LWE error vector $\mathbf{e}$ is unique given $(\mathbf{A}, \mathbf{y})$.

The reader should also notice that we can run the reduction in reverse, creating an LWE instance from a SIS instance. If the SIS instance is in the planted regime, this (reverse) reduction will produce an LWE instance.

In summary, the only difference between the SIS and the LWE problems is whether they live in the total world or the planted world, respectively. But the world you live in may make a big difference. Algorithmically, so far, we don't see a difference. In cryptography, SIS gives us applications in "minicrypt" (such as one-way functions) whereas we need LWE for applications in "cryptomania" and beyond (such as public-key encryption and fully homomorphic encryption).

Decision vs. Search for LWE. In the decisional version of LWE, the problem is to distinguish between $(\mathbf{A}, \mathbf{y}^T := \mathbf{s}^T\mathbf{A} + \mathbf{e}^T \bmod q)$ and a uniformly random distribution. One can show, through a reduction that runs in $\mathsf{poly}(q)$ time, that the two problems are equivalent. The interesting direction is to show that if there is a poly-time algorithm that solves the decision-LWE problem for a uniformly random matrix $\mathbf{A}$, then there is a poly-time algorithm that solves the search LWE problem for a (possibly different and possibly larger) uniformly random matrix $\mathbf{A}'$. We will see a search to decision reduction later in class.

Reductions Between SIS and LWE

SIS is at least as hard as LWE. We wish to show that if you have a solution for SIS w.r.t. $\mathbf{A}$, then it is immediate to solve decision-LWE w.r.t. $\mathbf{A}$. Indeed, given a SIS solution $\mathbf{e}$ such that $\mathbf{Ae} = \mathbf{0} \pmod q$, and a vector $\mathbf{b}^T$, compute $\mathbf{b}^T\mathbf{e} \pmod q$. If $\mathbf{b}$ is an LWE instance, then

$$\mathbf{b}^T\mathbf{e} = (\mathbf{s}^T\mathbf{A} + \mathbf{x}^T)\mathbf{e} = \mathbf{x}^T\mathbf{e} \pmod q$$

which is a "small" number (as long as $\mathbf{x}^T$ is small enough). On the other hand, if $\mathbf{b}$ is random, then this quantity is uniformly random mod q (in particular, with a non-negligible probability, not small). This gives us a distinguisher.

LWE is (quantumly) at least as hard as SIS. This turns out to be true, as we will see later in the course.

SIS, LWE and Lattice Problems

SIS and LWE are closely related to lattices and lattice problems. We will have much to say about this connection, in later lectures.

1.2 Basic Theorems

We start with some basic structural theorems on LWE and SIS.

Normal Form SIS and Short-Secret LWE

The normal form for SIS is where the matrix $\mathbf{A}$ is systematic, that is of the form $\mathbf{A} = [\mathbf{A}'||\mathbf{I}]$ where $\mathbf{A}' \in \mathbb{Z}_q^{n\times(m-n)}$.

Lemma 1. *Normal-form SIS is as hard as SIS.*

Proof. To reduce from normal-form SIS to SIS, simply multiply the input to normal-form SIS (nfSIS), denoted $[\mathbf{A}'||\mathbf{I}]$, on the left by a random matrix $\mathbf{B} \leftarrow \mathbb{Z}_q^{n\times n}$. We will leave it to the reader to verify that the resulting matrix denoted $\mathbf{A} := \mathbf{B}[\mathbf{A}'||\mathbf{I}]$ is uniformly random. Furthermore, a solution to SIS on input $(\mathbf{A}, \mathbf{B}\mathbf{b}')$ gives us a solution to nfSIS on input $(\mathbf{A}', \mathbf{b}')$.

In the other direction, to reduce from SIS to normal-form SIS, write $\mathbf{A}$ as $[\mathbf{A}'||\mathbf{B}]$ and generate $[\mathbf{B}^{-1}\mathbf{A}'||\mathbf{I}]$ as the normal-form SIS instance. Again, a solution to the normal form instance $(\mathbf{B}^{-1}\mathbf{A}', \mathbf{B}^{-1}\mathbf{b})$ gives us a solution to SIS on input $(\mathbf{A}, \mathbf{b})$. $\square$

The corresponding version of LWE is called short-secret LWE where both the entries of $\mathbf{s}$ and that of $\mathbf{e}$ are chosen from the error distribution χ. The proof of the following lemma follows along the lines of that for normal form SIS and is left as an exercise. (Indeed, a careful reader will observe that short-secret LWE is nothing but normal-form SIS in disguise.)

Lemma 2. *There is a polynomial-time reduction from* $\mathsf{ssLWE}(n, m, q, \chi)$ *to* $\mathsf{LWE}(n, m, q, \chi)$ *and one from* $\mathsf{LWE}(n, m, q, \chi)$ *to* $\mathsf{ssLWE}(n, m+n, q, \chi)$.

We will continue to see more structural theorems about LWE through the course, but this suffices for now.

1.3 Basic Cryptographic Applications

Collision-Resistant Hashing

A collision resistant hashing scheme $\mathcal{H}$ consists of an ensemble of hash functions $\{\mathcal{H}_n\}_{n\in\mathbb{N}}$ where each $\mathcal{H}_n$ consists of a collection of functions that map n bits to $m < n$ bits. So, each hash function compresses its input, and by pigeonhole principle, it has collisions. That is, inputs $x \neq y$ such that $h(x) = h(y)$. Collision-resistance requires that every p.p.t. adversary who gets a hash function $h \leftarrow \mathcal{H}_n$ chosen at random fails to find a collision except with negligible probability.

Collision-Resistant Hashing from SIS. Here is a hash family $\mathcal{H}_n$ that is secure under $\mathsf{SIS}(n, m, q, B)$ where $n \log q > m \log(B + 1)$. Each hash function $h_{\mathbf{A}}$ is parameterized by a matrix $\mathbf{A} \in \mathbb{Z}_q^{n \times m}$, takes as input $\mathbf{e} \in [0, \ldots, B]^m$ and outputs

$$h_{\mathbf{A}}(\mathbf{e}) = \mathbf{A}\mathbf{e} \bmod q$$

A collision gives us $\mathbf{e}, \mathbf{e}' \in [0, \ldots, B]^m$ where $\mathbf{A}\mathbf{e} = \mathbf{A}\mathbf{e}' \bmod q$ which in turn says that $\mathbf{A}(\mathbf{e} - \mathbf{e}') = 0 \bmod q$. Since each entry of $\mathbf{e} - \mathbf{e}'$ is in $[-B, \ldots, B]$, this gives us a solution to $\mathsf{SIS}(n, m, q, B)$.

Private-Key Encryption

A private-key encryption scheme has three algorithms: a probabilistic key generation Gen which, on input a security parameter λ, generates a private key sk; a probabilistic encryption algorithm Enc which, on input sk and a message m chosen from a message space $\mathcal{M}$, generates a ciphertext c; and a deterministic decryption algorithm Dec which, on input sk and the ciphertext c, outputs a message m'.

Correctness requires that for every sk generated by Gen and every $m \in \mathcal{M}$,

$$\mathsf{Dec}(sk, \mathsf{Enc}(sk, m)) = m$$

The notion of security for private-key encryption is semantic security or equivalently, CPA-security, as defined in the Pass-Shelat lecture notes (see References at the end of the notes.) In a nutshell, this says that no probabilistic polynomial time (p.p.t.) adversary which gets oracle access to either the Left oracle or the Right oracle can distinguish between the two. Here, the Left (resp. the Right) oracle take as input a pair of messages $(m_L, m_R) \in \mathcal{M}^2$ and outputs an encryption of m_L (resp. m_R).

Private-Key Encryption from LWE.

- $\mathsf{Gen}(1^\lambda)$: Compute $n = n(\lambda)$, $q = q(\lambda)$ and $\chi = \chi(\lambda)$ in a way we will describe later in this lecture. Let the private key sk be a uniformly random vector

  $$sk := \mathbf{s} \leftarrow \mathbb{Z}_q^n \ .$$

- $\mathsf{Enc}(sk, m)$: We will work with the message space $\mathcal{M} := \{0, 1\}$. Larger message spaces can be handled by encrypting each bit of the message independently. The ciphertext is

  $$c := (\mathbf{a}, b) := (\mathbf{a}, \mathbf{s}^T\mathbf{a} + e + m\lfloor q/2 \rceil \bmod q)$$

  where $\mathbf{a} \leftarrow \mathbb{Z}_q^n$ and $e \leftarrow \chi$ is chosen from the LWE error distribution.

- $\mathsf{Dec}(sk, c = (\mathbf{a}, b))$: Output 0 if

  $$\left|b - \mathbf{s}^T\mathbf{a} \bmod q\right| < q/4$$

  and 1 otherwise.

Lemma 3. *The scheme above is correct if the support of the error distribution* $\mathsf{Supp}(\chi) \subseteq (-q/4, q/4)$ *and CPA-secure under the LWE assumption* $\mathsf{LWE}(n, m = \mathsf{poly}(n), q, \chi)$.

Correctness and security are immediate and left as an exercise to the reader.

We left the issue of how to pick n, q and χ open, and indeed, they need to be chosen appropriately for the scheme to be secure. Correctness and security give us constraints on these parameters (see Lemma 3 above), but do not tell us how to completely specify them. To fully specify the parameters, we need to ensure security against attackers "running in 2^λ time" (this is the meaning of the security parameter λ that we will use throughout this course) and to do that, we need to evaluate the efficacy of various attacks on LWE which we will do (at least, asymptotically) in the next lecture.

Open Problem 1.1**.** Construct a *nice* private-key encryption scheme from the hardness of SIS.

Note that SIS implies a one-way function directly. Together with generic transformations in cryptography from one-way functions to pseudorandom generators (Håstad-Impagliazzo-Levin-Luby) and from pseudorandom generators to pseudorandom functions (Goldreich-Goldwasser-Micali) and from pseudorandom functions to private-key encryption (easy/folklore), this is possible. The problem is to avoid the ugliness that results from using these general transformations.

Public-Key Encryption

A public-key encryption scheme is the same as private-key encryption except for two changes: first, the key generation algorithm Gen outputs a public key pk as well as a private key sk; and second, the encryption algorithm requires only the public key pk to encrypt. Security requires that a p.p.t. adversary which is given pk (and thus can encrypt as many messages as it wants on its own) cannot distinguish between an encryption of any two messages $m_0, m_1 \in \mathcal{M}$ of its choice.

Public-Key Encryption from LWE (the LPR Scheme) There are many ways of doing this; we will present the cleanest one due to Lyubashevsky-Peikert-Regev.

- $\mathsf{Gen}(1^\lambda)$: Compute $n = n(\lambda)$, $q = q(\lambda)$ and $\chi = \chi(\lambda)$ in a way we will describe later in this lecture. Let the private key sk be a random vector $sk := \mathbf{s} \leftarrow \chi^n$ is <u>chosen from the error distribution</u> and the public key is
$$pk := (\mathbf{A}, \mathbf{y}^T := \mathbf{s}^T\mathbf{A} + \mathbf{e}^T) \in \mathbb{Z}_q^{n\times n} \times \mathbb{Z}_q^n$$
where $\mathbf{A}$ is a uniformly random n-by-n matrix and $\mathbf{e} \leftarrow \chi^n$ is chosen from the error distribution.

- $\mathsf{Enc}(sk, m)$: We will work with the message space $\mathcal{M} := \{0, 1\}$ as above. The ciphertext is
$$c := (\mathbf{a}, b) := (\mathbf{A}\mathbf{r} + \mathbf{x}, \mathbf{y}^T\mathbf{r} + x' + m\lfloor q/2 \rceil \bmod q)$$
where $\mathbf{r}, \mathbf{x} \leftarrow \chi^n$ and $x' \leftarrow \chi$ are chosen from the LWE error distribution.

- $\mathsf{Dec}(sk, c = (\mathbf{a}, b))$: Output 0 if
$$\left|b - \mathbf{s}^T\mathbf{a} \bmod q\right| < q/4$$
and 1 otherwise.

Lemma 4. *The scheme above is correct if* $\mathsf{Supp}(\chi) \subseteq (-\sqrt{q/4(2n+1)}, \sqrt{q/4(2n+1)})$ *and CPA-secure under the LWE assumption* $\mathsf{LWE}(n, m = 2(n+1), q, \chi)$.

Proof. For correctness, note that the decryption algorithm computes

$$b - \mathbf{s}^T\mathbf{a} \bmod q = \mathbf{s}^T\mathbf{x} + \mathbf{e}^T\mathbf{r} + x'$$

whose absolute value, as long as $\mathsf{Supp}(\chi) \subseteq (-\sqrt{q/4(2n+1)}, \sqrt{q/4(2n+1)})$ is at most

$$q/4(2n+1) \cdot (2n+1) = q/4 \ .$$

For security, we proceed by the following sequence of hybrid experiments.

Hybrid $0.m$**.** The adversary gets pk and $\mathsf{Enc}(pk, m)$ where $m \in \{0, 1\}$.

Hybrid $1.m$**.** Feed the adveresary with a "fake" public key $\widetilde{pk}$ computed as

$$\widetilde{pk} = (\mathbf{A}, \mathbf{y}) \leftarrow \mathbb{Z}_q^{n \times n} \times \mathbb{Z}_q^n$$

and $\mathsf{Enc}(\widetilde{pk}, m)$. This is indistinguishable from Hybrid 0 by the hardness of $\mathsf{ssLWE}(n, n, q, \chi)$ and therefore, by Lemma 2, $\mathsf{LWE}(n, 2n, q, \chi)$.

Hybrid $2.m$**.** Feed the adversary with $\widetilde{pk}$ and $\widetilde{\mathsf{Enc}}(\widetilde{pk}, m)$ computed as

$$\widetilde{\mathsf{Enc}}(\widetilde{pk}, m) = (\mathbf{a}, b' + m\lfloor q/2 \rceil \bmod q)$$

where $\mathbf{a} \leftarrow \mathbb{Z}_q^n$ is uniformly random. This is indistinguishable from Hybrid 1 by $\mathsf{ssLWE}(n, n+1, q, \chi)$ or by Lemma 2, $\mathsf{LWE}(n, 2n+1, q, \chi)$, since the entire ciphertext can easily be rewritten as

$$\begin{pmatrix} \mathbf{A} \\ \mathbf{y}^T \end{pmatrix} \mathbf{r} + \begin{pmatrix} \mathbf{x} \\ x' \end{pmatrix} + \begin{pmatrix} 0 \\ m\lfloor q/2 \rceil \end{pmatrix} \bmod q$$

which, since $\mathbf{y}$ is now uniformly random, is $n+1$ ssLWE samples and therefore can be indistinguishably replaced by

$$\begin{pmatrix} \mathbf{a} \\ b' \end{pmatrix} + \begin{pmatrix} 0 \\ m\lfloor q/2 \rceil \end{pmatrix} \bmod q$$

where $\mathbf{a} \leftarrow \mathbb{Z}_q^n$ and $\mathbf{b}' \leftarrow \mathbb{Z}_q$.

Hybrid $3.m$**.** Feed the adversary with uniformly random numbers from the appropriate domains. Follows from the previous expression for the fake ciphertext (random + anything = random).

For every $m \in \mathcal{M}$, Hybrid $0.m$ is computationally indistinguishable from Hybrid $3.m$. Furthermore, Hybrid 3 is completely independent of m. Therefore, Hybrids 0.0 and 0.1 are computationally indistinguishable from each other, establishing semantic security or CPA-security.

□

There are many ways to improve the *rate* of this encryption scheme, that is, lower the ratio of (#bits in ciphertext)/(#bits in plaintext) and indeed, even achieve a rate close to 1. We can also use these techniques as building blocks to construct several other cryptographic systems such as oblivious transfer protocols. This public-key encryption scheme has its origins in earlier works of Ajtai and Dwork (1997) and Regev (2004).

Public-Key Encryption from LWE (the Regev Scheme) We present a second public-key encryption scheme due to Regev. We will only provide a sketch of the correctness and security analysis and leave it as an exercise to the reader. We remark that the security proof relies on a beautiful lemma called the "leftover hash lemma" (Impagliazzo, Levin and Luby 1990).

- $\mathsf{Gen}(1^\lambda)$: Compute $n = n(\lambda)$, $q = q(\lambda)$ and $\chi = \chi(\lambda)$ in a way we will describe later in this lecture. Let the private key sk be a random vector $sk := \mathbf{s} \leftarrow \mathbb{Z}_q^n$ is *chosen uniformly at random from Z_q* and the public key is

  $$pk := (\mathbf{A}, \mathbf{y}^T := \mathbf{s}^T\mathbf{A} + \mathbf{e}^T) \in \mathbb{Z}_q^{n \times n} \times \mathbb{Z}_q^m$$

  where $\mathbf{A}$ is a uniformly random n-by-m matrix and $\mathbf{e} \leftarrow \chi^n$ is chosen from the error distribution. Here $m = \Omega(n \log q)$.

  Note the difference from LPR where the secret key had small entries. Note also that the matrix $\mathbf{A}$ is somewhat larger than in LPR.

- $\mathsf{Enc}(sk, m)$: We will work with the message space $\mathcal{M} := \{0, 1\}$ as above. The ciphertext is

  $$c := (\mathbf{a}, b) := (\mathbf{A}\mathbf{r}, \mathbf{y}^T\mathbf{r} + m\lfloor q/2 \rceil \bmod q)$$

  where $\mathbf{r} \leftarrow \{0, 1\}^m$. $x' \leftarrow \chi$ is chosen from the LWE error distribution.

  Note the difference from LPR where the vector $\mathbf{r}$ *was chosen from the error distribution and the first component of the ciphertext had an additive error as well. Roughly speaking, in Regev, we will argue that the first component is* statistically close to *random, whereas in LPR, we argued that it is* computationally close to random *under the decisional LWE assumption.*

- $\mathsf{Dec}(sk, c = (\mathbf{a}, b))$: Output 0 if

  $$\left| b - \mathbf{s}^T\mathbf{a} \bmod q \right| < q/4$$

  and 1 otherwise.

Decryption recovers $m\lfloor q/2 \rceil$ plus an error $\mathbf{e}^T\mathbf{r} + x'$ whose norm should be smaller than $q/4$ for the correctness of decryption. This is true as long as the support of the error distribution is $\mathsf{Supp}(\chi) \subseteq (-q/4(m+1), q/4(m+1))$.

In the security proof, we first replace the public key with a uniformly random vector relying on the LWE assumption. Once this is done, use the leftover hash lemma to argue that the ciphertext is statistically close to random.

Public-Key Encryption from LWE (the dual Regev Scheme) We present yet another public-key encryption scheme due to Gentry, Peikert and Vaikuntanathan called the "dual Regev" scheme. The nice feature of this scheme, which will turn out to be important when we get to *identity-based encryption* is that the distribution of the public key is *really* random. In other words, any string could be a possible public key in the scheme.

- $\mathsf{Gen}(1^\lambda)$: Compute $n = n(\lambda)$, $q = q(\lambda)$ and $\chi = \chi(\lambda)$ in a way we will describe later in this lecture. Let the private key sk be a random vector $sk := \mathbf{r} \leftarrow \{0,1\}^m$ is *chosen uniformly at random with* 0 *or* 1 *entries* and the public key is

  $$pk := (\mathbf{A}, \mathbf{a} := \mathbf{Ar} \in \mathbb{Z}_q^{n\times n} \times \mathbb{Z}_q^m)$$

  where $\mathbf{A}$ is a uniformly random n-by-m matrix. Here $m = \Omega(n \log q)$.

  Note the difference from Regev where the private key here seems to have a component similar to the first component of a Regev ciphertext. No wonder this is called "dual Regev".

- $\mathsf{Enc}(sk, m)$: We will work with the message space $\mathcal{M} := \{0,1\}$ as above. The ciphertext is

  $$c := (\mathbf{y}^T, b) := (\mathbf{s}^T\mathbf{A} + \mathbf{e}^T, \mathbf{s}^T\mathbf{a} + x' + m\lfloor q/2\rceil \bmod q)$$

  where $\mathbf{s} \leftarrow \mathbb{Z}_q^n$ and $\mathbf{e}^T \leftarrow \chi^m$. $x' \leftarrow \chi$ is chosen from the LWE error distribution.

- $\mathsf{Dec}(sk, c = (\mathbf{y}^T, b))$: Output 0 if

  $$\left|b - \mathbf{y}^T\mathbf{r} \bmod q\right| < q/4$$

  and 1 otherwise.

Open Problem 1.2**.** Construct a public-key encryption scheme from the hardness of LWE where the support of the error distribution χ is large, namely $[-cq, cq]$ for some constant c.

LWE with such large errors does imply a one-way function, and therefore, a private-key encryption scheme. The question therefore asks if there is a gap between the LWE parameters that gives us public-key vs private-key encryption.

References

The primary reference for the cryptographic definitions in this lecture is lecture notes by Pass and Shelat, available at this url.

CHAPTER 2

The Learning with Errors Problem: Algorithms

2.1 An algebraic Algorithm: Arora-Ge

This is an attack due to Arora and Ge. The basic idea is to view an LWE sample $(\mathbf{a}, b := \mathbf{a}^T\mathbf{s} + e)$ where $e \in S \subseteq \mathbb{Z}_q$ as a polynomial equation

$$f_{\mathbf{a},b}(\mathbf{s}) = \prod_{x \in S} (b - \mathbf{a}^T\underline{\mathbf{s}} - x) \bmod q$$

where b, $\mathbf{a}$ are known and $\mathbf{s}$ is treated as the unknown variable (denoted by the underline). Clearly, if $(\mathbf{a}, b)$ is an LWE sample, then $f_{\mathbf{a},b}(\mathbf{s}) = 0 \bmod q$, else it isn't. Solving the system of polynomial equations

$$\left\{f_{\mathbf{a}_i,b_i}(\mathbf{s}) = 0 \bmod q\right\}_{i=1}^{m}$$

of degree $|S|$ will give us the LWE secret.

This is all good except that solving systems of polynomial equations (even degree-2 equations) is NP-hard. Arora and Ge's observation is that if there are *sufficiently many* equations, one can *linearize* them and that the solution to the resulting linear system will give us the solution to the polynomial system w.h.p.

To see how to do this, note that the degree of the polynomials is $|S|$ (that is, the domain in which the error terms live) and the number of monomials is thus $\binom{n+|S|}{|S|}$. *Linearization* is the basic transformation where one substitutes each monomial by a new variable. Furthermore, if $m \gg \binom{n+|S|}{|S|}$, we have more equations than variables. To begin with, any solution to the polynomial system will be a solution to the linearized system; therefore, $\mathbf{s}$ is a solution. When m is large enough, we can also show that $\mathbf{s}$ is the *unique* solution.

Simplified Proof Intuition. For simplicity, think of S as $\{0, 1\}$ and think of $n = 1$.

Take each sample $(a, b = a \cdot s + e)$ where $e \in \{0, 1\}$ and $a, s \in \mathbb{Z}_q$. This gives us a polynomial equation

$$(b + a \cdot u) \cdot (b + a \cdot u - 1) = 0 \bmod q$$

Writing it out explicitly, we get

$$b(b-1) + (2b-1)a \cdot u + a^2 \cdot u^2 = 0 \bmod q$$

Linearizing this involves replacing u and u^2 by independent variables u_1 and u_2 giving us

$$p(a) = b(b-1) + (2b-1)a \cdot u_1 + a^2 \cdot u_2 = 0 \bmod q \tag{2.1}$$

It is tempting to argue that there are no (u_1, u_2) that satisfy this equation w.h.p. over $a \leftarrow \mathbb{Z}_q$. Indeed, suppose, there were a solution (u_1, u_2). Then, viewing this as a degree-2 equation over the variable a, we see that the probability that $p(a) = 0$ is at most $2/q$ by an invocation of Cauchy-Schwartz. However, that would be a mistake since a is not chosen independently of the coefficents of p. Indeed, $u_1 = s$ and $u_2 = s^2$ is a solution to this equation.

Instead, we proceed as follows. Substitute $b = as + e$ in equation 8.1. We get

$$\begin{aligned} p'(a) &= e(e-1) + (2e-1)(s+u_1) \cdot a + (u_2 + 2su_1 + s^2) \cdot a_2 \\ &= (2e-1)(s+u_1) \cdot a + (u_2 + 2su_1 + s^2) \cdot a_2 = 0 \bmod q \end{aligned}$$

since $e(e-1)$ is 0 by definition. (This, by the way, is easily seen to be a linearization of the polynomial $(e + a \cdot (s+u)) \cdot (e - 1 + a \cdot (s+u))$.)

Now, we can think of this as a polynomial in a with coefficients chosen independent of a, for any fixed u_1, u_2. We argue that there are no solutions with $u_1 \neq -s$. Fix a (u_1, u_2) where $u_1 \neq -s$. Then, $p'(a)$ is a non-zero polynomial in a which is 0 w.p. at most $2/q$ over the choice of a. A Chernoff and union bound now finish off the job for us.

For the full proof, see the paper of Arora and Ge.

When χ is the discrete Gaussian distribution. Let's now see what this does to $\mathsf{LWE}(n, m, q, \chi)$ where χ is a Gaussian with standard deviation s. The probability that the error parameter is less than $k \cdot s$ is $e^{-O(k^2)}$.

- We get a reasonable chance that all equations have error bounded by $k \cdot s$ if $m \cdot e^{-O(k^2)} \ll 1$.
- On the other hand, we need $m > \binom{n}{k \cdot s}$ for linearization to work.

Put together, we get an attack when $m \sim n^{\tilde{O}(s^2)}$. This is non-trivial when $s = \tilde{O}(\sqrt{n})$ which, by some (not so?) strange coincidence, defines the boundary of when the worst-case to average-case reductions (i.e., security proofs) for LWE stop working (as we will see in later lectures).

Open Problem 2.1. In the case of binary LWE (that is, LWE with 0-1 errors), Arora-Ge needs $m = \Omega(n^2)$ LWE samples. Come up with a more sample-efficient attack or prove that doing so is hard. A concrete way to demonstrate the latter would be to show that solving binary error LWE with $o(m^2)$ samples is as hard as solving the lattice (approximate) shortest vector problem.

2.2 A Combinatorial Algorithm: Blum-Kalai-Wasserman

This is an attack originally due to Blum, Kalai and Wasserman. A similar version was later discovered by Wagner.

The basic idea is to find small-weight linear combinations $\mathbf{x}_{i,j}$ of the columns of $\mathbf{A}$ that sums up to a fixed vector, say the unit vectors $\mathbf{u}_i$, that is $\mathbf{A}\mathbf{x}_{i,j} = \mathbf{u}_i \bmod q$. Once we find such vectors, we compute

$$\mathbf{b}^T\mathbf{x}_{i,j} = (\mathbf{s}^T\mathbf{A} + \mathbf{e}^T)\mathbf{x}_{i,j} = s_i + \mathbf{e}^T\mathbf{x}_{i,j} \bmod q$$

which, with many copies and averaging, gives us s_i as long as $|\mathbf{e}^T\mathbf{x}_{i,j}| \ll q$. Iterating for all $i \in [n]$ gives us $\mathbf{s}$.

In another variant, we find small-norm $\mathbf{x}_{i,j}$ such that $\mathbf{A}\mathbf{x}_{i,j} = 2^j\mathbf{e}_i \bmod q$. Upon multiplying with $\mathbf{b}$ as before, we get

$$\mathbf{b}^T\mathbf{x}_{i,j} = (\mathbf{s}^T\mathbf{A} + \mathbf{e}^T)\mathbf{x}_{i,j} = s_i 2^j + \mathbf{e}^T\mathbf{x}_{i,j} \bmod q$$

As long as $|\mathbf{e}^T\mathbf{x}_{i,j}| \ll q$, this allows us to "decode" s_i with many fewer copies, essentially $O(\log q)$ of them, using the following decoding algorithm: use $s_i 2^{\lfloor \log(q/2) \rceil} + \mathbf{e}^T\mathbf{x}_{i,j}$ to learn the least significant bit of s_i; this is possible as long as the additive error is sufficiently small; subtract the l.s.b., divide by 2, and repeat.

Back to BKW: The idea of the algorithm is to split the n rows of $\mathbf{A}$ into α groups of size $\beta := n/\alpha$ each.

- For each column $\mathbf{a}_i$ of $\mathbf{A}$ we put it into one of q^β buckets depending on what $\mathbf{a}_i[1 \dots \beta]$ is.
- Notice that the difference of any two vectors, one in bucket labeled $\mathbf{w} \in \mathbb{Z}_q^\beta$ and the other in bucket labeled $\mathbf{w} - \mathbf{v} \in \mathbb{Z}_q^\beta$, starts with $\mathbf{v}$ in the first β positions.
- This gives us many vectors whose first β locations match the target vector. The goal of the rest of the algorithm is to continue along this way while generating vectors whose $\beta \cdot i$ locations match the target, for $i \in [1 \dots \alpha]$.

The result is a linear combination with Hamming weight 2^α of the columns of $\mathbf{A}$ which sum to any given target vector. The process needs q^β vectors to begin with, by a balls-and-bins argument. Assuming the error magnitude is B, we need $2^\alpha \cdot B \ll q$ for correctness. That is,

$$\alpha \ll \log(q/B)$$

This means the sample and time complexity is roughly

$$q^\beta \gg q^{n/\log(q/B)}$$

When, say, $B = n$ and $q = n^2$, this gives us a $2^{O(n)}$-time algorithm (as opposed to the $B^n = n^{O(n)}$ that comes out of enumeration).

We remark that a more refined analysis is possible, using the fact that the linear combination of the columns of $\mathbf{A}$ can have entries larger than 1; and that the linear combination does not necessarily need to add up to 0, but only approximately so; and that the sample complexity can

Algorithm	(Some) Broken Parameter settings
Arora-Ge	$m = \Omega(n^B)$ samples+time where $\lvert\mathsf{Supp}(\chi)\rvert \le B < q$
Blum-Kalai-Wasserman	$m > q^{n/\log(q/B)}$ samples+time
Lattice Reduction	$m = \mathsf{poly}(n, \log q)$ and $q/B = \Omega(2^n)$ and $\mathsf{poly}(n, \log q)$ time

Figure 2.1: Summary of asymptotic parameter settings where attacks against LWE work.

be lowered by generating new LWE samples out of old ones, at the expense of noise growth. Some of these ideas are analyzed in Albrecht et al., Kirchner-Fouque and Lyubashevsky.

Although remarkably simple, the BKW idea has found other applications, such as in Kuperberg's sub-exponential time quantum algorithm for the dihedral hidden subgroup problem (Kuperberg) which we will see in later lectures and which, in turn, has connections to LWE.

2.3 A Geometric (Suite of) Algorithm(s): Lattice Reduction

This is an attack that follows using the LLL algorithm and (building on LLL) the BKZ algorithm that find approximately short vectors in integer lattices.

We will here use facts about integer lattices; we refer the reader to Regev's lecture notes for background on lattices and lattice algorithms.

The attacks use the fact that LWE is, at its core, a problem of finding short vectors in integer lattices. Consider the m-dimensional lattices

$$\mathcal{L} := \{\mathbf{s}^T\mathbf{A} : \mathbf{s} \in \mathbb{Z}_q^n\} \oplus \mathbb{Z}_q^n$$

and

$$\mathcal{L}_{\mathbf{y}} := \{\mathbf{s}^T\mathbf{A} : \mathbf{s} \in \mathbb{Z}_q^n\} \oplus \mathbb{Z}_q^n \oplus \{\mathbf{0}, \mathbf{y}\}$$

where $\oplus$ denotes the Minkowski sum of sets and $(\mathbf{A}, \mathbf{y} = \mathbf{s}^T\mathbf{A} + \mathbf{e}^T)$ is the presumed LWE instance.

Lets look at the case where χ is a B-bounded distribution. We argue that:

- $\mathcal{L}_{\mathbf{y}}$ has a short vector, in fact a vector of ℓ_2 norm $\tilde{O}(B)$ (where $\tilde{O}$ hides $\mathsf{poly}(m)$ factors) since $\mathbf{e} \in \mathcal{L}_y$.
- $\mathcal{L}$ does not have any short vectors. The shortest vector of $\mathcal{L}$ has ℓ_2-norm at least $q^{(m-n)/m} = q \cdot q^{-n/m}$ by a probabilistic argument. This also tells us that the second (linearly independent) shortest vector in $\mathcal{L}_{\mathbf{y}}$ has length $q \cdot q^{-n/m}$.

The LLL algorithm finds a vector of length at most $\tilde{O}(2^{m/\log m} \cdot B)$ in polynomial time. As long as this is smaller than $q \cdot q^{-n/m}$, LLL will find $\mathbf{e}$. That is, if $q/B \gg q^{n/m} \cdot 2^{m/\log m}$, LLL/BKZ is bad news for us. Optimizing for m, we get $m \sim \sqrt{n \log q}$ and thus, the attack succeeds if $q/B \gg 2^{\sqrt{n \log q}}$. Setting B to be $\mathsf{poly}(m)$, we get that the attack works if $q \gg 2^n$.

For more background on lattices, see lecture notes for Lectures 1–4 in Fall 2015 class on lattices. We will review some of this background in later lectures.

CHAPTER 3

Worst-case to Average-case Reduction for SIS

In this lecture, we will see a few basic theorems on lattices and a property called smoothing. We will then use smoothing as a tool to come up with worst-case to average-case reductions for SIS and LWE. In a nutshell, the worst-case to average-case reductions show how to transform any algorithm that solves SIS/LWE *on the average* into an algorithm that solves "approximate short vector problems" on lattices *in the worst case*.

3.1 Lattices and Minkowski's Theorem

We define the lattice $\mathcal{L}(\mathbf{B})$ and its fundamental parallelepiped as follows:

$$\mathcal{L}(\mathbf{B}) = \mathbf{B}\mathbb{Z}^n \text{ and } \mathcal{P}(\mathbf{B}) = \mathbf{B}[0,1)^n$$

As additive groups, $\mathcal{P}(\mathbf{B}) = \mathbb{R}^n/\mathcal{L}(\mathbf{B})$. (Think of the one-dimensional analogy: the torus $[0,1) = \mathbb{R}/\mathbb{Z}$.)

We define the determinant of the lattice as the volume of the fundamental paralellepiped. While the parallelepiped is itself defined by a basis and is basis-dependent, its volume is a lattice-invariant. Analytically, the determinant $\det(\mathcal{L})$ of full-rank lattices can be computed as the determinant of the basis matrix $\mathbf{B}$.

We define $\lambda_1(\mathcal{L})$ to be the length of the shortest non-zero vector of the lattice. Alternatively, and apparently somewhat more convolutedly,

$$\lambda_1(\mathcal{L}) = \inf_{r\in\mathbb{R}} \mathsf{dim}(\mathsf{Span}(\mathcal{L} \cap \mathcal{B}(\mathbf{0}, r))) \geq 1$$

where $\mathcal{B}(\mathbf{0}, r)$ denotes the ball of radius r centered at $\mathbf{0}$. In words, the smallest r such that the space generated by lattice vectors of length at most r has dimension at least 1. This naturally leads us to the definition of the i-th minimum $\lambda_i(\mathcal{L})$:

$$\lambda_i(\mathcal{L}) = \inf_{r\in\mathbb{R}} \mathsf{dim}(\mathsf{Span}(\mathcal{L} \cap \mathcal{B}(\mathbf{0}, r))) \geq i$$

Our intuition tells us that as the determinant of the lattice gets smaller, the lattice gets denser, and therefore has shorter vectors. Minkowski's theorems formalize this intuition.

Lemma 5 (Minkowski)**.** *For any rank-n lattice $\mathcal{L}$, we have*

- $\lambda_1(\mathcal{L}) \leq \sqrt{n} \cdot \det(\mathcal{L})^{1/n}$*; and even stronger,*
- $\det(\mathcal{L}) \leq \prod_{i=1}^{n} \lambda_i(\mathcal{L}) \leq n^{n/2} \cdot \det(\mathcal{L})$.

Computational Problems. The γ-approximate shortest vector problem (SVP) is to find a non-zero vector $\mathbf{v} \in \mathcal{L}(\mathbf{B})$ given a basis $\mathbf{B}$ such that $||\mathbf{v}|| \leq \gamma \cdot \lambda_1(\mathcal{L})$. The γ-approximate shortest independent vectors problem (SIVP) is to find n linearly indepedent vectors $\mathbf{v}_1, \ldots, \mathbf{v}_n$ such that $||\mathbf{v}_i|| \leq \gamma \cdot \lambda_n(\mathcal{L})$.

3.2 Lattice Smoothing

Lattice Duality

For a rank-n lattice $\mathcal{L}$, its dual denoted $\mathcal{L}^*$ is defined as

$$\mathcal{L}^* = \{\mathbf{x} \in \mathbb{R}^n : \forall \mathbf{y} \in \mathcal{L}, \langle \mathbf{x}, \mathbf{y} \rangle \in \mathbb{Z}\}$$

Indeed, each dual lattice vector $\mathbf{x}$ corresponds to a linear function $\phi_{\mathbf{x}} : \mathcal{L} \to \mathbb{Z}$ and the dual lattice corresponds to a basis of the space of such linear functions.

Let us start with examples and some properties.

- In one dimension, the only possible lattices are $k\mathbb{Z}$. Its dual is $(1/k) \cdot \mathbb{Z}$.
- The dual of $\mathbb{Z}^n$ is $\mathbb{Z}^n$ itself.
- If $\mathcal{L} = \mathcal{L}(\mathbf{B})$ for a basis matrix $\mathbf{B} \in \mathbb{R}^{n \times n}$, then $\mathcal{L}^*$ is generated by the columns of $\mathbf{B}^{-T}$, its transposed inverse. Indeed, the pairwise inner products of the basis vectors and the dual basis vectors is captured in the matrix $\mathbf{B} \cdot (\mathbf{B}^{-T})^T = \mathbf{I}$.

The determinant of the dual lattice is immediately seen to be the inverse of the determinant of the lattice. In an intuitive sense, as the lattice gets sparser (the determinant gets larger), the dual lattice gets denser (its determinant gets smaller). This leads us to the following lemma.

Lemma 6. *For any rank-n lattice $\mathcal{L}$, $\lambda_1(\mathcal{L}^*) \cdot \lambda_1(\mathcal{L}) \leq n$.*

Proof. We know from Minkowski that

$$\lambda_1(\mathcal{L}) \leq \sqrt{n}(\det(\mathcal{L}))^{1/n} \text{ and } \lambda_1(\mathcal{L}^*) \leq \sqrt{n}(\det(\mathcal{L}^*))^{1/n}$$

Multiplying the two, we get

$$\lambda_1(\mathcal{L}) \cdot \lambda_1(\mathcal{L}^*) \leq n$$

as desired. $\square$

In fact, using far more advanced tools, we can show something stronger, namely that $\lambda_1(\mathcal{L}^*) \cdot \lambda_n(\mathcal{L}) \leq n$. The following lemma goes in the other direction, has an elementary proof, and we will find it useful later on.

Lemma 7. *For any rank-n lattice $\mathcal{L}$, $\lambda_n(\mathcal{L}^*) \cdot \lambda_1(\mathcal{L}) \geq 1$.*

Proof. Let $\mathbf{x} \in \mathcal{L}$ be the shortest non-zero vector. Let $\mathbf{v}_1, \ldots, \mathbf{v}_n \in \mathcal{L}^*$ be linearly independent. At least one of the $\mathbf{v}_i$ has a non-zero inner product with $\mathbf{x}$, say $\langle \mathbf{v}_i, \mathbf{x} \rangle > 0$. Since the inner products of lattice vectors and dual vectors are integers, $\langle \mathbf{v}_i, \mathbf{x} \rangle \geq 1$. Therefore,

$$\lambda_1(\mathcal{L}) = ||\mathbf{x}|| \geq 1/||\mathbf{v}_i|| \geq 1/\lambda_n(\mathcal{L}^*) \ .$$

□

Gaussians

The Gaussian function over $\mathbb{R}$ with (zero mean and) parameter s is defined as

$$\rho_s(x) = e^{-\pi x^2/s^2}$$

We note that

$$\int_{-\infty}^{\infty} \rho_s(x)dx = \int_{-\infty}^{\infty} e^{-\pi x^2/s^2} dx = \frac{s}{\sqrt{\pi}} \int_{-\infty}^{\infty} e^{-z^2} dz = s$$

where the second equality is by a change of variables and the third by using value of the Gaussian integral ($= \sqrt{\pi}$). This fact can be used to turn the Gaussian function into a probability distribution over the reals by scaling ρ_s by $1/s$.

Something very similar can be done in n dimensions. That is, the n-dimensional Gaussian function over $\mathbb{R}^n$ is defined as

$$\rho_s(\mathbf{x}) = e^{-\pi ||\mathbf{x}||^2/s^2}$$

This can again be turned into a probability distribution after scaling by $1/s^n$.

Basic Fourier Analysis

We call a function $f : \mathbb{R}^n \to \mathbb{C}$ "nice" if it is absolutely integrable, that is, $\int_{\mathbb{R}^n} |f(\mathbf{x})| d\mathbf{x} < \infty$.

Definition 8 (Fourier Transform)**.** *For a nice function $f : \mathbb{R}^n \to \mathbb{C}$, we define its Fourier transform $\hat{f} : \mathbb{R}^n \to \mathbb{C}$ as*

$$\hat{f}(\mathbf{y}) = \int_{\mathbb{R}^n} f(\mathbf{x}) e^{-2\pi i \langle \mathbf{x}, \mathbf{y} \rangle} d\mathbf{x}$$

If $f, \hat{f}$ are nice and f is continuous, we can recover a function from its Fourier transform using the inverse formula:

$$f(\mathbf{x}) = \int_{\mathbb{R}^n} f(\mathbf{y}) e^{2\pi i \langle \mathbf{x}, \mathbf{y} \rangle} d\mathbf{x}$$

Lemma 9 (Fourier Transform of the Gaussian function)**.** *Let $\hat{\rho}_s$ denote the Fourier transform of the Gaussian function ρ_s. Then,*

$$\hat{\rho}_s(\mathbf{x}) = s^n \cdot \rho_{1/s}(\mathbf{x})$$

Proof. We provide a proof in one dimension.

$$\begin{aligned}\hat{\rho}_s(\mathbf{y}) &= \int_{\mathbb{R}^n} \rho_s(\mathbf{x}) e^{-2\pi i \langle \mathbf{x}, \mathbf{y} \rangle} d\mathbf{x} \\ &= \int_{\mathbb{R}^n} e^{-\pi ||\mathbf{x}||^2/s^2} e^{-2\pi i \langle \mathbf{x}, \mathbf{y} \rangle} d\mathbf{x} \\ &= e^{-\pi s^2 ||\mathbf{y}||^2} \int_{\mathbb{R}^n} e^{-\pi ||(\mathbf{x}/s + is\mathbf{y})||^2} d\mathbf{x}\end{aligned}$$

The latter integral, on a complex change of variables becomes $s^n \cdot \int_{\mathbb{R}^n} e^{-\pi ||\mathbf{z}||^2} d\mathbf{z}$ which is simply s^n. So,

$$\hat{\rho}_s(\mathbf{y}) = s^n e^{-\pi s^2 ||\mathbf{y}||^2} = s^n \cdot \rho_{1/s}(\mathbf{y})$$

$\square$

For periodic functions, we have the closely related notion of Fourier series.

Definition 10 (Fourier Series)**.** *We will define Fourier series for* periodic *functions. For a "nice enough" function $f : \mathbb{R}^n \to \mathbb{C}$ that is $\mathcal{L}$-periodic, that is, $f(\mathbf{x}+\mathbf{y}) = f(\mathbf{x})$ for all $\mathbf{x} \in \mathbb{R}^n$ and $\mathbf{y} \in \mathcal{L}$, we have its Fourier series $\hat{f} : \mathcal{L}^* \to \mathbb{C}$ defined as*

$$\hat{f}(\mathbf{y}) = \frac{1}{\det(\mathcal{L})} \cdot \int_{\mathcal{P}(\mathcal{L})} f(x) e^{-2\pi i \langle \mathbf{x}, \mathbf{y} \rangle} d\mathbf{x}$$

We will state the Fourier inversion formula below without proof.

Lemma 11 (Fourier Inversion)**.** $f(\mathbf{x}) = \sum_{\mathbf{y} \in \mathcal{L}^*} \hat{f}(\mathbf{y}) e^{2\pi i \langle \mathbf{x}, \mathbf{y} \rangle}$.

An important fact that connects a function f and its Fourier transform is the Poisson Summation formula. The proof of this formula goes via the Fourier series.

Lemma 12 (Poisson Summation)**.** *Given $f : \mathbb{R}^n \to \mathbb{C}$, and any full-rank lattice $\mathcal{L}$, we have*

$$\sum_{\mathbf{x} \in \mathcal{L}} f(\mathbf{x}) = \frac{1}{\det(\mathcal{L})} \cdot \sum_{\mathbf{y} \in \mathcal{L}^*} \hat{f}(\mathbf{y}) = \det(\mathcal{L}^*) \cdot \sum_{\mathbf{y} \in \mathcal{L}^*} \hat{f}(\mathbf{y})$$

Proof. Although f is not periodic, the proof of Poisson summation goes through the Fourier series of a "periodized" f. In particular, consider the function

$$\phi(\mathbf{x}) = \sum_{\mathbf{z} \in \mathcal{L}} f(\mathbf{x} + \mathbf{z})$$

Clearly ϕ is periodic over $\mathcal{L}$, therefore $\hat{\phi}$ is defined over $\mathcal{L}^*$. For any $\mathbf{y} \in \mathcal{L}^*$, we have

$$\begin{aligned}
\hat{\phi}(\mathbf{y}) &= \det(\mathcal{L}^*) \int_{\mathbf{x}\in\mathcal{P}(\mathcal{L})} \phi(\mathbf{x}) e^{-2\pi i \langle \mathbf{x}, \mathbf{y} \rangle} d\mathbf{x} \\
&= \det(\mathcal{L}^*) \int_{\mathbf{x}\in\mathcal{P}(\mathcal{L})} \left(\sum_{\mathbf{z}\in\mathcal{L}} f(\mathbf{x}+\mathbf{z}) \right) e^{-2\pi i \langle \mathbf{x}, \mathbf{y} \rangle} d\mathbf{x} \\
&= \det(\mathcal{L}^*) \sum_{\mathbf{z}\in\mathcal{L}} \int_{\mathbf{x}\in\mathcal{P}(\mathcal{L})} f(\mathbf{x}+\mathbf{z}) e^{-2\pi i \langle \mathbf{x}, \mathbf{y} \rangle} d\mathbf{x} \\
&= \det(\mathcal{L}^*) \sum_{\mathbf{z}\in\mathcal{L}} \int_{\mathbf{x}\in\mathcal{P}(\mathcal{L})} f(\mathbf{x}+\mathbf{z}) e^{-2\pi i \langle \mathbf{x}+\mathbf{z}, \mathbf{y} \rangle} d\mathbf{x} \\
&= \det(\mathcal{L}^*) \int_{\mathbf{x}\in\mathbb{R}^n} f(\mathbf{x}) e^{-2\pi i \langle \mathbf{x}, \mathbf{y} \rangle} d\mathbf{x} \\
&= \det(\mathcal{L}^*) \hat{f}(\mathbf{y})
\end{aligned}$$

where the first equality used the definition of the Fourier series for ϕ, the second used the definition of ϕ, the third used the "niceness" of f to switch the integral and summation, the fourth used the fact that $\langle \mathbf{y}, \mathbf{z} \rangle \in \mathbb{Z}$, and the final one used the definition of the Fourier transform of f.

Now use Fourier inversion for ϕ to show that

$$\sum_{\mathbf{x}\in\mathcal{L}} f(\mathbf{x}) = \phi(\mathbf{0}) = \sum_{\mathbf{y}\in\mathcal{L}^*} \hat{\phi}(\mathbf{y}) = \det(\mathcal{L}^*) \sum_{\mathbf{y}\in\mathcal{L}^*} \hat{f}(\mathbf{y})$$

□

Smoothing Lemma and Proof

Let ϕ_s denote the distribution obtained by picking a vector from the (continuous) Gaussian distribution defined by ρ_s and reducing it modulo the parallelepiped $\mathcal{P}(\mathbf{B})$. Thus,

$$\phi_s(\mathbf{x}) = 1/s^n \cdot \sum_{\mathbf{y}\in\mathcal{L}(\mathbf{B})} \rho_s(\mathbf{x}+\mathbf{y}) := 1/s^n \cdot \rho_s(\mathbf{x}+\mathcal{L}(\mathbf{B}))$$

Now, since ϕ_s is clearly a periodic function over the lattice $\mathcal{L}(\mathbf{B})$, we can compute it alternatively using the Poisson summation formula. For any $\mathbf{x} \in \mathcal{P}(\mathbf{B})$, we have

$$\begin{aligned}
\phi_s(\mathbf{x}) &= \sum_{\mathbf{y}\in\mathcal{L}^*} \hat{\phi}_s(\mathbf{y}) e^{2\pi i \langle \mathbf{x}, \mathbf{y} \rangle} \\
&= \det(\mathcal{L}^*) \cdot (1/s^n) \sum_{\mathbf{y}\in\mathcal{L}^*} \hat{\rho}_s(\mathbf{y}) e^{2\pi i \langle \mathbf{x}, \mathbf{y} \rangle} \\
&= s^n \cdot \det(\mathcal{L}^*) \cdot (1/s^n) \sum_{\mathbf{y}\in\mathcal{L}^*} \rho_{1/s}(\mathbf{y}) e^{2\pi i \langle \mathbf{x}, \mathbf{y} \rangle} \\
&= \det(\mathcal{L}^*) \cdot \left(1 + \sum_{\mathbf{y}\in\mathcal{L}^*\setminus\{\mathbf{0}\}} \rho_{1/s}(\mathbf{y}) \cdot e^{2\pi i \langle \mathbf{y}, \mathbf{x} \rangle} \right)
\end{aligned}$$

where the first equality is by the definition of Fourier inversion, the second by the definition of ϕ_s and by the linearity of the Fourier transform, the third by the Fourier transform of the Gaussian function (Lemma 9), and the final one just by grouping terms together.

We will use this formulation to compute the statistical distance of ϕ_s from the uniform distribution over the paralellepiped whose density function is $U_{\mathcal{P}(\mathbf{B})}(\mathbf{x}) = 1/\det(\mathcal{L}) = \det(\mathcal{L}^*)$.

$$
\begin{aligned}
\Delta(\phi_s, U_{\mathcal{P}(\mathbf{B})}) &= \int_{\mathcal{P}(\mathbf{B})} |\phi_s(\mathbf{x}) - U_{\mathcal{P}(\mathbf{B})}(\mathbf{x})| \, d\mathbf{x} \\
&= \det(\mathcal{L}^*) \int_{\mathcal{P}(\mathbf{B})} \left| \sum_{\mathbf{y} \in \mathcal{L}^* \setminus \{\mathbf{0}\}} \rho_{1/s}(\mathbf{y}) \cdot e^{2\pi i \langle \mathbf{y}, \mathbf{x} \rangle} \right| d\mathbf{x} \\
&= \det(\mathcal{L}^*) \cdot \det(\mathcal{L}) \cdot \max_{\mathbf{x} \in \mathcal{P}(\mathbf{B})} \left| \sum_{\mathbf{y} \in \mathcal{L}^* \setminus \{\mathbf{0}\}} \rho_{1/s}(\mathbf{y}) \cdot e^{2\pi i \langle \mathbf{y}, \mathbf{x} \rangle} \right| \\
&\leq \sum_{\mathbf{y} \in \mathcal{L}^* \setminus \{\mathbf{0}\}} \rho_{1/s}(\mathbf{y}) := \rho_{1/s}(\mathcal{L}^* \setminus \{\mathbf{0}\}) \qquad (3.1)
\end{aligned}
$$

In other words, we established $\rho_{1/s}(\mathcal{L}^* \setminus \{\mathbf{0}\})$ as the quantity that governs the variation (or statistical) distance between the continuous Gaussian reduced modulo $\mathcal{P}(\mathbf{B})$ and the uniform distribution over $\mathcal{P}(\mathbf{B})$. We will now bound this quantity.

Bounding the Gaussian Weight of Non-Zero (Dual) Lattice Vectors. Let us first try to build some intuition for why we should expect to bound the Gaussian weight $\rho_{1/s}(\mathcal{L}^*)$ by something close to 1. First of all, the heaviest vector is the zero vector that gets a weight of 1. Secondly, if $\lambda_1(\mathcal{L}^*) \gtrapprox (1/s\sqrt{2\pi}) \cdot \omega(\sqrt{\log n})$, then the next heaviest vector has weight $e^{-\omega(\log n)}$ which is negligible in n. However, there could be exponentially many vectors of that length which could make the collective contribution much larger. We have to balance these two effects: the fact that a large λ_1 results in the Gaussian weight of each individual non-zero lattice vector to be tiny, versus the fact that there may be exponentially many lattice vectors of a given length.

First, let us come up with a simple upper bound on the number of lattice vectors of a given length using a packing argument.

Lemma 13. *Let $\mathcal{L}$ be a rank-n lattice. The number of lattice vectors of length at most r is at most* $\left(1 + \frac{2r}{\lambda_1(\mathcal{L})}\right)^n$.

Proof. Draw balls of radius $\lambda_1/2$ around each lattice point. These balls do not intersect. As long as the length of each such lattice point is at most r, these balls are all contained in the ball of radius $r + \lambda_1/2$ around the origin. By a volume argument, we have

$$\mathsf{vol}_n(r + \frac{\lambda_1}{2}) \geq N_r \cdot \mathsf{vol}_n(\frac{\lambda_1}{2})$$

where N_r is the number of lattice vectors of length at most r. Put together, we get

$$N_r \leq \frac{\mathsf{vol}_n(r + \frac{\lambda_1}{2})}{\mathsf{vol}_n(\frac{\lambda_1}{2})} = \left(\frac{r + \frac{\lambda_1}{2}}{\frac{\lambda_1}{2}}\right)^n = \left(1 + \frac{2r}{\lambda_1(\mathcal{L})}\right)^n$$

□

We now use this to bound the sum $\sum_{\mathbf{y}\in\mathcal{L}} \rho_s(\mathbf{y})$. The proof is due to Noah Stephens-Davidowitz.

Lemma 14. *Let $\mathcal{L}$ be a rank-n lattice. $\sum_{\mathbf{y}\in\mathcal{L}} \rho_s(\mathbf{y}) = 1 + 2^{-O(n)}$ as long as $\lambda_1 > Cs \cdot \sqrt{n/2\pi e}$ for some absolute constant $C \approx 3$.*

Proof. Using a "Lebesgue integral trick" (mentioned in class), we have

$$\begin{aligned}\sum_{\mathbf{y}\in\mathcal{L}} \rho_s(\mathbf{y}) &= \int_0^1 \Big|\{\mathbf{y}\in\mathcal{L} : \rho_s(\mathbf{y}) \geq t\}\Big| dt \\ &= \int_0^1 \Big|\{\mathbf{y}\in\mathcal{L} : e^{-\pi\|\mathbf{y}\|^2/s^2} \geq t\}\Big| dt\end{aligned}$$

Now, we do a change of variables $t = e^{-\pi r^2/s^2}$, we get:

$$\begin{aligned}&= \frac{2\pi}{s^2}\int_0^\infty \Big|\{\mathbf{y}\in\mathcal{L} : \|\mathbf{y}\| \leq r\}\Big| re^{-\pi r^2/s^2} dr \\ &\leq \frac{2\pi}{s^2}\cdot (\int_0^{\lambda_1} + \int_{\lambda_1}^\infty)\Big|\{\mathbf{y}\in\mathcal{L} : \|\mathbf{y}\| \leq r\}\Big| re^{-\pi r^2/s^2} dr \\ &\leq (1 - e^{-\pi\lambda_1^2/s^2}) + \frac{2\pi}{s^2}\int_{\lambda_1}^\infty \Big|\{\mathbf{y}\in\mathcal{L} : \|\mathbf{y}\| \leq r\}\Big| re^{-\pi r^2/s^2} dr \\ &\leq 1 + \frac{2\pi}{s^2}\int_{\lambda_1}^\infty \left(\frac{3r}{\lambda_1}\right)^n re^{-\pi r^2/s^2} dr \\ &\leq 1 + \frac{2\pi C^n}{s^2\lambda_1^n}\int_{\lambda_1}^\infty r^{n+1} e^{-\pi r^2/s^2} dr\end{aligned}$$

where $C = 3$. After another change of variables ($w = \pi r^2/s^2$), we can bound this by

$$1 + \left(\frac{sC}{\lambda_1\sqrt{\pi}}\right)^n \Gamma(n/2)$$

where $\Gamma(\cdot)$ is the gamma function. Applying the bound on gamma functions, we get

$$1 + \left(\frac{sC}{\lambda_1}\sqrt{\frac{n}{2\pi e}}\right)^n$$

As long as $\lambda_1 > Cs \cdot \sqrt{n/2\pi e}$, we get a sum that is exponentially close to 1. □

Finally, applying this to our scenario, where the lattice is $\mathcal{L}^*$ and the function is $\rho_{1/s}$, we get that the sum $\sum_{\mathbf{y}\in\mathcal{L}^*} \rho_{1/s}(\mathbf{y})$ is exponentially close to 1 as long as

$$s \geq \lambda_n(\mathcal{L}) \cdot C \cdot \sqrt{\frac{n}{2\pi e}}$$

Indeed, if s is so large, we have $\lambda_1(\mathcal{L}^*) \geq 1/\lambda_n(\mathcal{L}) \geq \frac{C}{s} \cdot \sqrt{\frac{n}{2\pi e}}$ where the first inequality is by Lemma 7.

3.3 Worst-case to Average-case Reduction for SIS

The reduction is due to Ajtai originally, but our presentation follows the work of Micciancio and Regev, and borrows from Regev's lecture notes.

We first illustrate the intuition behind the worst-case to average-case reduction by showing how to reduce the approximate-SIVP problem to a variant of SIS over the torus $\mathbb{T} = \mathbb{R}/\mathbb{Z}$, $\mathsf{SIS}_{\mathbb{T}}$. $\mathsf{SIS}_{\mathbb{T}}$ is exactly as in SIS, except that you are given a matrix $\mathbf{A} \in \mathbb{T}^{n\times m}$ and you are asked to find a small integer linear combination that sums to zero. That is, find $\mathbf{x} \in \mathbb{Z}^m$ such that $\mathbf{Ax} = 0$ and $||\mathbf{x}||$ is "small".

How would such a reduction look like? On the one hand, the reduction has to generate a *uniformly random* $\mathsf{SIS}_{\mathbb{T}}$ instance from a given lattice $\mathcal{L}$; therefore, the SIS instance "forgets" the lattice $\mathcal{L}$ that was used to generate it. On the other hand, a solution to the SIS instance has to somehow be mapped back to a non-trivially short vector in $\mathcal{L}$. This (apparent) conundrum is common to all worst-case to average-case reductions, and the answer is that the reduction knows some information connecting the lattice to the SIS instance which, together with the SIS solution, helps it generate short vectors in $\mathcal{L}$.

The reduction first generates a random vector $\mathbf{v} \in \mathcal{P}(\mathbf{B})$ in the parallelpiped associated to the given basis. It does so by sampling a vector $\mathbf{x} \leftarrow \rho_s$ from the (zero-centered) Gaussian with standard deviation parameter $s \geq \eta_\varepsilon(\mathcal{L})$, the smoothing parameter for some negligible function $\varepsilon = \varepsilon(n)$, and setting

$$\mathbf{v} = \mathbf{x} \pmod{\mathcal{P}(\mathbf{B})}$$

By the smoothing lemma, $\mathbf{v}$ is (close to) random over the paraellelepiped. The first column of the SIS matrix $\mathbf{A}$ is then set to

$$\mathbf{a} = \mathbf{B}^{-1}\mathbf{v} \in \mathbb{T}^n$$

which is (close to) random over $[0,1)^n$. Repeat this process independently m times to generate the statistically close to uniform $\mathsf{SIS}_{\mathbb{T}}$ matrix $\mathbf{A} \in \mathbb{T}^{n\times m}$ where

$$\mathbf{A} = \mathbf{B}^{-1}\mathbf{V}$$

Call the Gaussian matrix corresponding to $\mathbf{V}$ as $\mathbf{X}$. The reduction will keep $\mathbf{X}$ to itself.

Assume now that there is a $\mathsf{SIS}_{\mathbb{T}}$ algorithm that gives us a non-zero integer vector $\mathbf{x} \in \mathbb{Z}^m$ such that $\mathbf{Ax} \in \mathbb{Z}^n$. (this is what it means for $\mathbf{Ax}$ to be $\mathbf{0} \pmod 1$.) Then we know that $\mathbf{B}^{-1}\mathbf{Vx} \in \mathbb{Z}^n$ and therefore, $\mathbf{Vx} \in \mathcal{L}(\mathbf{B})$ is a lattice vector. Now, since $\mathbf{X} \equiv \mathbf{V} \pmod{\mathcal{P}(\mathbf{B})}$, we know that $\mathbf{Xx} \in \mathcal{L}(\mathbf{B})$ is also a lattice vector.

We now argue that it is short. We know that $||\mathbf{Xx}|| \approx s||\mathbf{x}||\sqrt{n} \approx \lambda_n\sqrt{mn}$. Here, the first equality is because each column of $\mathbf{X}$ is a continuous Gaussian with parameter s and therefore $\mathbf{Xx}$ has parameter $s||\mathbf{x}||$ and therefore length $s||\mathbf{x}||\sqrt{n}$ w.h.p. The second equality is using the smoothing lemma, substituting λ_n for s upto logarithmic factors and $\sqrt{m}$ as the norm of $\mathbf{x}$, assuming it is a 0-1 vector.

This seems to work, except that we are uncomfortable working with real numbers. Furthermore, it is unclear that a "random" matrix $\mathbf{A} \in \mathbb{T}^{n\times m}$ will have an SIS solution at all. We therefore discretize.

Discretization. Consider splitting each entry into a multiple of $1/q$ (for some sufficiently large value of q that we will set shortly) and an error term. That is,

$$\mathbf{A} = \mathbf{Q} + \mathbf{E} \pmod 1$$

where $q\mathbf{Q} \in \mathbb{Z}^{n \times m}$ and $||\mathbf{E}||_\infty \le 1/2q$.

Our first try is to feed the SIS algorithm with the matrix $q\mathbf{Q}$ which is uniformly random mod q. The adversary returns an $\mathbf{x}$ such that $q\mathbf{Q}\mathbf{x} = 0 \pmod q$. This gives us

$$\mathbf{0} = \mathbf{Q}\mathbf{x} = (\mathbf{A} - \mathbf{E})\mathbf{x} = \mathbf{B}^{-1}(\mathbf{V} - \mathbf{B}\mathbf{E})\mathbf{x} = \mathbf{B}^{-1}(\mathbf{X} - \mathbf{B}\mathbf{E})\mathbf{x} \pmod 1$$

and therefore, $(\mathbf{X} - \mathbf{B}\mathbf{E})\mathbf{x}$ is a lattice vector. We would, in analogy to before, show that these are short lattice vectors.

$$||\mathbf{X}\mathbf{x} - \mathbf{B}\mathbf{E}\mathbf{x}|| \le ||\mathbf{X}\mathbf{x}|| + ||\mathbf{B}\mathbf{E}\mathbf{x}|| \le s||\mathbf{x}||\sqrt{n} + \frac{||\mathbf{x}||_1}{q} \cdot \max_i ||\mathbf{b}_i||_2$$

So, this does not give us short vectors, rather it reduces the length of the longest vector in the basis by a factor of $q/||\mathbf{x}||_1 \ge q/m$ (roughly, assuming SIS produces 0-1 vectors). So, as long as $q \gg m \approx n \log q$, we get an improvement. Repeat this iteratively many times to get to roughly $s\sqrt{mn} \approx \lambda_n \sqrt{mn} \approx \lambda_n \cdot \tilde{O}(n)$.

We are stuck at solving n-approximate SIVP given a solver for SIS. Can we improve this?

Open Problem 3.1**.** Show a reduction from $\sqrt{n}$-SIVP (or better) to average-case SIS.

In the regime of exponential reductions, we show such reductions in a recent joint work with Brakerski and Stephens-Davidowitz.

Another question is to improve the values of q for which one can show SIS average-case hard.

Open Problem 3.2**.** Show a reduction from approximate SIVP to SIS with modulus $q = O(1)$.

Why do we get a non-zero vector, again? There is one important issue that we overlooked. We showed that the reduction produces a short lattice vector, but why is the vector non-zero? Relatedly, when the reduction produces many shorter vectors that form a new basis to iterate on, why do we have the guarantee that we get n linearly independent vectors from the reduction?

We will now show non-zero-ness formally, but here is the intuition: we need to think of the SIS algorithm as the adversary who is trying to send us a vector $\mathbf{x}$ which is somehow cleverly designed so that $(\mathbf{X} - \mathbf{B}\mathbf{E})\mathbf{x}$ is the zero vector. What does the SIS algorithm see? It possibly sees $\mathbf{V} = \mathbf{X} \pmod{\mathcal{P}(\mathbf{B})}$ but (a) it never sees $\mathbf{X}$ itself; and (b) given $\mathbf{V}$, there are multiple possible values of $\mathbf{X}$, which is a consequence of smoothing-type arguments. In other words, the adversary is trying to force $(\mathbf{X} - \mathbf{B}\mathbf{E})\mathbf{x}$ to be $\mathbf{0}$, but it does not know what $\mathbf{X}$ is. We then argue that information-theoretically, it cannot succeed.

We omit the formal argument, but refer the reader to Regev's lecture notes for the full proof.

Other Open Problems

Vinod finds it rather bothersome that Ajtai's reduction (and essentially every other known reduction) that demonstrates average-case hardness of SIS starts from the *SIVP* problem, rather than the more natural *SVP*. This motivates the following open problem.

Open Problem 3.3. Show a reduction from worst-case SVP to (average-case) SIS.

In fact, he would ideally like a reduction from worst-case SIS to average-case SIS, bypassing lattices altogether. Indeed, observe that solving SIS is the same as finding a short vector in a lattice (namely, the lattice $\Lambda^{\perp}(\mathbf{A}) := \{\mathbf{x} \in \mathbb{Z}^m : \mathbf{A}\mathbf{x} = 0 \pmod{q}\}$). However, even viewing through these lens, what we have demonstrated is an algorithm that finds vectors of length related to λ_n, and not λ_1. That is, if the worst-case SIS lattice has a short vector but no n linearly independent short vectors, then the reduction will miss finding the short vector (!!) We view this as a deficiency in our understanding of SIS and worst-case to average-case reductions. Therefore, a related problem is:

Open Problem 3.4. Show a reduction from worst-case SIS to average-case SIS without going through lattices.

CHAPTER 4

Worst-case to Average-case Reduction for LWE

In this lecture, we will show a worst-case to average-case reduction for LWE.

4.1 Decision to Search Reduction for LWE

The first step is to come up with a way to reduce the search version of LWE to the decision version (which is the basis of cryptographic schemes, e.g., the public-key encryption schemes we already saw in Lecture 1). Later, we will show a reduction from worst-case lattice problems to search LWE, completing the chain of reductions.

Worst-case vs. Average-case Secret

We start with the simple observation that solving LWE with a worst-case secret $\mathbf{s}$ is just as easy as solving it with a uniformly random secret $\mathbf{s}$. That is, it is easy to re-randomize the secret $\mathbf{s}$. The key observation is that $\mathbf{A}$ is public and that everything here is additive.

Indeed, given an LWE input $(\mathbf{A}, \mathbf{b}_{wc}^T := \mathbf{s}_{wc}^T\mathbf{A} + \mathbf{e}^T)$ with an arbitrary secret $\mathbf{s}_{wc}$, the re-randomization algorithm (the reduction) computes

$$\mathbf{b}^T := \mathbf{b}_{wc}^T + \mathbf{s}_r^T\mathbf{A}$$

for a uniformly random vector $\mathbf{s}_r \leftarrow \mathbb{Z}_q^n$. Now, note that

$$\mathbf{b}^T := (\mathbf{s}_{wc} + \mathbf{s}_r)^T\mathbf{A} + \mathbf{e}^T$$

which is an LWE input with the uniformly random secret $\mathbf{s} := \mathbf{s}_{wc} + \mathbf{s}_r$. Clearly, if there is an algorithm that finds $\mathbf{s}$ given $(\mathbf{A}, \mathbf{b})$, the reduction can recover $\mathbf{s}_{wc} := \mathbf{s} - \mathbf{s}_r$.

A Simple Reduction

We now show a reduction from search LWE to decisional LWE. Before we begin, a few words about average-case reductions. These are quite tricky to get right. A typical reduction solves a distinguishing problem, such as coming up with an algorithm (typically probabilistic polynomial-time) that distinguishes between two probability distributions $\mathcal{D}_0$ and $\mathcal{D}_1$. Such an algorithm is said to be a (T, ε)-distinguisher if it runs in time T and has a (distinguishing) advantage of ε:

$$|\Pr[x \leftarrow \mathcal{D}_0; \mathsf{Dist}(x) = 1] - \Pr[x \leftarrow \mathcal{D}_1; \mathsf{Dist}(x) = 1]| \leq \varepsilon$$

Equivalently,

$$1/2 - \varepsilon/2 \leq \Pr[b \leftarrow \{0,1\}; x \leftarrow \mathcal{D}_b; \mathsf{Dist}(x) = b] \leq 1/2 + \varepsilon/2$$

Theorem 15. *If there is a (T, ε)-distinguisher for decisional $\mathsf{LWE}_{n,m,q,\chi}$, then there is a time $T' = \widetilde{O}(T \cdot nq/\varepsilon^2)$-time algorithm that solves search $\mathsf{LWE}_{n,m',q,\chi}$ with probability $1 - o(1)$, where $m' = \widetilde{O}(nmq/\varepsilon^2)$, where $\widetilde{O}(\cdot)$ hides polylogarithmic factors in n.*

Proof. Our approach to solve search $\mathsf{LWE}_{n,m',q,\chi}$ will be to "guess" the secret, one coordinate at a time. Let $s_1, \ldots, s_n \in \mathbb{Z}_q$ denote the coordinates of $\mathbf{s}$, that is, $\mathbf{s} = (s_1, \ldots, s_n)$. Consider the algorithm which, on input $(\mathbf{A}, \mathbf{s}^T\mathbf{A} + \mathbf{e}^T)$, for each $i \in [m]$, guesses the i^{th} coordinate of $\mathbf{s}$ as described in Algorithm 1 below. First of all, the algorithm partitions the columns of $\mathbf{A}$ into $n \cdot q \cdot \widetilde{O}(\frac{m}{\varepsilon^2})$ parts – n for the number of coordinates of $\mathbf{s}$; q for the number of possible guesses for each coordinate; and the rest is what a single iteration of the guessing algorithm uses.

Algorithm 1 "Guess" the i^{th} coordinate of $\mathbf{s}$

For $j = 0, \ldots, q-1$:

- Let $g_i := j$.
- For $\ell = 1, \ldots, L = \widetilde{O}(1/\varepsilon^2)$:
  - Choose a fresh block of the search LWE challenge, call it $(\mathbf{A}_\ell, \mathbf{b}_\ell)$.
  - Sample a random vector $\mathbf{c}_\ell \leftarrow \mathbb{Z}_q^m$, and let $\mathbf{C}_\ell \in \mathbb{Z}_q^{n \times m}$ be the matrix whose i-th row is $\mathbf{c}_\ell$, and whose other entries are all zero.
  - Let $\mathbf{A}'_\ell := \mathbf{A}_\ell + \mathbf{C}_\ell$, and $\mathbf{b}'_\ell = \mathbf{b}_\ell + g_i \cdot \mathbf{c}_\ell$.
  - Run the distinguisher $\mathcal{D}$ on input $(\mathbf{A}'_\ell, \mathbf{b}'_\ell)$ and let the output of $\mathcal{D}$ be called d_ℓ.
- If $\mathsf{maj}(d_1, \ldots, d_L) = 1$ (meaning that the distinguisher guesses "LWE") then output g_i. Else, continue to the next iteration of the loop.

If a guess g_i is correct, i.e. $s_i = g_i$, then the inputs $(\mathbf{A}'_\ell, \mathbf{b}'_\ell)$ given to $\mathcal{D}$ are fresh LWE samples, since

$$\begin{aligned}
\mathbf{b}'_\ell = \mathbf{b}_\ell + s_i \cdot \mathbf{c}_\ell &= \mathbf{s}^T\mathbf{A}_\ell + \mathbf{e}_\ell^T + s_i \cdot \mathbf{c}_\ell^T && \text{(expanding } \mathbf{b}_\ell\text{)} \\
&= (\mathbf{s}^T\mathbf{A}_\ell + s_i \cdot \mathbf{c}_\ell^T) + \mathbf{e}_\ell^T && \text{(rearranging)} \\
&= \mathbf{s}^T(\mathbf{A}_\ell + \mathbf{C}_\ell) + \mathbf{e}_\ell^T && \text{(by construction of } \mathbf{C}_\ell\text{)} \\
&= \mathbf{s}^T\mathbf{A}'_\ell + \mathbf{e}_\ell^T. && \text{(by definition of } \mathbf{A}'_\ell\text{)}
\end{aligned}$$

On the other hand, if the guess g_i is wrong, i.e. $s_i \neq g_i$, then the inputs $(\mathbf{A}'_\ell, \mathbf{b}'_\ell)$ given to $\mathcal{D}$ are uniformly random, since

$$\begin{aligned}\mathbf{b}'_\ell = \mathbf{b}_\ell + g_i \cdot \mathbf{c}_\ell &= \mathbf{s}^T \mathbf{A}_\ell + \mathbf{e}_\ell^T + g_i \cdot \mathbf{c}_\ell^T \\ &= (\mathbf{s}^T \mathbf{A}_\ell + g_i \cdot \mathbf{c}_\ell^T) + \mathbf{e}_\ell^T \\ &= \mathbf{s}^T \mathbf{A}'_\ell + \mathbf{e}_\ell^T + (g_i - s_i) \cdot \mathbf{c}_\ell,\end{aligned}$$

and the term $(g_i - s_i) \cdot \mathbf{c}_\ell$ is random and independent of the rest of the terms since (1) $g_i - s_i$ is nonzero and we are assuming that q is prime; and (2) $\mathbf{c}_\ell$ is random and independent of $\mathbf{A}'_\ell$, $\mathbf{s}$ and $\mathbf{e}_\ell$.

It follows that $\mathcal{D}$ will output 1 with probability at least $1/2 + \varepsilon$, in the case that $s_i = g_i$. Since we run $\mathcal{D}$ many times, namely $L = c \log n / \varepsilon^2$ times (for a sufficiently large constant c), it follows from a Chernoff bound that with probability $1 - 1/n^2$: if the majority of the outputs $d_1, \ldots, d_\ell$ from $\mathcal{D}$ are equal to 1, then we are in the case where $s_i = g_i$, and if not, we are in the case where $s_i \neq g_i$.

Hence, by a union bound, with overwhelming probability, namely at least $1 - 1/n$, Algorithm 1 guesses *all* coordinates of $\mathbf{s}$ correctly. Therefore, applying Algorithm 1 to each coordinate of $\mathbf{s}$ will, with overwhelming probability, correctly output all coordinates $s_1, \ldots, s_n$ of $\mathbf{s}$. □

Improvements.

- Sample-preserving reduction of Micciancio and Mol: Achieve $m' \approx m$. The key is to use ideas from the Goldreich-Levin and Impagliazzo-Naor search to decision reductions which work with pairwise independence as opposed to full independence as we did.
- Runtime scaling with $\mathsf{poly} \log q$: A major problem with the reduction is that the runtime scales linearly with q, which could make the reduction meaningless for large $q \approx 2^n$, even when the LWE problem is likely hard, e.g., when the error has magnitude $q/\mathsf{poly}(n)$. We will sketch a modification of the above reduction which works even when q is large but of a specific form, e.g., $q = 2^k$ is a power of two, or $q = q_1 q_2 \ldots q_k$ is a product of many small primes in which case the runtime will scale with $\max_i q_i$.
- Direct reduction from worst-case by Peikert, Regev and Stephens-Davidowitz: This is more relevant in the context of Ring-LWE which we will discuss later in the course.

A Reduction with $\mathsf{poly}(\log q)$ Runtime

Assume that $q = 2^k$. We show how to make the runtime scale with k rather than 2^k. The key idea (due to Micciancio and Peikert) is to guess each number $s_i \in \mathbb{Z}_q$ (coordinate of the secret vector $\mathbf{s}$) bit by bit, rather than make one guess for every possible value of s_i.

In particular, we will modify the guessing algorithm as follows. Unlike the previous algorithm, this one will employ the following *iterative procedure* for each coordinate, to guess each bit of it in turn, starting from the least significant bit.

- Define distributions $\mathcal{D}_0, \mathcal{D}_1, \ldots, \mathcal{D}_k$ where $\mathcal{D}_i$ produces

$$(\mathbf{a}, \langle \mathbf{a}, \mathbf{s} \rangle + e + r \cdot 2^j \pmod q)$$

where, as above, $q = 2^k$ and r is uniformly random mod q. Note that $\mathcal{D}_0$ is uniformly random and $\mathcal{D}_k$ is LWE. Since the decisional LWE adversary can distinguish between $\mathcal{D}_0$ and $\mathcal{D}_k$ with a $1/\mathsf{poly}(n)$ advantage, there is a $j \in [k]$ such that it distinguishes between $\mathcal{D}_{j-1}$ and $\mathcal{D}_j$ with advantage at least $1/k \cdot 1/\mathsf{poly}(n)$. Focus on such a j.

- We will now use the distinguisher to learn the LSB of s_1 (and analogously, that of all other s_i) as follows. Given an LWE sample $(\mathbf{a}, b)$, create a sample

  $$(\mathbf{a}', b') = (\mathbf{a} + r \cdot 2^{j-1} \cdot \mathbf{u}_1, b)$$

  where $\mathbf{u}_1$ is the unit vector with 1 in the first coordinate and 0 elsewhere.

  If the LSB of s_1 is 0, then this looks like

  $$(\mathbf{a}', b' = \langle \mathbf{a}', \mathbf{s} \rangle + e + r \cdot 2^j \pmod q)$$

  where $\mathbf{a}'$ and r are uniformly random and independent. On the other hand, if the LSB of s_1 is 1, this looks like

  $$(\mathbf{a}', b' = \langle \mathbf{a}', \mathbf{s} \rangle + e + r \cdot 2^{j-1} \pmod q)$$

  where again, $\mathbf{a}'$ and r are uniformly random and independent.

  A distinguisher that tells these two apart also helps us determine the LSB of s_1 (and analogously, of all the s_i).

- We now proceed in two steps. First, we observe that this can be used to recover the successive bits of $\mathbf{s}$, up to a certain point. We first transform the given LWE sample $(\mathbf{a}, b)$ so that it corresponds to a secret whose LSBs are 0. For example, to go from predicting the LSB to the second least significant bit, we transform $(\mathbf{a}, b) \to (\mathbf{a}, b - \langle \mathbf{a}, \mathsf{LSB}(s_1) \cdot \mathbf{u}_1 \rangle)$.

  From then on, to recover the k-th least significant bit, we do:

  $$(\mathbf{a}', b') = (\mathbf{a} + r \cdot 2^{j-k} \cdot \mathbf{u}_1, b)$$

  This ends up being either $\mathcal{D}_{j-1}$ or $\mathcal{D}_j$ depending on whether the k-th LSB of s_1 is either 1 or 0 (respectively).

- However, we can only recover up to j LSBs this way. What do we do with the rest? The key idea is to make sure that j is not too small. To do this, consider the modified distributions $\mathcal{D}'_j$ which output

  $$(\mathbf{a}, b + r \cdot 2^j + e' \pmod q)$$

  where $(\mathbf{a}, b)$ is an LWE sample with noise rate αq and e' is a fresh Gaussian with noise rate about αq.

  The effect of doing this is that the distributions $\mathcal{D}'_0, \mathcal{D}'_1, \ldots, \mathcal{D}'_{2^{j'} \approx \alpha q}$ are *statistically indistinguishable*. Thus, the j in question for which the distinguisher succeeds in distinguishing $\mathcal{D}'_{j-1}$ and $\mathcal{D}'_j$ is necessarily larger than j'. This lets us recover j' LSBs of all the s_i. The remaining space has size about $q/2j' \approx 1/\alpha = \mathsf{poly}(n)$.

  To recover this part of the secret, observe the following: if we only had the MSB of the secret to recover and the error was small enough, we would be done. Indeed, b then is a multiple of

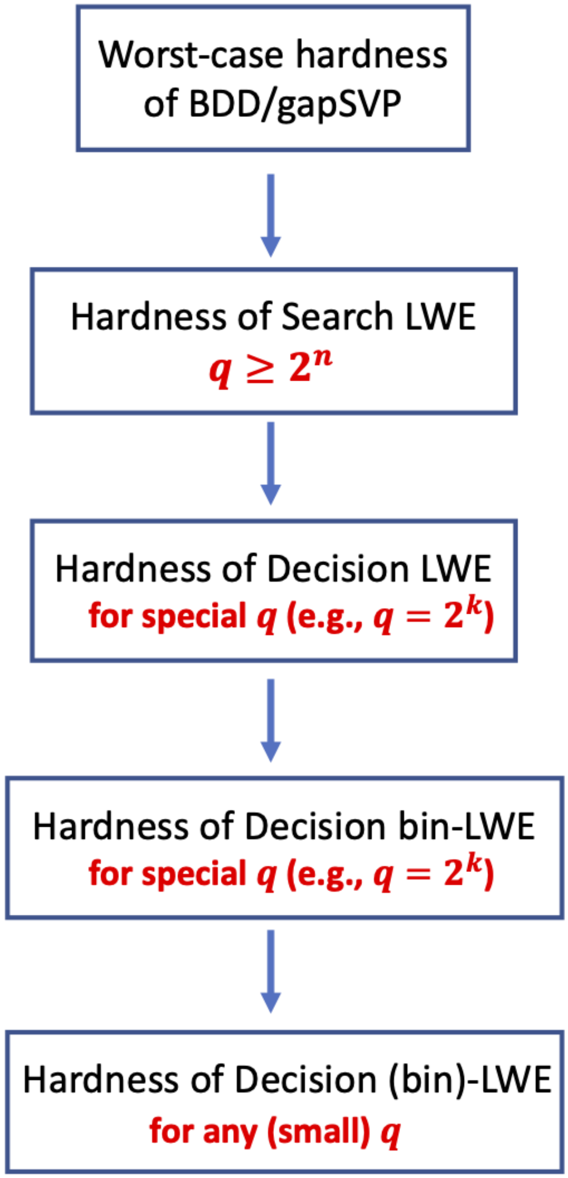


Figure 4.1: The Sequence of Reductions from Worst-case BDD/gapSVP to decision LWE for small modulus.

$q/2$ plus a small amount of noise. From this, we can recover exactly the multiple of $q/2$ which by Gaussian elimination will tell us the MSB of s_1. The key is to extend this argument to recover sufficiently many MSBs, in fact $k - j'$ of them (everything that we couldn't recover by the procedure above).

A Better Reduction: A Sketch

We will show how to reduce LWE mod q to LWE mod $p \ll q$, with a commensurate noise rate, in two steps. The (very rough) intuition is that the hardness of LWE (for a fixed n) depends on the ratio between the noise magnitude and the modulus, and not on the modulus itself. This suggests that it *should* be possible to scale q while keeping the noise-to-modulus ratio the same. We will show a (sketch of a) formal version of this intuition.

We will proceed in steps.

Idea 1. From LWE to binary secret LWE. We will use an idea of Goldwasser, Kalai, Peikert and Vaikuntanathan [GKPV10]. The rough idea is as follows: look at an LWE input $(\mathbf{A}, \mathbf{s}^T\mathbf{A} + \mathbf{e}^T)$ where $\mathbf{s} \in \{0,1\}^n$. Suppose $\mathbf{A}$ were decomposable into $\mathbf{BC}$ where $\mathbf{B} \in \mathbb{Z}_q^{n\times k}$ and $\mathbf{C} \in \mathbb{Z}_q^{k\times m}$ are uniformly random. The reader should think of $k \approx n/\log q = H_\infty(\mathbf{s})$, the min-entropy of the vector $\mathbf{s}$. Then,

$$\mathbf{s}^T\mathbf{A} + \mathbf{e}^T = \mathbf{s}^T\mathbf{BC} + \mathbf{e}^T = (\mathbf{s}^T\mathbf{B})\mathbf{C} + \mathbf{e}^T$$

In other words, one can think of this as an LWE input w.r.t. the public matrix $\mathbf{C}$ with the secret being $\mathbf{s}^T\mathbf{B}$. The key point is that multiplication by $\mathbf{B}$ extracts randomness from $\mathbf{s}$ and makes $\mathbf{s}^T\mathbf{B}$ (statistically close to) uniformly random by the leftover hash lemma (LHL). (Clealy, we are omitting details such as the slack between the min-entropy and the output length that LHL needs, but they are not very important to this outline.)

In other words, this says that the LWE input with a binary secret $\mathbf{s}$ w.r.t. $\mathbf{A}$ looks statistically close to an LWE input with a uniformly random secret $\mathbf{s}' := \mathbf{B}^T\mathbf{s}$ which, in turn, is pseudorandom. QED.

If this argument did work, it will prove the hardness of LWE where the secret comes from *any* distribution with sufficient min-entropy (eg $H_\infty(\mathbf{s})/\log q \geq \lambda$ for some security parameter λ.)

There is a major glitch in this argument, however: a matrix of the type $\mathbf{BC}$ has rank at most k, whereas a random matrix $\mathbf{A}$ has rank $n \approx k \log q$. In other words, they are *very distinguishable.*

Goldwasser et al. [GKPV10] nevertheless show how to fix this idea in the following way: assume that $\mathbf{A} = \mathbf{BC} + \mathbf{N}$ where $\mathbf{N}$ is an LWE error matrix. Such a matrix is computationally close to uniform under LWE (with the uniformly random secret matrix $\mathbf{B}$.) Now let's do the calculation again.

$$\mathbf{s}^T\mathbf{A} + \mathbf{e}^T = \mathbf{s}^T(\mathbf{BC} + \mathbf{N}) + \mathbf{e}^T = (\mathbf{s}^T\mathbf{B})\mathbf{C} + (\mathbf{s}^T\mathbf{N} + \mathbf{e}^T)$$

$\mathbf{s}^T\mathbf{B}$ is statistically close to uniform by the argument above. However, the error term is different and it raises two problems: (1) it potentially leaks information about $\mathbf{s}$, ruining the LHL; and (2) it makes the error distribution wonky. A cheap way to get around this problem is to ensure that $||\mathbf{s}^T\mathbf{N}||$ is small, say $\mathsf{poly}(n)$, for example by ensuring that $\mathbf{s}$ is binary and $\mathbf{N}$ has $\mathsf{poly}(n)$-bounded entries, and using the so-called *noise flooding* trick, setting $\mathbf{e}^T$ to be a Gaussian with a superpolynomially larger standard deviation. This ensures that $\mathbf{s}^T\mathbf{N} + \mathbf{e}^T$ looks statistically like a fresh Gaussian, independent of $\mathbf{s}^T\mathbf{N}$. This kills both problems in one shot.

Unfortunately, this means that q has to be larger than the error, ie at least $2^{\omega(\log n)}$ and one has to assume LWE where the noise-to-modulus ratio is $2^{-\omega(\log n)}$. This issue has been resolved in a subsequent work of Brakerski et al. [BLP+13]. Nevertheless, the following question is still open:

Open Problem 4.1. For which distributions of the secret $\mathbf{s}$ does the LWE assumption hold (assuming LWE with uniform secrets holds)?

The most recent development along these lines is the very recent work of Dottling and Brakerski [BD20]. A more concrete question that, to the best of the instructor's knowledge, remains open is the following:

Open Problem 4.2. Does LWE remain hard if the secret vector is a random 0-1 vector with at most $\log n$ ones?

Idea 2. **Modulus Reduction.** Now, we utilize a technique called "modulus reduction" invented by Brakerski and Vaikuntanathan [BV11] in the context of fully homomorphic encryption.

The rough idea is as follows: Assume that you are given LWE samples $(\mathbf{A}, \mathbf{b})$ with a 0-1 secret relative to a matrix $\mathbf{A}$ mod q. We would like to produce LWE samples modulo p in such a way that solving LWE mod p gives us a solution mod q. Consider computing

$$\left(\left\lfloor \frac{p}{q} \mathbf{A} \right\rceil, \left\lfloor \frac{p}{q} \mathbf{b} \right\rceil \right)$$

The matrix $\mathbf{A}' := \lfloor p/q \cdot \mathbf{A} \rceil$ is uniformly random mod p (modulo boundary issues which can be taken care of with some work.) Now,

$$(p/q)\mathbf{b} = (p/q) \cdot (\mathbf{s}^T\mathbf{A} + \mathbf{e}^T + q\mathbf{z}^T) = \mathbf{s}^T\mathbf{A}' + \mathbf{s}^T\{p/q\mathbf{A}\} + (p/q)\mathbf{e}^T + p\mathbf{z}^T$$

where $\mathbf{z}$ is some integer vector and $\{\cdot\}$ denotes the fractional part of a number (or each number in a matrix). This is almost LWE mod p. Let us analyze the error term. $(p/q)\mathbf{e}^T$ is a Gaussian with parameter αp if $\mathbf{e}$ is Gaussian with paramter αq. Assuming p is quasipolynomially large, one can use the noise-flooding lemma to "absorb" the error $\mathbf{s}^T\{p/q\mathbf{A}\}$ which has polynomially bounded norm. This completes the proof sketch.

We remark that much better versions of this gameplan has been executed successfully by Brakerski, Langlois, Peikert, Regev and Stehlë [BLP$^+$13]. We refer the reader to their paper for more details.

4.2 Bounded Distance Decoding and LWE

The bounded distance decoding (BDD) problem is a promise variant of the closest vector problem (CVP) on lattices, where the target point is guaranteed to be so close to the lattice that there is a *unique* closest vector. In other words, in the c-BDD problem for a $c \in [0, 1/2)$, one is given a basis $\mathbf{B} \in \mathbb{Z}^{m \times m}$ of a lattice $\mathcal{L}(\mathbf{B})$ and a target vector $\mathbf{t} \in \mathbb{Z}^m$ such that $\mathcal{D}(t, \mathcal{L}(\mathbf{B})) \leq c \cdot \lambda_1(\mathcal{L}(\mathbf{B}))$, and the goal is to find the lattice vector that is closest to $\mathbf{t}$.

BDD and LWE are very closely related as the reader may have noticed already. In particular, LWE can be seen as an average-case version of BDD in the following way. Define the LWE lattice

$$\Lambda(\mathbf{A}) := \{\mathbf{z} \in \mathbb{Z}^m : \exists\ \mathbf{s} \in \mathbb{Z}_q^n \text{ s.t. } \mathbf{z} = \mathbf{s}^T\mathbf{A} \pmod{q}\}$$

(Note that $q\mathbb{Z}^m \subseteq \Lambda(\mathbf{A}) \subseteq \mathbb{Z}^m$.) It is not hard to show that the minimum distance of $\Lambda(\mathbf{A})$ for a uniformly random matrix $\mathbf{A} \in \mathbb{Z}_q^{n\times m}$ is $c'q^{1-n/m}$ with high probability. (We will leave this calculation as an exercise.)

LWE is then the regime where the secret $\mathbf{s}$ (which defines the closest vector) is uniquely determined given $\mathbf{s}^T\mathbf{A} + \mathbf{e}^T$.

4.3 Discrete Gaussians

As we saw in the last lecture, the Gaussian function

$$\rho_s(\mathbf{x}) := e^{-\pi||\mathbf{x}||^2/s^2}$$

from $\mathbb{R}^n$ to $\mathbb{R}$ can be turned into a probability distribution over $\mathbb{R}^n$ by normalizing with $\int_{\mathbb{R}^n} \rho_s(\mathbf{x})d\mathbf{x} = s^n$. Henceforth, we will call this the (n-dimensional) Gaussian distribution N_s. Thus,

$$N_s(\mathbf{x}) = \frac{1}{s^n} \cdot e^{-\pi||\mathbf{x}||^2/s^2}$$

Given a lattice $\mathcal{L}$, we will define the discrete Gaussian distribution $D_{\mathcal{L},s}$ as the probability distribution that assigns the value 0 to all $\mathbf{x} \notin \mathcal{L}$ and the values

$$D_{\mathcal{L},s}(\mathbf{x}) = \frac{\rho_s(\mathbf{x})}{\rho_s(\mathcal{L})}$$

for every $\mathbf{x} \in \mathcal{L}$. Here, $\rho_s(\mathcal{L}) := \sum_{\mathbf{v}\in\mathcal{L}} \rho_s(\mathbf{v})$.

The latter definition can be generalized to any discrete set; for example, we will let $D_{\mathcal{L}+\mathbf{c},s}$ denote the discrete Gaussian over the lattice coset $\mathcal{L} + \mathbf{c} = \{\mathbf{v} + \mathbf{c} : \mathbf{v} \in \mathcal{L}\}$ which assigns the Gaussian mass (normalized appropriately) to each vector in $\mathcal{L} + \mathbf{c}$ and 0 to all other vectors.

We will also define off-centered versions of these quantities $\rho_{s,\mathbf{c}}$, $N_{s,\mathbf{c}}$ and $D_{\mathcal{L},s,\mathbf{c}}$; for example, $\rho_{s,\mathbf{c}}(\mathbf{x}) := e^{-\pi||\mathbf{x}-\mathbf{c}||^2/s^2}$, and so on.

When s exceeds the smoothing parameter of the lattice $\eta_\varepsilon(\mathcal{L})$, the discrete Gaussian over $\mathcal{L}$ starts having a number of nice regularity properties that make it behave essentially as if it were a continuous Gaussian distribution. Some examples follow.

Lemma 16. *For any $\mathbf{c} \in \mathbb{R}^n$, and $s \geq \eta_\varepsilon(\mathcal{L})$,*

$$\rho_s(\mathcal{L} + \mathbf{c}) \in [1 - 2\varepsilon, 1 + 2\varepsilon] \cdot \rho_s(\mathcal{L})$$

Proof. Let $\mathbf{c}'$ denote the shortest vector in the lattice coset $\mathcal{L} + \mathbf{c}$. Then,

$$\begin{aligned}
\rho_s(\mathcal{L} + \mathbf{c}) &= \rho_{s,-\mathbf{c}}(\mathcal{L}) \\
&= \det(\mathcal{L}^*) \cdot \widehat{\rho_{s,-\mathbf{c}}}(\mathcal{L}^*) \\
&= \det(\mathcal{L}^*) \cdot \sum_{\mathbf{z}\in\mathcal{L}^*} \widehat{\rho_{s,-\mathbf{c}}}(\mathbf{z}) \\
&= \det(\mathcal{L}^*) \cdot \sum_{\mathbf{z}\in\mathcal{L}^*} e^{2\pi i\langle \mathbf{c},\mathbf{z}\rangle} \rho_{1/s}(\mathbf{z}) \\
&= \det(\mathcal{L}^*) \cdot \left(1 + \sum_{\mathbf{z}\in\mathcal{L}^*\setminus\{\mathbf{0}\}} e^{2\pi i\langle \mathbf{c},\mathbf{z}\rangle} \rho_{1/s}(\mathbf{z})\right) \\
&\in [1 - \varepsilon, 1 + \varepsilon] \cdot \det(\mathcal{L}^*)
\end{aligned}$$

The claim follows. □

A direct corollary is the following statement about discrete Gaussians modulo sublattices. It says that if you choose a vector from a discrete Gaussian over a dense (rank n) lattice $\mathcal{L}$ and reduce it modulo a sparser (also rank n) lattice $\mathcal{L}' \subseteq \mathcal{L}$, you get a uniformly random element of the finite group $\mathcal{L}/\mathcal{L}'$. This will be instantiated later in the lecture where $\mathcal{L}$ will be an arbitrary lattice and $\mathcal{L}' = q\mathcal{L}$ will be a scaling of it. Here, $\mathcal{L}/\mathcal{L}' \cong \mathbb{Z}_q^n$.

Lemma 17 (Discrete+Continuous Convolution)**.** *Let $\mathcal{L}$ be a lattice. Consider the distribution obtained by sampling a vector $\mathbf{v}$ from the discrete Gaussian $D_{\mathcal{L},s}$ and a vector $\mathbf{w}$ from the continuous Gaussian N_r and adding them together, where $s, r \geq \eta_\varepsilon(\mathcal{L}) \cdot \sqrt{2}$ (where ε is a negligible function of n). Then, the resulting distribution is statistically close to the continuous Gaussian $N_{\sqrt{r^2+s^2}}$.*

Proof. Consider the distribution Y obtained by adding up the two vectors. Let $t = \sqrt{r^2+s^2}$.

$$\begin{aligned}
Y(\mathbf{x}) &= \sum_{\mathbf{v}\in\mathcal{L}} \Pr_{D_{\mathcal{L},s}}[\mathbf{v}] \cdot \Pr_{N_r}[\mathbf{x}-\mathbf{v}] \\
&= \frac{1}{\rho_s(\mathcal{L}) \cdot r^n} \sum_{\mathbf{v}\in\mathcal{L}} \rho_s(\mathbf{v}) \cdot \rho_r(\mathbf{x}-\mathbf{v}) \\
&= \frac{1}{\rho_s(\mathcal{L}) \cdot r^n} \sum_{\mathbf{v}\in\mathcal{L}} e^{-\pi||\mathbf{v}||^2/s^2} \cdot e^{-\pi||\mathbf{x}-\mathbf{v}||^2/r^2} \\
&= \frac{1}{\rho_s(\mathcal{L}) \cdot r^n} \sum_{\mathbf{v}\in\mathcal{L}} e^{-\pi\left(||\mathbf{v}||^2\cdot(t^2/r^2s^2) - 2\langle\mathbf{x},\mathbf{v}\rangle/r^2 + ||\mathbf{x}||^2/r^2\right)} \\
&= \frac{e^{-\pi||\mathbf{x}||^2\cdot\frac{1}{r^2}\cdot(1-\frac{s^2}{t^2})}}{\rho_s(\mathcal{L}) \cdot r^n} \sum_{\mathbf{v}\in\mathcal{L}} e^{-\pi\left(||\mathbf{v}||^2\cdot(t^2/r^2s^2) - 2\langle\mathbf{x},\mathbf{v}\rangle/r^2 + ||\mathbf{x}||^2\cdot(s^2/t^2r^2)\right)} \\
&= \frac{e^{-\pi||\mathbf{x}||^2/t^2}}{\rho_s(\mathcal{L}) \cdot r^n} \sum_{\mathbf{v}\in\mathcal{L}} e^{-\pi||\mathbf{v}-s^2/t^2\cdot\mathbf{x}||^2/(rs/t)^2} \\
&= \frac{\rho_t(\mathbf{x})}{t^n} \cdot \frac{t^n}{r^n} \cdot \frac{\rho_{rs/t, s^2/t^2\cdot\mathbf{x}}(\mathcal{L})}{\rho_s(\mathcal{L})} \\
&\in \left[1-\varepsilon, 1+\varepsilon\right] \cdot \frac{\rho_t(\mathbf{x})}{t^n} \cdot \frac{t^n}{r^n} \cdot \frac{\rho_{rs/t}(\mathcal{L})}{\rho_s(\mathcal{L})}
\end{aligned}$$

where we used Lemma 16 on the numerator since $rs/t \geq \eta_\varepsilon(\mathcal{L})$.

By Proposition 18, we have $\frac{\rho_{rs/t}(\mathcal{L})}{\rho_s(\mathcal{L})} \in [1-2\varepsilon, 1+2\varepsilon] \cdot (r/t)^n$. Put together with the above, we have

$$Y(\mathbf{x}) \in \left[1-3\varepsilon, 1+3\varepsilon\right] \cdot N_t(\mathbf{x})$$

from which it follows that the statistical distance between the two distributions in question is at most 3ε. ☐

Proposition 18. *Assume that $s_1, s_2 \geq \eta_\varepsilon(\mathcal{L})$. Then,*

$$\frac{\rho_{s_1}(\mathcal{L})}{\rho_{s_2}(\mathcal{L})} \in [1-2\varepsilon, 1+2\varepsilon] \cdot \left(\frac{s_1}{s_2}\right)^n$$

Proof. We have

$$\rho_s(\mathcal{L}) = \det(\mathcal{L}^*) \cdot s^n \rho_{1/s}(\mathcal{L}^*) \in [1-\varepsilon, 1+\varepsilon] \cdot s^n \cdot \det(\mathcal{L}^*)$$

where the first equality uses Poisson summation and the fact that $\widehat{\rho_s} = s^n \rho_{1/s}$, and the second the definition of the smoothing parameter and the fact that $s \geq \eta_\varepsilon(\mathcal{L})$. Thus,

$$\frac{\rho_{s_1}(\mathcal{L})}{\rho_{s_2}(\mathcal{L})} \in [1-2\varepsilon, 1+2\varepsilon] \cdot \left(\frac{s_1}{s_2}\right)^n$$

□

Poor Person's Discrete Gaussian Sampling

For the first step of our reduction in the next section, we need an algorithm to sample from the discrete Gaussian distribution $D_{\mathcal{L},s}$ given s and some basis $\mathbf{B}$ of $\mathcal{L}$. Clearly, this is hard to do if $s < 1/\sqrt{n} \cdot \max_i ||\mathbf{b}_i||$ as it will then give us a way to make the vectors of $\mathbf{B}$ shorter, a computationally hard problem. However, one can hope that for significantly larger s, this is possible. Indeed, Gentry, Peikert and Vaikuntanathan [GPV08], following an algorithm of Klein [Kle00], show such a (polynomial-time) algorithm with $s \geq \omega(\sqrt{\log n}) \cdot \max_i ||\mathbf{b}_i||$ (in fact, something slightly stronger but it will not matter to us). Their algorithm samples from a distribution that is negligibly close (in statistical distance) to the discrete Gaussian.

Here, we will make do with something significantly weaker.

We will show a *very* simple algorithm $\mathsf{SimpleDGS}$ that samples from the discrete Gaussian $D_{\mathcal{L},s}$ where $s \geq 2^n \cdot \max_i ||\mathbf{b}_i||$. The algorithm simply samples a vector $\mathbf{v} \leftarrow N_s$ from the continuous Gaussian distribution with parameter s and "rounds" it modulo the parallelepiped $\mathcal{P}(\mathbf{B})$. That is, output

$$\mathbf{v}' = \mathbf{B}\lfloor \mathbf{v} \rceil \in \mathcal{L}(\mathbf{B})$$

To show that this is statistically close to $D_{\mathcal{L},s}$, we calculate the two probabilities:

- $\Pr[\mathbf{w} \sim D_{\mathcal{L},s}] = c \cdot \rho_s(\mathbf{w})$ for some constant normalization factor c.
- $\Pr[\mathbf{w} \sim \mathsf{SimpleDGS}] = c' \cdot \int_{\mathbf{x} \in \mathcal{P}(\mathbf{B})} \rho_s(\mathbf{w}+\mathbf{x}) d\mathbf{x}$.

The intuition is that $\rho_s(\mathbf{w}+\mathbf{x})$ is very close to $\rho_s(\mathbf{w})$ for all the *typical* vectors, that is, vectors of length at most $s\sqrt{n}$. Indeed,

$$\rho_s(\mathbf{w}+\mathbf{x}) = \rho_s(\mathbf{w}) \cdot e^{-\pi(2\langle \mathbf{w},\mathbf{x}\rangle + ||\mathbf{x}||^2)/s^2}$$

It suffices to show that $|2\langle \mathbf{w},\mathbf{x}\rangle + ||\mathbf{x}||^2)/s^2|$ is very small. Note that this quantity is at most $(2||\mathbf{w}||||\mathbf{x}|| + ||\mathbf{x}||^2)/s^2$ by Cauchy-Schwartz. Since $||\mathbf{w}|| \approx s\sqrt{n}$ is the length of the typical vectors (Exercise: Check this!) and $s \gg 2^n \max_i ||\mathbf{b}_i|| \geq 2^n ||\mathbf{x}||$, we are done.

A remark to a reader who might be wondering if this algorithm in fact performs better, i.e., with a smaller s, and if the large s is merely an artifact of our analysis. To show that it is not, the reader is recommended to let $\mathcal{L} = \mathbb{Z}$ and show that for small s, the rounded continuous Gaussian (our distribution) and the discrete Gaussian over $\mathbb{Z}$ are in fact statistically far.

Regev's BDD to LWE Reduction

Input: Lattice basis $\mathbf{B} \in \mathbb{Z}^{n \times n}$, $\mathbf{t} = \mathbf{Bs} + \mathbf{e} \in \mathbb{Z}^n$.
(*For simplicity, we will assume that* $||\mathbf{e}||$ *is known.*)
Output: LWE instance $\mathbf{A} \in \mathbb{Z}_q^{n \times m}$, $\mathbf{y} \in \mathbb{Z}_q^m$.

Repeat m times:

- ▶ Let $q \geq 2^{2n}$, where $s \geq q\sqrt{2} \cdot \eta_\varepsilon(\mathcal{L}^*)$ and $r \geq \sqrt{2} \cdot ||\mathbf{x}|| \cdot \eta_\varepsilon(\mathcal{L}^*)$.
- ▶ Sample a vector $\mathbf{v}_i \leftarrow D_{\mathcal{L}^*,s}$.
- ▶ Compute

$$\mathbf{a}_i := (\mathbf{B}^*)^{-1}\mathbf{v}_i = \mathbf{B}^T\mathbf{v}_i \pmod{q} \text{ and } b_i := \mathbf{t}^T\mathbf{v}_i + e_i' \pmod{q}$$

where $e_i' \leftarrow N_r$.

Run the LWE algorithm on input $(\mathbf{A}, \mathbf{b})$ where the columns of $\mathbf{A}$ are the $\mathbf{a}_i$, and output what it outputs.

4.4 From (Worst-case) BDD to (Average-case) LWE

We show the reduction from the worst-case bounded distance decoding problem, which we saw was morally the same as the LWE problem, to the average-case LWE problem.

We will produce LWE samples where the LWE noise are drawn from a continuous Gaussian. It is easy to discretize it and make the noise comes from the rounded continuous Gaussian distribution.

Claim 19. *The vectors $\mathbf{a}_i$ are statistically close to uniformly random in $\mathbb{Z}_q^n$ and independent.*

Proof. By inspection, we see that the probability of getting $\mathbf{a}_i$ is the probability that the discrete Gaussian $D_{\mathcal{L}^*,s}$ lands up in the set $q\mathcal{L}^* + \mathbf{B}^*\mathbf{a}_i$. This is precisely

$$\frac{\rho_s(q\mathcal{L}^* + \mathbf{c})}{\sum_{\mathbf{c}} \rho_s(q\mathcal{L}^* + \mathbf{c})} \tag{4.1}$$

Since $s \geq q\eta_\varepsilon(\mathcal{L}^*) = \eta_\varepsilon(q\mathcal{L}^*)$, we know by Lemma 16 that

$$\sum_{\mathbf{c}} \rho_s(q\mathcal{L}^* + \mathbf{c}) \in [1 - 2\varepsilon, 1 + 2\varepsilon] \cdot \rho_s(q\mathcal{L}^*) \cdot q^n$$

and

$$\rho_s(q\mathcal{L}^* + \mathbf{c}) \in [1 - 2\varepsilon, 1 + 2\varepsilon] \cdot \rho_s(q\mathcal{L}^*)$$

therefore, the ratio in equation 8.1 is in the range $\frac{1}{q^n} \cdot [1 - 4\varepsilon, 1 + 4\varepsilon]$. Consequently, the statistical distance is at most 4ε. □

Claim 20. *$b_i = \mathbf{s}^T\mathbf{a}_i + e_i$ and e_i is statistically close to a (1-dimensional) continuous Gaussian N_t where $t = ||\mathbf{x}|| \cdot \sqrt{2}\eta_\varepsilon(\mathcal{L}^*)$.*

Proof. For the reduction and the proof, we will assume that $||\mathbf{e}||$ is known. This assumption can be removed with more care; we refer to [Reg09] for more details.

Start by noting that

$$\begin{aligned} b_i &= \mathbf{t}^T\mathbf{v}_i + e'_i \pmod q \\ &= (\mathbf{s}^T\mathbf{B}^T + \mathbf{e}^T)\mathbf{B}^*\mathbf{a}_i + e'_i \pmod q \\ &= \mathbf{s}^T\mathbf{B}^T\mathbf{B}^{-T}\mathbf{a}_i + \mathbf{e}^T\mathbf{v}_i + e'_i \pmod q \\ &= \mathbf{s}^T\mathbf{a}_i + e_i \pmod q \end{aligned}$$

where the second equality follows from the definition of $\mathbf{t} := \mathbf{B}\mathbf{s}+\mathbf{e}$ and that of $\mathbf{a}_i$, and $e_i := \mathbf{e}^T\mathbf{v}_i + e'_i$. It remains to analyze the distribution of e_i.

First, e'_i is distributed like $\mathbf{e}^T\mathbf{w}_i$ where $\mathbf{w}_i$ is a continuous Gaussian with parameter $\sqrt{2}\eta_\varepsilon(\mathcal{L}^*)$. Thus,

$$e'_i = \mathbf{e}^T(\mathbf{v} + \mathbf{w}) = \mathbf{e}^T\mathbf{w}'$$

where $\mathbf{w}'$ is distributed like $N_{s'}$ by Lemma 17 with

$$s' \approx q \cdot ||\mathbf{x}|| \cdot \eta_\varepsilon(\mathcal{L}^*) \leq cq\lambda_1(\mathcal{L})\eta_\varepsilon(\mathcal{L}^*) \in cq \cdot [1, \sqrt{n}]$$

by Banaszczyk's theorem. In the worst case, if $c \ll 1/\sqrt{n}$, this gives us an LWE distribution with meaningfully bounded error. □

In summary, the reduction solves $1/\sqrt{n}$-BDD assuming an LWE solver with a constant factor noise-to-modulus ratio.

4.5 From (Worst-case) SIVP to (Worst-case) BDD

A Classical Reduction

We now present a classical reduction from gapSVP to BDD due to Peikert [Pei09]. We contrast this with Regev's quantum reduction from SIVP to BDD [Reg09].

The advantage of Peikert's reduction, of course, is that it is classical. However, it is a reduction from a decision problem (gapSVP) to a search problem (BDD), as opposed to Regev's quantum reduction that reduces from search SIVP. For classes of lattices such as ideal lattices, the gapSVP problem for small factors turns out to be easy making the (analogous) reduction vacuous, so it is important to find a reduction starting from a *search* problem. Thus, the following question is wide open.

Open Problem 4.1. Show a (worst-case) reduction from SIVP (or SVP or CVP) to BDD.

We sketch the idea behind Peikert's reduction which in turn draws inspiration from a beautiful *coAM* protocol for gapSVP due to Goldreich and Goldwasser. Let $\mathcal{L}$ be the input lattice with the promise that $\lambda_1(\mathcal{L}) \leq 1$ or $\lambda_1(\mathcal{L}) > \gamma$. Assume that we have access to a c-BDD solver, namely an algorithm that returns the closest lattice vector given the promise that the target point is within distance $c \cdot \lambda_1(\mathcal{L})$ from the lattice. The reduction works as follows.

- Pick a random lattice point $\mathbf{z} \in \mathcal{L}$ and add a random point $\mathbf{e}$ from a ball of radius $c \cdot \gamma$.
- Run the BDD solver with input $\mathbf{t} := \mathbf{z} + \mathbf{e}$.
- If the BDD solver produces a vector $\mathbf{z}' = \mathbf{z}$, output NO ("large λ_1") else output NO ("small λ_1").

On the one hand, if $\lambda_1(\mathcal{L}) > \gamma$, then the distance of $\mathbf{t}$ from the lattice is at most $c \cdot \lambda_1(\mathcal{L})$ and thus it satisfies the BDD promise. Consequently, the BDD solver will return $\mathbf{z}$. On the other hand, if $\lambda_1(\mathcal{L}) \leq 1$, the (uniform distribution on the) balls centered at $\mathbf{z}$ and $\mathbf{z} + \mathbf{u}$ where $||\mathbf{u}|| = \lambda_1(\mathcal{L})$ are statistically close, *if* $c\gamma \geq \sqrt{n}$. Therefore, a c-BDD algorithm helps us solve $\sqrt{n}/c$-gapSVP.

Putting this together with the worst-case to average-case reduction, we get a $O(n)$-gapSVP algorithm given an LWE solver with constant noise-to-modulus ratio.

CHAPTER 5

Pseudorandom Functions from Lattices

Pseudorandom functions (PRF) can in principle be constructed from LWE (and even SIS) completely generically following the Goldreich-Goldwasser-Micali paradigm that constructs PRFs from pseudorandom generators and even one-way functions. However, direct constructions often come equipped with other nice properties such as parallelism, key homomorphism, constrained evaluation, and more.

5.1 Pseudorandom Generator from LWE

The LWE function

$$G_{\mathbf{A}}(\mathbf{s}, \mathbf{e}) = \mathbf{s}^T \mathbf{A} + \mathbf{e}^T$$

is a pseudorandom generator with two caveats:

- It is a family of PRGs indexed by $\mathbf{A}$. A random function chosen from the family is then a PRG. This is not a big issue usually (except when considering questions related to who picks the $\mathbf{A}$).
- As-is, the domain seems to be $\mathbb{Z}_q^n \times \mathbb{Z}_q^m$ and the range is $\mathbb{Z}_q^m$ so the function does not even seem to expand! However, in reality, the function takes as input a smaller number of random bits used to sample $\mathbf{e}$, roughly $m \log(\alpha q)$ to sample from a Gaussian of standard deviation αq. When this is done, for sufficiently large m, the function does expand, and is pseudorandom.

5.2 GGM Construction

Goldreich, Goldwasser and Micali show how to construct a pseudorandom function family starting from any pseudorandom generator. This can well be applied to the LWE PRG described above, however it results in a rather unwieldy construction. We show below constructions that are much prettier, and as a side-effect, give us several advantages such as key homomorphism and parallel evaluation (as we will see today) and constrained evaluation (as we will see in later lectures).

5.3 BLMR13 Construction

The Gadget Matrix

We need the *gadget matrix* which will make its appearance several times in the next few lectures.

In a nutshell, our gadget matrix $\mathbf{G}$ is an $n \times m$ matrix (where $m \geq n \log q$) with the property that $\mathbf{G} \cdot \{0,1\}^m \supseteq \mathbb{Z}_q^n$. That is, for every vector $\mathbf{v} \in \mathbb{Z}_q^n$, there is a 0-1 vector $\mathbf{w}$ such that $\mathbf{Gw} = \mathbf{v}$ (mod q). For example, the matrix $\mathbf{G} \in \mathbb{Z}_7^{2\times 6}$ is the following matrix:

$$\mathbf{G} = \begin{bmatrix} 1 & 2 & 4 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 2 & 4 \end{bmatrix}$$

Indeed for every vector $v = \begin{bmatrix} v_1 \\ v_2 \end{bmatrix}$, let $v_1 = v_{12}v_{11}v_{10}$ denote its bit representation (and similarly for v_2). Then,

$$\mathbf{G} \begin{bmatrix} v_{10} \\ v_{11} \\ v_{12} \\ v_{20} \\ v_{21} \\ v_{22} \end{bmatrix} = \begin{bmatrix} v_1 \\ v_2 \end{bmatrix}$$

More generally, let $\mathbf{g}$ denote the gadget vector $[1\ 2\ 4\ \ldots 2^{\lceil \log_2 q \rceil - 1}] \in \mathbb{Z}_q^{1\times\lceil \log_2 q \rceil}$. Then, $\mathbf{G} = \mathbf{g} \otimes \mathbf{I}_n$ is the tensor product of $\mathbf{g}$ with the $n \times n$ identity matrix $\mathbf{I}_n$. (If $m > n\lceil \log q \rceil$, pad this with a block of the zero matrix.)

We will denote the inverse mapping by $\mathbf{G}^-$. That is, $\mathbf{G}^-(\mathbf{v}) = \mathbf{w}$ if (a) $\mathbf{w}$ has 0 or 1 entries; and (b) $\mathbf{Gw} = \mathbf{v}$ (mod q). Note that there could be many such $\mathbf{w}$ that satify these properties, so $\mathbf{G}^{-1}$ is best thought of as a multi-valued function.

Flipped LWE: Small A, Random s

We start with the proof that LWE with roles reversed, namely where the entries of $\mathbf{A}$ are random small, and $\mathbf{s}$ is random, is as secure as LWE. Note that (a) we showed that "Normal Form LWE" where $\mathbf{A}$ is random and $\mathbf{s}$ is random small, is as secure as LWE (in Lecture 1 and lecture 4) and (b) if both $\mathbf{A}$ and $\mathbf{s}$ have small entries, the problem is easy, as it is essentially just linear regression, a convex optimization problem.

Assume that $\mathbf{A} \leftarrow \{0,1\}^{N\times m}$ and $\mathbf{s} \leftarrow \mathbb{Z}_q^N$ are uniformly random. The Flipped LWE problem asks to distinguish between $(\mathbf{A}, \mathbf{s}^T\mathbf{A} + \mathbf{e}^T)$ from a truly random pair from the same domains. Note that $\mathbf{s}$ is likely not uniquely determined, rather only determined up to small additive error, so if one wanted to define the search version, it should be done with some care.

Lemma 21. *Flipped LWE*$(N = n\log q, m, q, \chi)$ *is as hard as LWE*(n, m, q, χ).

Proof. We show a reduction from (decisional) LWE to flipped-LWE. Given an LWE sample $(\mathbf{A}, b = \mathbf{s}^T\mathbf{A} + e)$, we rewrite it as

$$(\mathbf{A}, b = (\mathbf{G}^T\mathbf{s})^T\mathbf{G}^-(\mathbf{A}) + e)$$

pick a random $\mathbf{s}' \in \mathbb{Z}_q^N$ and compute $b' = b + \mathbf{s}'^T\mathbf{G}^-(\mathbf{A})$. Pass $(\mathbf{G}^-(\mathbf{A}), b')$ to the flipped LWE adversary.

First, notice that $\mathbf{G}^{-}(\mathbf{A})$ is a uniformly random 0-1 matrix – this is true either when q is close to a power of two, or by extending the definition of the $\mathbf{G}$ matrix by adding more powers of two.

Secondly, notice that

$$b' = b + \mathbf{s}'^T \mathbf{G}^{-}(\mathbf{A}) = (\mathbf{G}^T \mathbf{s} + \mathbf{s}')^T \mathbf{G}^{-}(\mathbf{A}) + e$$

which is exactly a flipped LWE sample when b is an LWE sample and uniformly random otherwise.

This transforms a flipped-LWE distinguisher into an LWE distinguisher. □

Construction

Both constructions we show will follow the following general template. The PRF family will be indexed by a secret seed $\mathbf{s} \in \mathbb{Z}_q^n$, and a sequence of public matrices $\mathbf{A} = (\mathbf{A}_0, \mathbf{A}_1, \ldots)$. On input $\mathbf{x} \in \{0,1\}^\ell$, the function will be defined as

$$\mathsf{PRF}_{\mathbf{s},\mathbf{A}}(x) = \mathbf{s}^T \mathbf{A}_x + \mathbf{e}_x^T \pmod q$$

where $\mathbf{A}_x$ is defined as some function (depending on the construction) of $\mathbf{A}$ and $\mathbf{x} \in \{0,1\}^\ell$.

The first problem that one encounters with this framework is where does the error $\mathbf{e}_x^T$, which is supposed to be different and "pseudo-fresh" for every x, come from? The first trick we will play is to sidestep this question entirely, and go via the learning with rounding paradigm of Banerjee, Peikert and Rosen [BPR12]. That is, we will define

$$\mathsf{PRF}_{\mathbf{s},\mathbf{A}}(x) = \lfloor \mathbf{s}^T \mathbf{A}_x + \mathbf{e}_x^T \rceil_p \pmod p$$

where $\lfloor \cdot \rceil_p : \mathbb{Z}_q \to \mathbb{Z}_p$ refers to a function that, on input $x \in \mathbb{Z}_q$ outputs the multiple of p that is closest to it. That is,

$$\lfloor x \rceil_p = \left\lfloor \frac{p}{q} x \right\rceil$$

where $\lfloor \cdot \rceil$ refers to the function that rounds to the nearest integer.

In the BLMR construction, the public parameters are $\mathbf{A} := (\mathbf{A}_0, \mathbf{A}_1)$ where both matrices are drawn at random from $\mathbb{Z}_q^{n \times n}$ and $\mathbf{A}_x$ is defined as a subset product. We are now ready to define the BLMR construction. The construction sets

$$\mathbf{A}_x = \mathbf{G}^{-}(\mathbf{A}_{x_1}) \cdot \mathbf{G}^{-}(\mathbf{A}_{x_2}) \ldots \mathbf{G}^{-}(\mathbf{A}_{x_\ell}) = \prod_{i=1}^{\ell} \mathbf{G}^{-}(\mathbf{A}_i)$$

and therefore,

$$\mathsf{PRF}_{\mathbf{s},\mathbf{A}_0,\mathbf{A}_1}(x) = \lfloor \mathbf{s}^T \mathbf{A}_x \rceil_p \pmod p$$

The only remaining loose end is how to choose p. Intuitively, the larger the p, the less secure the construction is. Indeed, if $p = q$, there is no rounding and the PRF is a linear function! The smaller the p, the less efficient the construction is, in terms of how many pseudorandom bits it produces per invocation.

Parallelism. The pseudorandom function can be computed in $\log \ell$ levels of matrix multiplication, or in the complexity class NC^2.

(Approximate) Key Homomorphism. The PRF has the attractive feature that $\mathsf{PRF}_{\mathbf{s}}(x) + \mathsf{PRF}_{\mathbf{s}'}(x)$ (where both PRFs use the same two public matrices $\mathbf{A}_0$ and $\mathbf{A}_1$) is approximately equal to $\mathsf{PRF}_{\mathbf{s}+\mathbf{s}'}(x)$. This feature has a number of applications such as constructing a distributed PRF and a (additively) related-key secure PRF.

Proof of Security

We will, for simplicity, prove that the truth table of the PRF is indistinguishable from i.i.d. random strings using a reduction that runs in time exponential in the input length, namely ℓ. More refined approaches, following the GGM proof, are possible, but omitted from our exposition.

The proof proceeds in a number of hybrids. Define "intermediate" pseudorandom functions $\mathsf{PRF}^{(i)}$ for $i = 0, \ldots, \ell$ as follows.

$$\mathsf{PRF}^{(i)}_{\mathbf{s}_0,\ldots,\mathbf{s}_{2^i-1}}(x'||x'') = \lfloor \mathbf{s}_{x'}^T \mathbf{A}_{x''} \rceil_p$$

where x' is the i-bit prefix of $x = x'||x''$.

Note that $\mathsf{PRF}^{(0)}$ is exactly the PRF we defined with $\mathbf{s}_0 = \mathbf{s}$. On the other hand, $\mathsf{PRF}^{(\ell)}$ is a random function. The proof goes via a hybrid argument that switches from $\mathsf{PRF}^{(0)}$ to $\mathsf{PRF}^{(\ell)}$ in ℓ steps. We will now show that each such switch is computationally indistinguishable to the adversary. For simplicity, we show this for $\mathsf{PRF}^{(0)}$ versus $\mathsf{PRF}^{(1)}$.

- First, consider

$$\mathsf{PRF}^{(0)}_{\mathbf{s}}(x_1 \ldots x_\ell) = \lfloor \mathbf{s}^T \mathbf{A}_x \rceil_p = \lfloor \mathbf{s}^T \mathbf{G}^-(\mathbf{A}_{x_1}) \cdot \prod_{i=2}^{\ell} \mathbf{G}^-(\mathbf{A}_{x_i}) \rceil_p$$

- We first show that this distribution is statistically close to

$$\lfloor (\mathbf{s}^T \mathbf{G}^-(\mathbf{A}_{x_1}) + \mathbf{e}_{x_1}) \cdot \prod_{i=2}^{\ell} \mathbf{G}^-(\mathbf{A}_{x_i}) \rceil_p$$

  Indeed, the intuition is that the difference between the distributions is only noticeable when the addition of $\mathbf{e}_{x_1} \cdot \prod_{i=2}^{\ell} \mathbf{G}^-(\mathbf{A}_{x_i})$ flips over one of the coordinates of the vector $\mathbf{s}^T \prod_{i=1}^{\ell} \mathbf{G}^-(\mathbf{A}_{x_i})$ over a multiple of p. First, notice that since $\prod_{i=1}^{\ell} \mathbf{G}^-(\mathbf{A}_{x_i})$ is full-rank w.h.p. and $\mathbf{s}$ is uniformly random, so is $\mathbf{s}^T \prod_{i=1}^{\ell} \mathbf{G}^-(\mathbf{A}_{x_i})$. The probability of flipping over is at most $N \cdot ||\mathbf{e}_{x_1} \cdot \prod_{i=2}^{\ell} \mathbf{G}^-(\mathbf{A}_{x_i})||_\infty/(q/p)$ which is negligible if $||\mathbf{e}_{x_1}||_\infty \ll q/p \cdot 1/N^{\ell+1} \cdot 2^{-\omega(\log \lambda)}$. Assume that $p = \Omega(q)$, this is like assuming LWE with noise-to-modulus ratio that is roughly N^ℓ. In turn, this translates to assuming that gapSVP is hard to approximate to within N^ℓ, a factor exponential in the input length of the PRF.

- Next, observe that this is computationally indistinguishable from

$$\mathbf{s}_{x_1}^T \prod_{i=2}^{\ell} \mathbf{G}^-(\mathbf{A}_{x_i})$$

  by LWE. Finally, this distribution is precisely $\mathsf{PRF}^{(1)}$.

5.4 BP14 Construction

The only difference between the BLMR13 and BP14 constructions is in the definition of $\mathbf{A}_x$. Let $x = x_1 x_2 \dots x_\ell$. BP14 defines $\mathbf{A}_x$ recursively as follows. $\mathbf{A}_\varepsilon = \mathbf{I}_{m\times m}$ (where ε is the empty string) and

$$\mathbf{A}_{bx} = \mathbf{G}^{-}(\mathbf{A}_b \cdot \mathbf{A}_x)$$

Thus,

$$\mathbf{A}_x = \mathbf{G}^{-}(\mathbf{A}_{x_1} \cdot \mathbf{G}^{-}(\mathbf{A}_{x_2} \dots \mathbf{G}^{-}(\mathbf{A}_{x_\ell})))$$

This allows us to base security on LWE with slightly superpolynomial noise-to-modulus ratio. Roughly speaking, we will switch from

$$\lfloor \mathbf{s}^T \mathbf{G} \mathbf{A}_x \rceil_p = \lfloor \mathbf{s}^T \mathbf{A}_{x_1} \cdot \mathbf{A}_{x_{2\dots\ell}} \rceil_p$$

to

$$\lfloor (\mathbf{s}^T \mathbf{A}_{x_1} + \mathbf{e}_{x_1}) \cdot \mathbf{A}_{x_{2\dots\ell}} \rceil_p$$

by a statistical argument similar to the above. However, now, the norm of $\mathbf{A}_{x_{2\dots\ell}}$ is polynomial in N, independent of ℓ which makes the argument considerably more efficient. We still will need the $2^{-\omega(\log\lambda)}$ term for the statistical argument.

Note that this construction loses parallelism.

Open Problem 5.1**.** Construct an LWE-based pseudorandom function that can be computed in NC1 and is based on LWE with polynomial modulus.

The computation in NC1 is satisfied by the BLMR construction (and by a construction of [BPR12] using "synthesizers"), and the polynomial modulus is satisfied by the a direct construction based on GGM (also in [BPR12]). We refer to [Kim20] for a detailed taxonomy of the existing PRF constructions as of Feb 2020.

Open Problem 5.2**.** Come up with a "direct" construction of a SIS-based PRG and PRF.

Of course, SIS gives us a one-way function (as described below) and can be used to construct a PRG by the result of Hastad-Impagliazzo-Levin-Luby and then a PRF by Goldreich-Goldwasser-Micali. But the resulting construction is very complex, and in particular, does not have the parallel evaluation property. A concrete question is to construct a PRF from SIS with parallel evaluation.

Collision-Resistant Hashing

We finish by describing a simple collision-resistant hash function based on SIS.

A collision resistant hashing scheme $\mathcal{H}$ consists of an ensemble of hash functions $\{\mathcal{H}_n\}_{n\in\mathbb{N}}$ where each $\mathcal{H}_n$ consists of a collection of functions that map n bits to $m < n$ bits. So, each hash function compresses its input, and by pigeonhole principle, it has collisions. That is, inputs $x \neq y$ such that $h(x) = h(y)$. Collision-resistance requires that every p.p.t. adversary who gets a hash function $h \leftarrow \mathcal{H}_n$ chosen at random fails to find a collision except with negligible probability.

Collision-Resistant Hashing from SIS. Here is a hash family $\mathcal{H}_n$ that is secure under $\mathsf{SIS}(n, m, q, B)$ where $n \log q > m \log(B+1)$. Each hash function $h_{\mathbf{A}}$ is parameterized by a matrix $\mathbf{A} \in \mathbb{Z}_q^{n \times m}$, takes as input $\mathbf{e} \in [0, \ldots, B]^m$ and outputs

$$h_{\mathbf{A}}(\mathbf{e}) = \mathbf{A}\mathbf{e} \bmod q$$

A collision gives us $\mathbf{e}, \mathbf{e}' \in [0, \ldots, B]^m$ where $\mathbf{A}\mathbf{e} = \mathbf{A}\mathbf{e}' \bmod q$ which in turn says that $\mathbf{A}(\mathbf{e} - \mathbf{e}') = 0 \bmod q$. Since each entry of $\mathbf{e} - \mathbf{e}'$ is in $[-B, \ldots, B]$, this gives us a solution to $\mathsf{SIS}(n, m, q, B)$.

CHAPTER 6

Trapdoors, Gaussian Sampling and Digital Signatures

We will work with the ℓ_∞ norm throughout these lecture notes; tighter bounds are sometimes possible with the Euclidean norm but we would like to avoid the complication of computing the exact factors in favor of simplicity and conceptual clarity.

6.1 Lattice Trapdoors

Recall that

$$\Lambda^\perp(\mathbf{A}) = \{\mathbf{z} \in \mathbb{Z}^m : \mathbf{A}\mathbf{z} = \mathbf{0} \pmod{q}\}$$

is a rank-m lattice. A lattice trapdoor for a matrix $\mathbf{A} \in \mathbb{Z}_q^{n\times m}$ is a short basis for the lattice $\Lambda^\perp(\mathbf{A})$. More generally, a set of short linearly independent vectors in $\Lambda^\perp(\mathbf{A})$ suffices. More explicitly:

Definition 22. *A matrix $\mathbf{T} \in \mathbb{Z}^{m\times m}$ is a β-good lattice trapdoor for a matrix $\mathbf{A} \in \mathbb{Z}_q^{n\times m}$ if*

1. *Each column vector of $\mathbf{T}$ is in the (right) mod-q kernel of $\mathbf{A}$, namely, $\mathbf{AT} = \mathbf{0} \pmod{q}$;*
2. *Each column vector of $\mathbf{T}$ is short, namely for all $i \in [m]$, $||\mathbf{t}_i||_\infty \leq \beta$; and*
3. *$\mathbf{T}$ has rank m over $\mathbb{R}$.*

Note that the rank of $\mathbf{T}$ over $\mathbb{Z}_q$ can be no more than $m - n$; so, at first sight, the first and the third conditions may appear to be contradictory. However, the fact that we require *the real rank* over $\mathbf{T}$ to be large is the crucial thing here. This is related to why $\Lambda^\perp(\mathbf{A})$ as a lattice has rank m, even though as a linear subspace of $\mathbb{Z}_q^m$ has rank only $m - n$. Another way to look at $\mathbf{T}$ is that each of its columns is a homogenous SIS solution with respect to $\mathbf{A}$.

What good is such a trapdoor? We will demonstrate (in Section 6.3) its usefulness by showing that it can be used to solve both LWE and (inhomogenous) SIS with respect to $\mathbf{A}$.

6.2 Trapdoor Sampling

Leftover Hash Lemma

We will use the following form of the leftover hash lemma.

Lemma 23. *Let P be a probability distribution over $\mathbb{Z}^m$. The following two distributions have statistical distance at most ε as long as $H_\infty(P) \geq n \log q + 2\log(1/\varepsilon)$:*

$$(\mathbf{A}, \mathbf{A}\mathbf{e} \pmod q) \approx (\mathbf{A}, \mathbf{u})$$

where $\mathbf{A} \leftarrow \mathbb{Z}_q^{n\times m}$ is uniformly random, $\mathbf{e} \leftarrow X$ is drawn from the probability distribution P and $\mathbf{u} \leftarrow \mathbb{Z}_q^n$ is uniformly random. Here, $H_\infty(P)$ refers to the min-entropy of P.

For a proof, we refer the reader to these lecture notes.

Sampling a Random A with a Single Trapdoor Vector

Ajtai in 1996 gave us a procedure to sample a (statistically close to) uniformly random matrix $\mathbf{A} \in \mathbb{Z}_q^{n\times m}$ together with a single short vector $\mathbf{t} \in \mathbb{Z}^m$ such that $\mathbf{At} = \mathbf{0} \pmod q$. We begin our journey into trapdoors by describing this simple procedure.

1. Pick a uniformly random matrix $\mathbf{A}' \in \mathbb{Z}_q^{n\times(m-1)}$.
2. Pick a uniformly random vector $\mathbf{t} \in \{0,1\}^{m-1}$.
3. Define
$$\mathbf{A} = [\mathbf{A}' || -\mathbf{A}'\mathbf{t}] \quad \text{and} \quad \mathbf{t} = \begin{bmatrix} \mathbf{t} \\ 1 \end{bmatrix}$$
as the matrix and trapdoor vector, respectively.

It is clear that $\mathbf{t}$ is a short vector in the right-mod-q kernel of $\mathbf{A}$. It remains to show that $\mathbf{A}$ is close to uniformly random, which reduces to showing that $\mathbf{A}'\mathbf{t}$ is close to uniform given $\mathbf{A}'$. This follows directly from the leftover hash lemma assuming that $m \geq n \log q + \lambda$.

More generally, if we let $||\mathbf{t}||_\infty \leq B$, then we need $m \geq n \log q / \log B + \lambda$.

Ajtai-MP Trapdoor Sampling

Now, one can try to extend the above procedure to sample $\mathbf{A}$ together with more and more short vectors until you reach m (hopefully) linearly independent vectors and then we have a trapdoor! However, this naïve idea fails to work. Indeed, letting $m^* := n \log q + \lambda$, we can generate a close to uniform matrix $\mathbf{A} \in \mathbb{Z}_q^{n\times(m^*+\ell)}$ together with ℓ trapdoor vectors (We leave it as an exercise to the reader to figure out how.) However, this will never "catch up" as the number of trapdoor vectors (ℓ) always remains short of the rank ($m^* + \ell$).

We start with the observation that an "inhomogenous trapdoor" (a notion that we will define in a minute) will let us achieve our goals of solving LWE and SIS just as well. An inhomogenous trapdoor $\mathbf{T} \in \mathbb{Z}^{m\times n \log q}$ is a matrix with short columns such that $\mathbf{AT} = \mathbf{G} \pmod q$ where $\mathbf{G}$ is the *gadget matrix* that we constructed and used in the last lecture.

To jog our memory, we defined

$$\mathbf{g} := \begin{bmatrix} 1 & 2 & 4 & \ldots & 2^{\lceil \log q \rceil - 1} \end{bmatrix} \text{ and } \mathbf{G} := \mathbf{I} \otimes \mathbf{g}$$

where $\mathbf{I}$ is the $n \times n$ identity matrix. In other words, $\mathbf{G}$ is the block diagonal $n \times (n\lceil \log q \rceil)$ matrix with $\mathbf{g}$ in each of its diagonal blocks.

Why does this suffice to solve LWE and SIS? Let's just do LWE here and leave SIS as an exercise. Given $\mathbf{b}^T = \mathbf{s}^T\mathbf{A} + \mathbf{e}^T \pmod q$, we do

$$\mathbf{b}^T\mathbf{T} = (\mathbf{s}^T\mathbf{A} + \mathbf{e}^T)\mathbf{T} = \mathbf{s}^T\mathbf{G} + \mathbf{e}^T\mathbf{T} \pmod q$$

In other words, we just transformed an LWE sample relative to $\mathbf{A}$ into an LWE sample relative to $\mathbf{G}$, with a slight increase in error. Now, if we have a trapdoor (in the sense of Definition **??**) for $\mathbf{G}$ (and we will show in a few minutes that we do indeed have such a trapdoor), we can solve LWE!

Trapdoor for G: The case of $q = 2^k$. We invite the reader to think about this a bit before reading on. Let us first construct a trapdoor $\mathbf{T_g} \in \mathbb{Z}^{\lceil \log q \rceil \times \lceil \log q \rceil}$. We will then see that $\mathbf{T_G} = \mathbf{I} \otimes \mathbf{T_g}$. Indeed,

$$\mathbf{G} \cdot \mathbf{T_G} = (\mathbf{I} \otimes \mathbf{g}) \cdot (\mathbf{I} \otimes \mathbf{T_g}) = \mathbf{I} \otimes (\mathbf{g}\mathbf{T_g}) = \mathbf{0} \pmod q$$

Here is the trapdoor for $\mathbf{g}$:

$$\mathbf{T_g} = \begin{bmatrix} 2 & & & & \\ -1 & 2 & & & \\ & -1 & \ldots & & \\ & & \ddots & & \\ & & & 2 & \\ & & & -1 & 2 \end{bmatrix}$$

Let us check.

- $\mathbf{T_g}$ has short columns. Indeed $||\mathbf{T_g}||_\infty = 2$.
- $\mathbf{g}\mathbf{T_g} = 0 \pmod q$.
- The determinant of $\mathbf{T_g}$ is $q = 2^k$. Therefore, it has full rank over $\mathbb{R}$. It decidedly *does not* have full rank over $\mathbb{Z}_q$ since its determinant is 0 mod q. (And this had better be the case!)

Trapdoor for G: The general case. As before, let us construct a trapdoor $\mathbf{T_g} \in \mathbb{Z}^{\lceil \log q \rceil \times \lceil \log q \rceil}$. We will then see that $\mathbf{T_G} = \mathbf{I} \otimes \mathbf{T_g}$. Here is the trapdoor for $\mathbf{g}$:

$$\mathbf{T_g} = \left[\begin{array}{ccccc|c} 2 & & & & & \\ -1 & 2 & & & & \\ & -1 & \ldots & & & \\ & & \ddots & & & \mathsf{bits}(q) \\ & & & 2 & & \\ & & & -1 & & \end{array}\right]$$

The only difference is in the last column which is now the bit representation of the modulus q. Checking that this is indeed a trapdoor for $\mathbf{g}$ is left as an exercise. (Hint: for the full rank property, prove that the determinant of this matrix is q.)

Sampling A together with an Inhomogenous Trapdoor. Sample a uniformly random $\mathbf{B} \in \mathbb{Z}_q^{n\times m^*}$ where $m^* = n\log q + \lambda$ (as before). Set

$$\mathbf{A} = [\mathbf{B}||\mathbf{BR} + \mathbf{G}] \quad (\text{over } \mathbb{Z}_q)$$

where $\mathbf{R} \in \mathbb{Z}_q^{m^*\times m}$ is a uniformly random 0-1 matrix. Notice that

$$\mathbf{A} \cdot \begin{bmatrix} -\mathbf{R} \\ \mathbf{I} \end{bmatrix} = \mathbf{G} \pmod q$$

and since $||\mathbf{R}||_\infty \leq 1$, we have an inhomogenous trapdoor! Furthermore, $\mathbf{A}$ is close to random by leftover hash lemma (as before).

One could directly use the inhomogenous trapdoor to solve LWE and SIS but we will go one step further and show how to get a trapdoor for $\mathbf{A}$.

Sampling A with a Trapdoor, Finally. First of all, we have

$$[\mathbf{B}||\mathbf{BR} + \mathbf{G}] \cdot \begin{bmatrix} -\mathbf{R} \\ \mathbf{I} \end{bmatrix} = \mathbf{G}$$

Thus,

$$[\mathbf{B}||\mathbf{BR} + \mathbf{G}] \cdot \begin{bmatrix} \mathbf{I} & -\mathbf{R} \\ \mathbf{0} & \mathbf{I} \end{bmatrix} = [\mathbf{B}||\mathbf{G}]$$

Finally, multiplying this on the right by $\begin{bmatrix} \mathbf{I} & \mathbf{0} \\ -\mathbf{G}^-(\mathbf{B}) & \mathbf{T_G} \end{bmatrix}$, we get

$$[\mathbf{B}||\mathbf{BR} + \mathbf{G}] \cdot \underbrace{\begin{bmatrix} \mathbf{I} & -\mathbf{R} \\ \mathbf{0} & \mathbf{I} \end{bmatrix} \cdot \begin{bmatrix} \mathbf{I} & \mathbf{0} \\ -\mathbf{G}^-(\mathbf{B}) & \mathbf{T_G} \end{bmatrix}}_{=\mathbf{T_A}} = [\mathbf{B}||\mathbf{G}] \cdot \begin{bmatrix} \mathbf{I} & \mathbf{0} \\ -\mathbf{G}^-(\mathbf{B}) & \mathbf{T_G} \end{bmatrix} = \mathbf{0} \pmod q$$

Thus, the lattice trapdoor

$$\mathbf{T_A} = \begin{bmatrix} \mathbf{I} + \mathbf{RG}^-(\mathbf{B}) & -\mathbf{RT_G} \\ -\mathbf{G}^-(\mathbf{B}) & \mathbf{T_G} \end{bmatrix}$$

We already saw that $\mathbf{AT_A} = \mathbf{0} \pmod q$. The ℓ_∞ norm of $\mathbf{T_A}$ is $O(m)$. Finally, since $\mathbf{T_A}$ is a product of two full-rank matrices, it is full-rank as well. (It has determinant q^n.)

6.3 Trapdoor Functions

Definition 24. *A family of functions*[1] $\mathcal{F}_n = \{f_i : \{0,1\}^n \to \{0,1\}^m\}$ *for some* $m = m(n)$ *is called a trapdoor function family if it comes with the following three associated polynomial-time algorithms.*

- *A probabilistic function generation algorithm that, on input* 1^n*, outputs an index* i *of a function* f_i *in the family as well as a trapdoor* t_i*.*

[1] To be precise, we should be talking about ensemble of such families one for every input length n. However, we will refrain from unnecessary notational gymnastics and will take that as understood.

- *A deterministic evaluation algorithm that, on input i and $x \in \{0,1\}^n$, outputs y. We need that $y = f_i(x)$.*
- *A deterministic inversion algorithm that, on input i, t_i and $y \in \{0,1\}^m$, outputs $x \in \{0,1\}^n$ or a special symbol $\perp$. We require that if $y \in \mathsf{Image}(f_i)$, then x is* an *inverse, namely $f_i(x) = y$.*

Injective Trapdoor Function

The function

$$f_{\mathbf{A}}(\mathbf{s}, \mathbf{e}) = \mathbf{s}^T \mathbf{A} + \mathbf{e}^T \pmod q$$

where $\mathbf{A} \in \mathbb{Z}_q^n$, $\mathbf{s} \in \mathbb{Z}_q^n$ and $\mathbf{e} \leftarrow \chi^m$ is a one-way family of functions, under LWE. Given the trapdoor $\mathbf{T}$, one inverts this as follows.

$$(\mathbf{s}^T \mathbf{A} + \mathbf{e}^T)\mathbf{T} = \mathbf{e}^T \mathbf{T} \pmod q$$

Now, since the latter quantity has absolute value at most $q/4$, it is $\mathbf{e}^T\mathbf{T}$ (over the integers). The mod-q has no effect, and this is the key observation. Now, multiplying the latter by $\mathbf{T}^{-1}$ (the inverse of $\mathbf{T}$ over the reals) recovers $\mathbf{e}$. Here, it is *very* important that $\mathbf{T}$ had full rank over the reals; otherwise, $\mathbf{T}^{-1}$ would not exist.

Surjective Trapdoor Function

The function

$$g_{\mathbf{A}}(\mathbf{e}) = \mathbf{T}\mathbf{e} \pmod q$$

where $\mathbf{A} \in \mathbb{Z}_q^{n \times m}$ where $m > n \log q$ and $\mathbf{e} \in [-\beta, \beta]^m$ is a one-way family of functions as well, under SIS, where $\beta = \mathsf{poly}(m)$.

One way to do this is the following. On input $\mathbf{v} \in \mathbb{Z}_q^n$, find some $\mathbf{w} \in \mathbb{Z}_q^m$ such that $\mathbf{A}\mathbf{w} = \mathbf{v} \pmod q$. Consider outputting

$$\mathbf{T} \cdot \{\mathbf{T}^{-1}\mathbf{w}\}$$

where $\{x\}$ denotes the fractional part of $x \in \mathbb{R}$. Why does this work?

- First of all,

$$\mathbf{A}\mathbf{T} \cdot \{\mathbf{T}^{-1}\mathbf{w}\} = \mathbf{A}\mathbf{T} \cdot (\mathbf{T}^{-1}\mathbf{w} - \lfloor \mathbf{T}^{-1}\mathbf{w} \rfloor) = \mathbf{v} - \mathbf{0} = \mathbf{v} \pmod q$$

  so we have an inverse.

- Secondly,

$$||\mathbf{T} \cdot \{\mathbf{T}^{-1}\mathbf{w}\}||_\infty \le m \cdot ||\mathbf{T}||_\infty \le m^2$$

This is an instance of Babai's "rounding algorithm" for the closest vector problem. In class, we saw yet another way to do this, which is Babai's nearest plane algorithm.

One could also use the inhomogenous trapdoor to accomplish this. For example, we saw that it is easy to compute a vector $\mathbf{e}' \in \{0,1\}^{m^*}$ such that $\mathbf{G}\mathbf{e}' = \mathbf{v} \pmod q$. Now, we claim that $\begin{bmatrix} \mathbf{R} \\ \mathbf{I} \end{bmatrix} \cdot \mathbf{e}'$ is a required inverse. Indeed,

$$\mathbf{A} \cdot \begin{bmatrix} \mathbf{R} \\ \mathbf{I} \end{bmatrix} \cdot \mathbf{e}' = \mathbf{G} \cdot \mathbf{e}' = \mathbf{v} \pmod q$$

6.4 Digital Signatures

Here is a simple digital signature scheme. (For a definition of digital signatures and what we mean by a secure digital signature, see Rafael Pass and abhi shelat's book.)

- The key generation algorithm samples a function together with a trapdoor. This would be $\mathbf{A}$ and $\mathbf{T}$. The public key is $\mathbf{A}$ and the secret key is $\mathbf{T}$.
- To sign a message m, first map it into the range of the function, e.g., by hashing it. That is, compute $\mathbf{v} = H(m)$. The signature is an inverse of $\mathbf{v}$ under the function $g_{\mathbf{A}}$. That is, a short vector $\mathbf{e}$ such that $\mathbf{A}\mathbf{e} = \mathbf{v} \pmod{q}$. This is guaranteed by the surjectivity of the function $g_{\mathbf{A}}$.
- Verification, given a message m, public key $\mathbf{A}$ and signature $\mathbf{e}$, consists of checking that $\mathbf{A}\mathbf{e} = H(m) \pmod{q}$ and that $||\mathbf{e}||_\infty \leq m^2$.

Unforgeability (given no signature queries) reduces to SIS in the random oracle model, i.e., assuming that H is a random oracle.

However, given signatures on adversarially chosen messages (in fact, even random messages), this scheme is broken. The key issue is that there are many inverses of $H(m)$, and the particular inverse computed using a trapdoor $\mathbf{T}$ leaks information about $\mathbf{T}$. Collecting this leakage over sufficiently many (polynomially many) signature queries enables an adversary to find $\mathbf{T}$, allowing her to forge signatures at will going forward.

This is most easily seen when the inversion procedure for $g_{\mathbf{A}}$ uses the inhomogenous trapdoor. Note that given $\mathbf{v}$, an adversary can compute $\mathbf{G}^{-}(\mathbf{v}) = \mathbf{e}'$ herself. She now gets a signature

$$\sigma = \begin{bmatrix} \mathbf{R} \\ \mathbf{I} \end{bmatrix} \cdot \mathbf{e}'$$

which gives her one equation on the secret $\mathbf{R}$. Given about m equations, she can solve linear equations and learn $\mathbf{R}$.

The situation remains essentially as dire even if you use the trapdoor (as opposed to the inhomogenous trapdoor). Using rounding vs the nearest plane algorithm does not help either; see the paper of Nguyen and Regev for robust attacks against this signature scheme. The fundamental difficulty seems to stem from the fact that the inversion procedure is deterministic!

To mitigate the difficulty, we need a special kind of inverter for $g_{\mathbf{A}}$. The inverter is a "pre-image sampler"; that is, it is given the trapdoor $\mathbf{T}$ and produces a "random" pre-image. More precisely, we need the following distributions to be statistically close (computational indistinguishabilty is fine, but we will achieve statistical closeness):

$$\left(\mathbf{A} \leftarrow \mathbb{Z}_q^{n\times m}, \mathbf{e} \leftarrow D_{\mathbb{Z}^m,s}, \mathbf{v} := \mathbf{A}\mathbf{e} \pmod{q}\right)$$
$$\approx_s \left(\mathbf{A} \leftarrow \mathbb{Z}_q^{n\times m}, \mathbf{e} \leftarrow \mathsf{PreSamp}(\mathbf{A}, \mathbf{T}, \mathbf{v}), \mathbf{v} \leftarrow \mathbb{Z}_q^n\right)$$

That is, the following processes produce statistically close outputs: (a) first sample $\mathbf{e}$ from a discrete Gaussian, and deterministically set $\mathbf{v}$ to be $\mathbf{A}\mathbf{e} \pmod{q}$; and (b) sample $\mathbf{v}$ uniformly and use the pre-image sampler to produce an inverse of $\mathbf{v}$ under $g_{\mathbf{A}}$ that is distributed according to the right conditional distribution. This distribution happens to be the discrete Gaussian over a coset of the lattice, that is,

$$\Lambda_{\mathbf{v}}^{\perp}(\mathbf{A}) := \{\mathbf{e} \in \mathbb{Z}^m : \mathbf{A}\mathbf{e} = \mathbf{v} \pmod q\}$$

In fact, this not quite enough; we need a multi-sample version of this. That is,

$$\left(\mathbf{A} \leftarrow \mathbb{Z}_q^{n\times m}, \{\mathbf{e}_i \leftarrow D_{\mathbb{Z}^m,s}, \mathbf{v} := \mathbf{A}\mathbf{e} \pmod q)\}_{i=1}^{\mathsf{poly}(\lambda)}\right)$$
$$\approx_s (\mathbf{A} \leftarrow \mathbb{Z}_q^{n\times m}, \{\mathbf{e} \leftarrow \mathsf{PreSamp}(\mathbf{A}, \mathbf{T}, \mathbf{v}), \mathbf{v} \leftarrow \mathbb{Z}_q^n\}_{i=1}^{\mathsf{poly}(\lambda)}\Big)$$

This is quite cumbersome to work with, so we propose an alternate stronger definition. That is, we require that for most $\mathbf{A} \leftarrow \mathbb{Z}_q^{n\times m}$ and any trapdoor $\mathbf{T}$ of length bounded by ℓ and $s \gg \ell$:

$$\left(\mathbf{e} \leftarrow D_{\mathbb{Z}^m,s}, \mathbf{v} := \mathbf{A}\mathbf{e} \pmod q\right)$$
$$\approx_s (\mathbf{e} \leftarrow \mathsf{PreSamp}(\mathbf{A}, \mathbf{T}, \mathbf{v}), \mathbf{v} \leftarrow \mathbb{Z}_q^n\Big)$$

Proof of Security. With the one change that the inverter is replaced by a pre-image sampler, our signature scheme becomes secure in the random oracle model. We showed the proof in the class.

6.5 Discrete Gaussian Sampling

Throughout, we will deal with sampling from a zero-centered discrete Gaussian.

Naïve Sampling

Let us first consider sampling from a discrete Gaussian over the simplest possible lattice, namely the one-dimensional lattice of integers $\mathbb{Z}$. The first idea to sample from the discrete Gaussian $D_{\mathbb{Z},s}$ is to sample from a continuous Gaussian N_s with parameter s and round to the nearest integer. Unfortunately, this is not a discrete Gaussian, not even statistically close to it. This is true even if s is much larger than the smoothing parameter.

Lemma 25. *The statistical distance between $D_{\mathbb{Z},s}$ and $\mathsf{Round}(N_s)$ is at least $1/s^3$.*

Proof. First, the probability assigned to zero by $\mathsf{Round}(N_s)$ is

$$\frac{2}{s} \cdot \int_0^{1/2} e^{-\pi x^2/s^2} dx = \frac{2}{\sqrt{\pi}} \cdot \int_0^{\sqrt{\pi}/2s} e^{-t^2} dt = \frac{2}{\sqrt{\pi}} \cdot \mathsf{erf}(\sqrt{\pi}/2s) \geq \frac{2}{\sqrt{\pi}} \cdot \left(\frac{\sqrt{\pi}}{2s} - \Omega\left(\frac{\sqrt{\pi}}{2s}\right)^3\right)$$

where the latter is due to a Taylor series approximation of the erf function and holds for a sufficiently large s. This quantity is at most

$$1/s - \Omega(1/s^3)$$

On the other hand, let's compute

$$\sum_{x\in\mathbb{Z}} \rho_s(x) = s \cdot \sum_{x\in\mathbb{Z}} \rho_{1/s}(x) = s \cdot (1 + \text{negl}(\lambda))$$

if s is above the negl(λ)-smoothing parameter of $\mathbb{Z}$ which is $\omega(\sqrt{\log \lambda})$.

Therefore, the probability assigned to zero by $D_{\mathbb{Z},s}$ is

$$\frac{1}{\sum_{x\in\mathbb{Z}} e^{-\pi x^2/s^2}} \approx 1/s$$

upto a negligible term.

Thus, the statistical distance between the two distributions in question is $\Omega(1/s^3)$ which is non-negligible unless s itself is super-polynomial. $\square$

For n-dimensional lattices, this statistical distance degrades with n as well making the situation much worse.

The reader may recall that the first step of Regev's worst-case to average-case reduction was sampling from a discrete Gaussian over a lattice for which Regev used the above procedure. However, he could afford to use an exponential s which makes the statistical distance small.

Sampling Discrete Gaussians over $\mathbb{Z}$

So, how do we sample from $D_{\mathbb{Z},s}$ for polynomial s? We will show that the general method of rejection sampling works. Let $Z = [-t\cdot s, t\cdot s]$ be a sufficiently large interval, where $t = \omega(\sqrt{\log \lambda})$. We do the following:

1. Sample a random integer $z \leftarrow Z$.
2. Output z with probability $\rho_s(z) := e^{-\pi z^2/s^2}$; else go to step 1 and repeat.

First of all, we will show that the probability that $D_{\mathbb{Z},s}$ assigns to numbers outside of the interval Z is negligible.

Lemma 26. *Let $s \geq \eta_\varepsilon(\mathbb{Z})$ for some $\varepsilon = \mathrm{negl}(\lambda)$, and $t > 0$. We have*

$$\Pr_{x\leftarrow D_{\mathbb{Z},s}}[|x| > t\cdot s] \leq c\cdot e^{-\pi t^2}$$

for some absolute constant $c > 0$.

Consider the probability distribution $D'_{\mathbb{Z},s}$ which assigns probability $\rho_s(x)$ for all $x \in Z \cap \mathbb{Z}$ and 0 otherwise. The lemma above shows that $D'_{\mathbb{Z},s}$ is close to $D_{\mathbb{Z},s}$ if s is larger than the negl(λ)-smoothing parameter of $\mathbb{Z}$, namely $\omega(\sqrt{\log n})$, and $t = \omega(\sqrt{\log n})$.

It is not hard to see that the procedure above samples from the distribution $D'_{\mathbb{Z},s}$ exactly. It remains to see that it terminates in polynomial time. We show two things which we leave as an exercise: (a) the probability that z sampled in step 1 lies in $[-s, s]$ is $\Omega(1/t)$ and (b) if such a z is sampled, it is output with probability $\Omega(1)$. Put together, the expected time for termination is $O(t) = \mathsf{poly}(\lambda)$.

Klein-GPV algorithm

We demonstrate the sampler in two dimensions. The generalization to n dimensions follows quite naturally.

CHAPTER 7

Identity-Based Encryption and Friends

7.1 Identity-based Encryption

Let us think first about deploying a public-key encryption scheme on a large scale. We need a mechanism to maintain a directory of (ID, PK) pairs where ID is the identifying information of a person, say Alice's e-mail address or phone number, that other people use to send her a message. Then, when you wish to send an email to Alice, you look up her public key in the directory and encrypt to the public key.

The directory, which forms part of a public-key infrastructure (PKI), has to be authenticated and trusted. For example, an adversary should not be able to insert an entry of the form (ID_A, PK'_A), where she presumably knows SK'_A, into the directory.

Identity-based encryption (IBE) solves the problem of having to maintain an authenticated PKI. In an IBE:

- there is a master authority who generates a master public key MPK together with a master secret key MSK, and publishes the MPK.
- To encrypt a message μ, one needs to know MPK and the identity ID (e.g., the e-mail address) of the recipient.
- Each user goes to the master authority and receives SK_{ID} after authenticating that they indeed are the owner of ID.
- Using SK_{ID}, the user can decrypt ciphertexts encrypted to the identity ID.

Let us now define the syntax of an IBE, formalizing the discussion above.

- $\mathsf{Setup}(1^\lambda)$: is a probabilistic algorithm that generates a master public key MPK and a master secret key MSK.

- $\mathsf{Enc}(MPK, ID, \mu)$: is a probabilistic algorithm that generates a ciphertext C of a message μ (for simplicity, we will encrypt bits but that is largely irrelevant) w.r.t. identity ID.

- $\mathsf{KeyGen}(MSK, ID \in \{0,1\}^*)$: is a probabilistic algorithm that generates a secret key SK_{ID}.

- $\mathsf{Dec}(SK_{ID}, C)$: is a deterministic decryption algorithm.

You may have noticed that the master authority can decrypt *all* the ciphertexts generated in this system and is therefore *very powerful*.

Application: Access Delegation across Space. I can act as the master authority and use an IBE to delegate decryption of certain subsets of messages to other people (e.g., my administative assistant). For example, all messages are tagged with a keyword $ID = \texttt{CS294}$, and I can issue the SK_{ID} to my assistant that lets him decrypt only those messages tagged with ID.

Application: Access Delegation across Time. Imagine that I go on (virtual) vacation to Cancun and want to take my laptop. However, I am worried that it will be stolen. So, I ask folks encrypting messages to me to use an IBE and tag the messages with an ID which is the current date. This allows me to generate a small set of secret keys, corresponding to the days that I am away, which allows me to decrypt only the corresponding small subset of messages. IBE lets me enjoy my vacation worry-free!

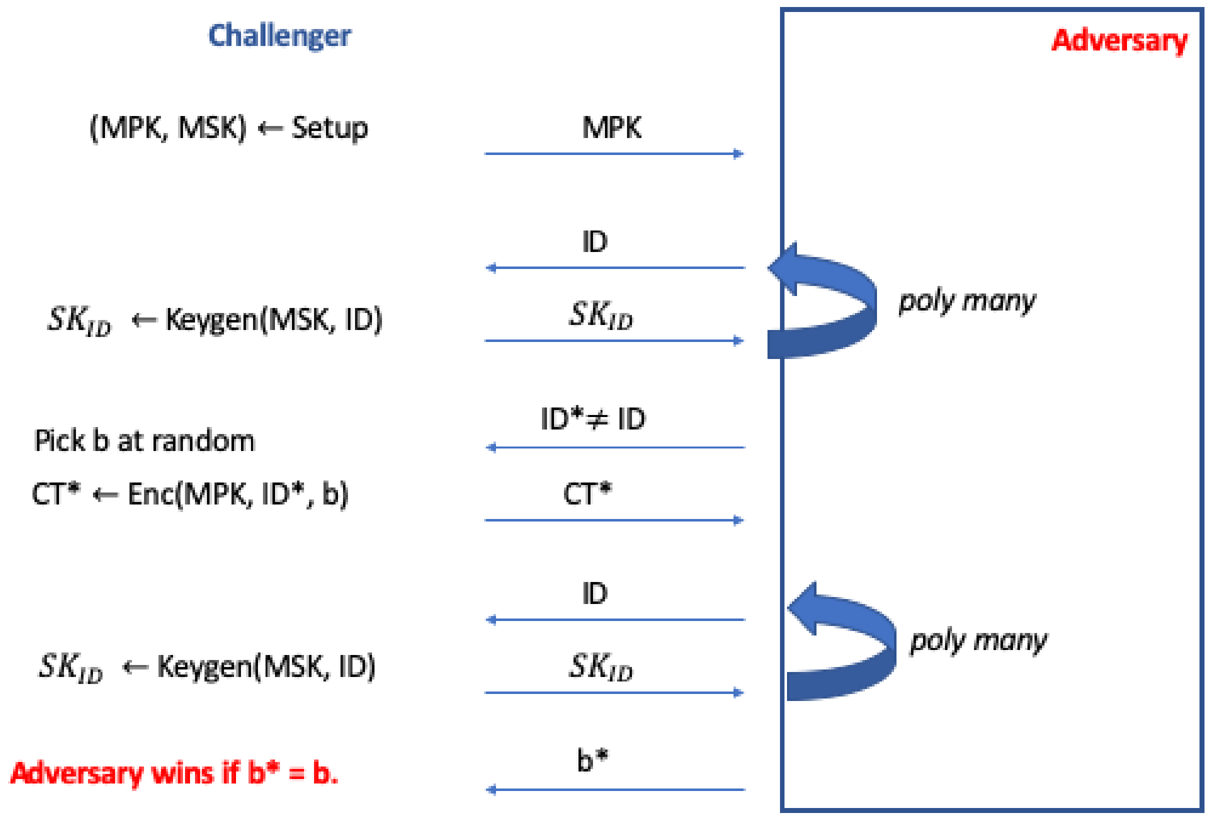
Application: Chosen-Ciphertext Security. IBE can be used in surprisingly non-trivial ways to construct other cryptographic systems, e.g., chosen ciphertext secure public-key encryption schemes and digital signature schemes (that we will describe later in this lecture).

Constructions. The first constructions used bilinear maps on elliptic curves (Boneh-Franklin'00) and quadratic residuosity (Cocks'00). We will present the third IBE scheme from LWE (Gentry-Peikert-Vaikuntanathan'08) and several variants today. Recently, Garg and Dottling have come up with a completely different scheme that relies on Diffie-Hellman groups (no need for bilinear maps!) Following up, Brakerski-Lombardi-Segev-Vaikuntanathan came up with a scheme based on learning parity with very low noise.

Definitions of Security

We imagine a PPT adversary that plays the following game with a challenger. This captures the requirement that encryptions relative to ID^* should be secure even to an adversary that can obtain secret keys for polynomially many *different* identities $ID \neq ID^*$. This is called the *adaptive security* or *full security* definition. The weaker *selective* security definition restricts the adversary to pick the identity it is attacking at the very beginning of the game (before it receives MPK).

Selectively secure IBE schemes can be generically proven to be fully secure under a sub-exponentially stronger assumption. Therefore, we will not attempt to optimize the strength of the assumption and focus on selective security for this lecture.



IBE=Signatures+Public-Key Encryption

Moni Naor observed that any IBE scheme gives us *for free* a digital signature scheme. **The intuition is that the identity secret key SK_{ID} can act as a signature for the "message" ID.** How so?

- It can be generated using the master secret key MSK (which will serve as the secret signing key.)
- It can be verified using the master public key MPK – indeed, encrypt a bunch of random messages using MPK and attempt to use the "signature" to decrypt. If decryption produces the correct message, accept the signature. Otherwise, reject.
- after receiving signatures SK_{ID} on polynomially many messages ID, being able to produce the "signature" on a different message ID^* constitutes a signature forgery; but being able to do that breaks IBE security. Conversely, in a signature scheme derived from a secure IBE scheme, it should be infeasible to do that.

Indeed, turning this around, we will use the GPV signature scheme we saw in the last class as a starting point to build an IBE scheme.

7.2 Recap: GPV Signatures

- $\mathsf{KeyGen}(1^\lambda)$: Generate a random matrix $\mathbf{A} \in \mathbb{Z}_q^{n \times m}$ and its trapdoor $\mathbf{T}$ by running $\mathsf{TrapSamp}$.
- $\mathsf{Sign}(\mu)$: first compute $\mathbf{v} = H(\mu) \in \mathbb{Z}_q^n$ where H is treated as a random oracle in the analysis. Then, use Gaussian sampling (via the GPV algorithm) to compute a Gaussian solution $\mathbf{e} \in \mathbb{Z}^m$ to the equation

  $$\mathbf{A}\mathbf{e} = \mathbf{v} \pmod q$$

  Let $\Lambda^\perp(\mathbf{A})$ denote the lattice

  $$\{\mathbf{e} \in \mathbb{Z}^m : \mathbf{A}\mathbf{e} = \mathbf{0} \pmod q\}$$

  and let $\Lambda^\perp_{\mathbf{v}}(\mathbf{A})$ denote a coset of $\Lambda^\perp(\mathbf{A})$ indexed by $\mathbf{v}$. That is,

  $$\Lambda^\perp_{\mathbf{v}}(\mathbf{A}) = \{\mathbf{e} \in \mathbb{Z}^m : \mathbf{A}\mathbf{e} = \mathbf{v} \pmod q\}$$

  Note that the distribution of $\mathbf{e}$ is $D_{\Lambda^\perp_{\mathbf{v}}(\mathbf{A}),\sigma}$ where $\sigma \approx ||\mathbf{T}|| \cdot \omega(\sqrt{\log n})$. (The $\omega(\sqrt{\log n})$ is so that the sampling algorithm can achieve negligible statistical distance from a true discrete Gaussian.)
- $\mathsf{Verify}(\mathbf{A}, \mathbf{e}, \mu)$: check that (1) $\mathbf{e}$ is short, that is $||\mathbf{e}|| \leq ||T|| \cdot \omega(\sqrt{n \log n})$; and (2) $\mathbf{A}\mathbf{e} = H(\mu) \pmod q$.

The key question now is how to we build an encryption algorithm whose public key is $\mathbf{v}$ (which will be treated as $H(ID)$) and the corresponding private key is $\mathbf{e}$ as above. Indeed, we have seen precisely such a scheme in the first lecture (cf. lecture notes) called the GPV encryption scheme or more commonly, the dual-Regev encryption scheme.

But before we get there, the scheme as stated above is insecure – do you see why? Bonus points if you see how to fix it.

7.3 The Dual Regev Encryption Scheme

- KeyGen: the public key is an LWE matrix $\mathbf{A} \in \mathbb{Z}_q^{n \times m}$ and a random vector $\mathbf{v} \in \mathbb{Z}_q^n$. The private key is a short vector $\mathbf{e}$ such that $\mathbf{A}\mathbf{e} = \mathbf{v} \pmod q$.

  $$pk = (\mathbf{A}, \mathbf{v}) \quad sk = \mathbf{e}$$
- $\mathsf{Enc}(pk, \mu)$: pick an LWE secret $\mathbf{s} \in \mathbb{Z}_q^n$ and output

  $$(\mathbf{c}_1^T, c_2) := \left(\mathbf{s}^T\mathbf{A} + \mathbf{x}^T, \mathbf{s}^T\mathbf{v} + x' + m\lfloor q/2 \rfloor\right)$$

  as the ciphertext. We will call this ciphertext the dual Regev encryption of μ relative to $\mathbf{A}$ and $\mathbf{v}$.

- $\mathsf{Dec}(sk, (\mathbf{c}_1^T, c_2))$: Compute
$$\tilde{\mu} := \mathsf{Round}(c_2 - \mathbf{c}_1^T \mathbf{e})$$
where $\mathsf{Round}(\alpha)$ outputs 1 if $|\alpha - q/2| \le q/4$ and 0 otherwise.

We will leave the correctness and security as an exercise. (Alternatively, look at lecture 1.)

7.4 The GPV IBE Scheme

- $\mathsf{Setup}(1^\lambda)$: Pick the right $n = n(\lambda)$ for a security level of λ bits. Generate a matrix $\mathbf{A} \in \mathbb{Z}_q^{n\times m}$ and its trapdoor $\mathbf{T} \in \mathbb{Z}^{m\times m}$ by running the trapdoor sampling algorithm.
$$(\mathbf{A}, \mathbf{T}) \leftarrow \mathsf{TrapSamp}(1^n)$$
(The parameters m and q are picked internally by the trapdoor sampling algorithm.) The master public key is $\mathsf{mpk} = \mathbf{A}$ and the master secret key is $\mathsf{msk} = \mathbf{T}$.

- $\mathsf{KeyGen}(\mathsf{msk}, ID)$: Compute $\mathbf{v} := H(ID) \in \mathbb{Z}_q^n$ where $H : \{0,1\}^* \to \mathbb{Z}_q^n$ is a hash function (which, in the security analysis, will be treated as a random oracle.) Generate a short vector
$$\mathbf{e} \leftarrow \mathsf{DGSamp}(\mathbf{A}, \mathbf{T}, \mathbf{v})$$
by running the discrete Gaussian sampling algorithm. Recall that $\mathbf{Ae} = \mathbf{v} \pmod q$. Output the secret key $sk_{ID} = \mathbf{e}$.

- $\mathsf{Enc}(\mathsf{mpk}, ID, \mu)$: Run the dual Regev encryption algorithm with $pk := (\mathbf{A}, \mathbf{v} = H(ID))$ and message μ and output the resulting ciphertext.

- $\mathsf{Dec}(sk_{ID}, c)$: Run the dual Regev decryption algorithm with $sk := sk_{ID} = \mathbf{e}$.

Proof of (Full) Security

We will come up with alternate algorithms called $\mathsf{Setup}^*, \mathsf{KeyGen}^*$ and Enc^* (Dec^* will be the same as Dec) which the challenger will run. Our goal will be to show that (1) the adversary cannot distinguish between the challenger running Algorithm *vs* Algorithm* and (2) Algorithms* do not need the master secret key and moreover, a challenger using Algorithm* can use a successful adversary to break LWE.

A crucial advantage of Algorithm* for the GPV scheme is that it can use the programmability of the random oracle as we will see below. We will for simplicity first create algorithms for the selective security game.

- $\mathsf{Setup}^*(ID^*, 1^\lambda)$: Sample random $\mathbf{A}^*$ which forms the MPK^* (no need for trapdoor).

- $\mathsf{Hash}^*(ID)$: Set $H(ID^*) = \mathbf{v}^*$, a random vector in $\mathbb{Z}_q^n$. For all other IDs, set $H(ID) = \mathbf{A}^*\mathbf{e}_{ID}$ where $\mathbf{e}_{ID}$ is chosen from a Gaussian. Remember $\mathbf{e}_{ID}$.

- $\mathsf{KeyGen}^*(ID)$: We know that $ID \neq ID^*$. So, we know the $\mathbf{e}_{ID}$ by construction! This is a consequence of working in the random oracle model!

- $\mathsf{Enc}^*(MPK^*, ID^*, \mu)$: return the dual Regev encryption of μ relative to $\mathbf{A}^*$ and $\mathbf{v}^*$.

The Algorithm* produce the same distribution as the original algorithms. Thus, an adversary will break the challenge ciphertext when interacting with Algorithm* just as well as with Algorithms. By embedding the dual-Regev challenge matrix $\mathbf{A}$ as the master public key and the dual-Regev public key $\mathbf{v}^*$ as the hash of ID^*, we can easily turn the IBE adversary into an attack against the dual Regev public key encryption scheme.

A Note on Full Security. Since Setup^* does not know ID^*, it guesses which of the (polynomially many) hash queries will be for ID^*. (1) any adversary that succeeds has to know $H(ID^*)$ which it can only find out by making a hash query; and (2) if the guess is correct (happens with probability $1/Q$) we can translate an IBE breaker into a dual-Regev breaker just as above.

7.5 The CHKP IBE Scheme

The CHKP Trick: Trapdoor Extension.
Given the trapdoor for a matrix $\mathbf{A}$, can you generate a trapdoor for $[\mathbf{A}||\mathbf{B}]$ where $\mathbf{B}$ is an arbitrary matrix?

The Scheme

- $\mathsf{Setup}(1^\lambda)$: Pick the right $n = n(\lambda)$ for a security level of λ bits. Generate matrices

$$\mathbf{A}_{1,0}, \mathbf{A}_{1,1}, \ldots, \mathbf{A}_{\ell,0}, \mathbf{A}_{\ell,1} \in \mathbb{Z}_q^{n\times m}$$

  where ℓ is the length of the identities. The master public key is

$$\mathsf{mpk} = \left(\mathbf{A}_{i,b}\right)_{i\in[\ell],b\in\{0,1\}}, \mathbf{v}$$

  where $\mathbf{v} \in \mathbb{Z}_q^n$ is a random vector, and the master secret key is

$$\mathsf{msk} = (\mathbf{T}_{\mathbf{A}_0}, \mathbf{T}_{\mathbf{A}_1})$$

  We will never use the trapdoors for the other matrices (except in the security proof.)

- $\mathsf{KeyGen}(\mathsf{msk}, ID \in \{0,1\}^\ell)$: Let

$$\mathbf{A}_{ID} := [\mathbf{A}_{1,ID_1}||\mathbf{A}_{2,ID_2}||\ldots||\mathbf{A}_{\ell,ID_\ell}]$$

  where $ID_1, \ldots, ID_\ell$ are the bits of ID. Generate a short vector $\mathbf{e} \leftarrow \mathsf{DGSamp}(\mathbf{A}_{ID}, \mathbf{T}_{\mathbf{A}_{ID}}, \mathbf{v})$ by running the discrete Gaussian sampling algorithm. Recall that $\mathbf{A}_{ID} \cdot \mathbf{e} = \mathbf{v} \pmod q$. Output the secret key $sk_{ID} = \mathbf{e}$.

- $\mathsf{Enc}(\mathsf{mpk}, ID, \mu)$: Run the dual Regev encryption algorithm with $pk := (\mathbf{A}_{ID}, \mathbf{v})$ and message μ and output the resulting ciphertext.

- $\mathsf{Dec}(sk_{ID}, c)$: Run the dual Regev decryption algorithm with $sk := sk_{ID} = \mathbf{e}$.

Proof of (Selective) Security

As before, we will come up with alternate algorithms called $\mathsf{Setup}^*, \mathsf{KeyGen}^*$ and Enc^* (Dec^* will be the same as Dec) which the challenger will run. We will not be able to use random oracles here.

- $\mathsf{Setup}^*(ID^*, 1^\lambda)$: sample random $\mathbf{v}^*$. sample ℓ random matrices $\mathbf{B}_1, \ldots, \mathbf{B}_\ell$ and set

  $$\mathbf{A}_{i,ID_i^*} = \mathbf{B}_i$$

  sample ℓ matrices $\mathbf{B}'_1, \ldots, \mathbf{B}'_\ell$ together with their trapdoors and set

  $$\mathbf{A}_{i,1-ID_i^*} = \mathbf{B}'_i$$

  MPK^* consists of all the $\mathbf{A}_{i,b}$ and $\mathbf{v}^*$. MSK^* consists of the trapdoors of all $\mathbf{A}_{i,1-ID_i^*}$.

- $\mathsf{KeyGen}^*(ID)$: We know that $ID \neq ID^*$. Therefore, I know the trapdoor of the matrix

  $$\mathbf{A}_{ID} := [\mathbf{A}_{1,ID_1}|| \ldots ||\mathbf{A}_{\ell,ID_\ell}]$$

  (do you see why?)

- $\mathsf{Enc}^*(MPK^*, ID^*, \mu)$: return the dual Regev encryption of μ relative to $\mathbf{A}_{ID^*}$ and $\mathbf{v}^*$. (note that MSK^* does not tell us anything about a trapdoor for $\mathbf{A}_{ID^*}$.)

One can also prove full security with a more sophisticated proof. In one sentence, the idea is to set up $\mathbf{A}_{i,b}$ so that Algorithm* can generate secret keys for all the Q secret key queries and yet *not* be able to generate the secret key for ID^*.

CHKP: Pros and Cons

- PLUS: the scheme is secure without resorting to the random oracle model.
- MINUS: the public parameters are rather large, namely $O(nm \log q \cdot \ell)$ as opposed to GPV where it is $O(nm \log q)$. Consequently, also ciphertexts are large.
- PLUS: While we only showed selective security, one can augment the scheme to be adaptively (fully) secure.
- PLUS: The scheme naturally extends to a hierarchical IBE scheme, described next.

A Brief Note on Hierarchical IBE

Think of hierarchies in an organization. The CEO (the master key generator) can delegate access to the VP of Engineering who can in turn delegate to programmers and so forth (but not the other way round). In a hierarchical IBE, one can generate SK_{ID} using MSK; in turn, the owner of SK_{ID} can generate $SK_{ID||ID'}$ etc.

The CHKP scheme has a natural hierarchical structure. Namely, if you know the trapdoor for $\mathbf{A}_{ID}$, you can generate a trapdoor for $\mathbf{A}_{ID||ID'} = [\mathbf{A}_{ID}||\mathbf{A}_{ID'}]$. Constructing a HIBE scheme building off of this idea is left as an exercise.

7.6 The ABB IBE Scheme

The ABB Trick: Punctured Trapdoors.
Given the trapdoor for a matrix $\mathbf{A}_0$, a matrix $\mathbf{R}$ with small entries, and a trapdoor for $\mathbf{G}$, can you generate a trapdoor for

$$[\mathbf{A}_0||\mathbf{A}_0\mathbf{R} + \alpha \cdot \mathbf{G}]$$

for an arbitrary integer $\alpha \neq 0 \pmod q$?

How about for $\alpha = 0 \pmod q$, that is, $[\mathbf{A}_0||\mathbf{A}_0\mathbf{R}]$?

The Scheme

- $\mathsf{Setup}(1^\lambda)$: Pick the right $n = n(\lambda)$ for a security level of λ bits. Generate matrices

  $$\mathbf{A}_0, \mathbf{A}_1 \in \mathbb{Z}_q^{n\times m}$$

  The master public key is

  $$\mathsf{mpk} = \mathbf{A}_0, \mathbf{A}_1, \mathbf{v}$$

  where $\mathbf{v} \in \mathbb{Z}_q^n$ is a random vector, and the master secret key is

  $$\mathsf{msk} = \mathbf{T}_{\mathbf{A}_0}$$

  We will never use the trapdoor for $\mathbf{A}_1$.

- $\mathsf{KeyGen}(\mathsf{msk}, ID \in \{0,1\}^\ell)$: Let h be a collision-resistant hash function that maps identities to $\mathbb{Z}_q^*$. Define

  $$\mathbf{A}_{ID} := [\mathbf{A}_0||\mathbf{A}_1 + h(ID)\cdot\mathbf{G}]$$

  where $\mathbf{G}$ is the gadget matrix. Note that by trapdoor extension, KeyGen knows a trapdoor for $\mathbf{A}_{ID}$ for any ID.

  Generate a short vector $\mathbf{e} \leftarrow \mathsf{DGSamp}(\mathbf{A}_{ID}, \mathbf{T}_{\mathbf{A}_{ID}}, \mathbf{v})$ by running the discrete Gaussian sampling algorithm. Recall that $\mathbf{A}_{ID}\cdot\mathbf{e} = \mathbf{v} \pmod q$. Output the secret key $sk_{ID} = \mathbf{e}$.

- $\mathsf{Enc}(\mathsf{mpk}, ID, \mu)$: Run the dual Regev encryption algorithm with $pk := (\mathbf{A}_{ID}, \mathbf{v})$ and message μ and output the resulting ciphertext.

- $\mathsf{Dec}(sk_{ID}, c)$: Run the dual Regev decryption algorithm with $sk := sk_{ID} = \mathbf{e}$.

ABB: Proof of Selective Security

As before, we will come up with a bunch of alternate algorithms called $\mathsf{Setup}^*, \mathsf{KeyGen}^*$ and Enc^* (Dec^* will be the same as Dec) which the challenger will run. We will not be able to use random oracles here either.

- $\mathsf{Setup}^*(ID^*, 1^\lambda)$: sample random $\mathbf{v}^*$. sample a random matrix $\mathbf{A}_0$ and a matrix $\mathbf{R}$ with small entries. Set

  $$\mathbf{A}_1 := [\mathbf{A}_0||\mathbf{A}_0\mathbf{R} - h(ID^*)\mathbf{G}]$$

  MPK^* consists of $\mathbf{A}_0, \mathbf{A}_1$ and $\mathbf{v}^*$. MSK^* consists of $\mathbf{R}$ (and the trapdoor for $\mathbf{G}$.)

- $\mathsf{KeyGen}^*(ID)$: We know that $ID \neq ID^*$. Therefore, I know the trapdoor of the matrix

$$\mathbf{A}_{ID} := [\mathbf{A}_0 || \mathbf{A}_1 + h(ID)\mathbf{G}] = [\mathbf{A}_0 || \mathbf{A}_0\mathbf{R} + (h(ID) - h(ID^*)\mathbf{G}]$$

  (do you see why?)

- $\mathsf{Enc}^*(MPK^*, ID^*, \mu)$: given a dual Regev encryption of μ relative to $\mathbf{A}_0$ and $\mathbf{v}^*$, compute a dual Regev encryption of μ relative to

$$\mathbf{A}_{ID^*} = [\mathbf{A}_0 || (\mathbf{A}_0\mathbf{R} - h(ID^*)\mathbf{G}) + h(ID^*)\mathbf{G}] = [\mathbf{A}_0 || \mathbf{A}_0\mathbf{R}]$$

  and $\mathbf{v}^*$. (do you see how to do this?)

ABB: Pros and Cons

- PLUS: the scheme is secure without resorting to the random oracle model.
- PLUS: the public parameters and ciphertexts are as small as GPV, namely $O(nm \log q)$.
- PLUS: Can be extended to full security.
- PLUS: Extensible to hierarchical IBE. A different ABB paper uses additional techniques to construct a "better" HIBE (where the lattice dimension stays the same regardless of the number of levels of delegation).

7.7 Application: Chosen Ciphertext Secure Public-key Encryption

We will now show a very simple construction of a chosen ciphertext secure (CCA2-secure) public-key encryption scheme from IBE. This is due to Canetti, Halevi and Katz [CHK04]. In fact, here we will describe a solution for the weaker notion of CCA1-security.

But first, the definition of CCA1-security. In the CCA1 game, the adversary gets the public-key PK of the encryption scheme, and can ask to get polynomially many ciphertexts decrypted. That is, a challenger will, on input c, run $\mathsf{Dec}(SK, c)$ and return the answer to the adversary. Note that c need not be distributed like an honestly generated ciphertext, and may not even live in the range of the encryption algorithm (i.e., may not be a valid ciphertext). Eventually, the adversary gets an encryption of a random bit b under PK and is asked to guess b. CCA1 security requires that no PPT adversary can guess b with probability better than $1/2 + \mathsf{negl}(\lambda)$.

Here is the construction.

- $\mathsf{KeyGen}(1^\lambda)$: run $\mathsf{IBE.Setup}(1^\lambda)$ to get an MPK_{IBE} and an MSK_{IBE}. The public key PK of the CCA scheme is MPK_{IBE} and the secret key SK is MSK_{IBE}.
- $\mathsf{Enc}(PK, \mu)$: pick a random string ID. Run $\mathsf{IBE.Enc}(PK = MPK_{IBE}, ID, \mu)$ and output ID together with the resulting ciphertext.
- $\mathsf{Dec}(SK, (ID, c))$: use $SK = MSK_{IBE}$ to create SK_{ID} and run the IBE decryption algorithm $\mu = \mathsf{IBE.Dec}(SK_{ID}, c)$.

The CCA security proof is super simple. The intuition?

- the decryption algorithm only uses SK_{ID} (and not the MSK per se) and
- the identity in the challenge ciphertext is random and hence different w.h.p. from the (adversarially chosen) identities in all the decryption queries.

Put together, IBE security should say that breaking the security of the challenge ciphertext is hard.

7.8 Registration-based Encryption

We will say just a few words about RBE here. Recall from the beginning of the lecture that a major disadvantage of IBE is the power of the master key authority to decrypt all ciphertexts.

A completely orthogonal approach which does not have this problem starts from the following strawman scheme: the master public key, curated by the authority, is the concatenation of all the users' public keys... Of course, this leads us back to exactly the PKI problem we wanted to solve. However, it is possible that the authority can publish a *short digest* of the concatenation of all public keys, which is nevertheless good enough for encryption (although it should not be clear exactly how yet!)

It turns out that this idea can be brought to fruition using the methodology of deferred encryption due to Garg et al. We refer the reader to the papers [GHMR18, GHM$^+$19]. The construction proceeds in a completely different way from everything we saw today, and is quite inefficient. An open problem is to come up with an RBE that is as efficient as (or more efficient than!) the IBE schemes we saw here.

CHAPTER 8

Encrypted Computation from Lattices

In this lecture, we will explore various facets of encrypted computation which, generally speaking, refers to the set of cryptographic tasks where you *encrypt* computational objects – for example, a program or a circuit and/or its input – in a way that anyone holding these encrypted objects can perform meaningful manipulations on them. Examples include (fully) homomorphic encryption, (various flavors of) attribute-based encryption, (fully) homomorphic signatures, constrained pseudorandom functions, functional encryption and indistinguishability obfuscation.

We will see constructions of all but the last two in this lecture. Indeed, we will present a single lattice tool, the *key lattice equation*, that will give us *all* these constructions.

8.1 Fully Homomorphic Encryption

In a fully homomorphic (private or public-key) encryption, anyone can take a set of encrypted messages $\mathsf{Enc}(x_1), \ldots, \mathsf{Enc}(x_k)$ and produce an encryption of *any polynomial-time computable function* of them, that is, $\mathsf{Enc}(f(x_1, \ldots, x_k))$ where f is any function with a $\mathsf{poly}(\lambda)$-size circuit. By a result of Rothblum, any private-key (even additively) homomorphic encryption scheme can be converted to a public-key homomorphic scheme, so we will focus our attention on private-key schemes henceforth.

The formal definition of the functionality of fully homomorphic encryption follows.

- $\mathsf{KeyGen}(1^\lambda)$: produces a secret key sk, possibly together with a public evaluation key ek.
- $\mathsf{Enc}(sk, \mu)$, where $\mu \in \{0, 1\}$: produces a ciphertext c.
- $\mathsf{Dec}(sk, c)$: outputs μ.
  // So far, everything is exactly as in a regular secret-key encryption scheme.
- $\mathsf{Eval}(\mathsf{ek}, f, c_1, \ldots, c_k)$ takes as input a $\mathsf{poly}(\lambda)$-size circuit that computes a function $f : \{0, 1\}^k \to \{0, 1\}$, as well as k ciphertexts $c_1, \ldots, c_k$, and outputs a ciphertext c_f.

Correctness says that

$$\mathsf{Dec}(sk, \mathsf{Eval}(\mathsf{ek}, f, \mathsf{Enc}(sk, \mu_1), \ldots, \mathsf{Enc}(sk, \mu_k))) = f(\mu_1, \ldots, \mu_k)$$

for all $f, \mu_1, \ldots, \mu_k$ with probability 1 over the sk, ek and the randomness of all the algorithms.

Security is just semantic (IND-CPA) security, that is the encryptions of any two sequences of messages $(\mu_i)_{i \in \mathsf{poly}(\lambda)}$ and $(\mu_i')_{i \in \mathsf{poly}(\lambda)}$ are computationally indistinguishable. (the fact that the encryption scheme is homomorphic is a functionality requirement, and does not change the notion of security.)

$$\left(\mathsf{Enc}(sk, \mu_1), \ldots, \mathsf{Enc}(sk, \mu_{p(\lambda)})\right) \approx_c \left(\mathsf{Enc}(sk, \mu_1'), \ldots, \mathsf{Enc}(sk, \mu_{p(\lambda)}'')\right)$$

A final and important property is compactness, that is, $|c_f| = \mathsf{poly}(\lambda)$, independent of the circuit size of f. (Weaker compactness conditions are possible, and indeed, we will see one in the sequel.)

8.2 The GSW Scheme

The first candidate FHE scheme was due to Gentry in 2009. The first LWE-based FHE Scheme was due to Brakerski and Vaikuntanathan in 2011. We will present a different FHE scheme due to Gentry, Sahai and Waters (2013) which is both simple and quite flexible.

- KeyGen: the secret key is a vector $\mathbf{s} = \begin{bmatrix} \mathbf{s}' \\ -1 \end{bmatrix}$ where $\mathbf{s}' \in \mathbb{Z}_q^n$.
- Enc: output $\mathbf{A} + \mu\mathbf{G}$ where $\mathbf{A}$ is a random matrix such that

  $$\mathbf{s}^T\mathbf{A} \approx \mathbf{0} \pmod q$$

  Here is one way to do it: choose a random matrix $\mathbf{A}'$ and let

  $$\mathbf{A} := \begin{bmatrix} \mathbf{A}' \\ (\mathbf{s}')^T\mathbf{A}' + \mathbf{e}' \end{bmatrix}$$

- Dec: exercise.
- Eval: we will show how to ADD (over the integers) and MULT (mod 2) the encrypted bits which will suffice to compute all Boolean functions.

8.3 How to Add and Multiply (without errors)

Let's start with a variant of the scheme where the ciphertext is

$$\mathbf{C} = \mathbf{A} + \mu\mathbf{I}$$

where $\mathbf{I}$ is the identity matrix and $\mathbf{s}^T\mathbf{A} = \mathbf{0}$ (as opposed to $\mathbf{s}^T\mathbf{A} \approx \mathbf{0}$.)

Now,

$$\mathbf{s}^T\mathbf{C} = \mu\mathbf{s}^T$$

- ADD($\mathbf{C}_1, \mathbf{C}_2$) outputs $\mathbf{C}_1 + \mathbf{C}_2$. This is an encryption of $\mu_1 + \mu_2$ since

$$\mathbf{s}^T(\mathbf{C}_1 + \mathbf{C}_2) = (\mu_1 + \mu_2)\mathbf{s}^T$$

  Eigenvalues add.

- MULT($\mathbf{C}_1, \mathbf{C}_2$) outputs $\mathbf{C}_1\mathbf{C}_2$. This is an encryption of $\mu_1\mu_2$ since

$$\mathbf{s}^T(\mathbf{C}_1\mathbf{C}_2) = \mu_1\mathbf{s}^T\mathbf{C}_2 = \mu_1\mu_2\mathbf{s}^T$$

  Eigenvalues multiply.

We need one ingredient now to turn this into a *real* FHE scheme.

8.4 How to Add and Multiply (without errors)

We have to be careful to multiply approximate equations by small numbers. Once we make adjustments to this effect, we get the GSW scheme. The ciphertext is

$$\mathbf{C} = \mathbf{A} + \mu\mathbf{G}$$

where $\mathbf{s}^T\mathbf{A} \approx \mathbf{0}$. Think of $\mathbf{G}$ as an error correcting artifact for the message μ.

Now,

$$\mathbf{s}^T\mathbf{C} \approx \mu\mathbf{s}^T\mathbf{G}$$

which is the approximate eigenvalue equation.

- ADD($\mathbf{C}_1, \mathbf{C}_2$) outputs $\mathbf{C}_1 + \mathbf{C}_2$. This is an encryption of $\mu_1 + \mu_2$ since

$$\mathbf{s}^T(\mathbf{C}_1 + \mathbf{C}_2) \approx (\mu_1 + \mu_2)\mathbf{s}^T\mathbf{G}$$

  Approximate eigenvalues add (if you don't do it too many times.)

- MULT($\mathbf{C}_1, \mathbf{C}_2$) outputs $\mathbf{C}_1\mathbf{G}^-(\mathbf{C}_2)$. This is an encryption of $\mu_1\mu_2$ since

$$\mathbf{s}^T(\mathbf{C}_1\mathbf{G}^-(\mathbf{C}_2)) = (\mathbf{s}^T\mathbf{C}_1)\mathbf{G}^-(\mathbf{C}_2) \approx (\mu_1\mathbf{s}^T\mathbf{G})\mathbf{G}^-(\mathbf{C}_2) = \mu_1(\mathbf{s}^T\mathbf{C}_2) \approx \mu_1\mu_2\mathbf{s}^T\mathbf{G}$$

  where the first $\approx$ is because $\mathbf{G}^-(\mathbf{C}_2)$ is small and the second $\approx$ because μ_1 is small.

  Approximate eigenvalues multiply if you only multiply by small numbers/matrices.

Put together, it is not hard to check that you can evaluate depth-d circuits of NAND gates with error growth $m^{O(d)}$. (You can do better for log-depth circuits by converting them to branching programs; see Brakerski-Vaikuntanathan 2014.)

8.5 Bootstrapping to an FHE

With this, we get a *leveled FHE* scheme. That is, we can set parameters (in particular $q = m^{\Omega(d)}$) such that the scheme is capable of evaluating depth-d circuits. What if we want to set parameters such that the scheme can evaluate circuits of *any polynomial depth*? That would be an FHE scheme for real.

The only way we know to construct an FHE scheme at this point is using Gentry's bootstrapping technique which we describe below. Doing so involves making an additional assumption on the *circular security* of the GSW encryption scheme, which we don't know how to reduce to LWE.

The Idea

Assume that you are the homomorphic evaluator and in the course of homomorphic evaluation, you get two ciphertexts C and C' which are (a) *decryptable* to μ and μ' respectively, in the sense that their decryption noise has ℓ_∞ norm less than $q/4$; but (b) *not computable*, in the sense that they will become undecryptable after another homomorphic evaluation, say of a NAND. What should you do with these ciphertexts?

Here is an idea: If you had the secret key, you could decrypt C and C', re-encrypt them with *fresh small* noise and proceed with the computation. In fact, you could do this after every gate. But this is clearly silly. If you had the secret key, why bother with encrypted computation in the first place?

Here is a better idea: assume that you have *a ciphertext $\tilde{C}$ of the FHE secret key encrypted under the secret key* itself (a so-called "circular encryption"). Then, you could homomorphically evaluate the following circuit on input $\tilde{C}$:

$$\mathsf{BootNAND}_{C,C'}(sk) = \mathsf{Dec}_{sk}(C) \ \ \mathsf{NAND} \ \ \mathsf{Dec}_{sk}(C')$$

What you get out is an encryption of μ NAND μ'. How did this happen (and what *did* happen?) First of all, note that $\widetilde{C}$ is a *fresh* encryption of sk. Secondly, assume that the $\mathsf{BootNAND}$ circuit (which is predominantly the decryption circuit) has small depth, small enough that the homomorphic evaluation can handle it. The output of the circuit on input sk is indeed μ NAND μ'; therefore, putting together this discussion, the output of the homomorphic evaluation of the circuit is *an encryption of* μ NAND μ' under sk.

Once we can implement $\mathsf{BootNAND}$, this is how we evaluate every NAND gate. You get as input two ciphertexts C and C'. You **do not** homomorphically evaluate on them, as then you will get garbage. Instead, use them to construct the circuit $\mathsf{BootNAND}_{C,C'}$. and homomorphically evaluate it on an encryption $\widetilde{C}$ of the secret key sk that you are given as an additional *evaluation key*.

Voila! This gives us a fully homomorphic encryption scheme.

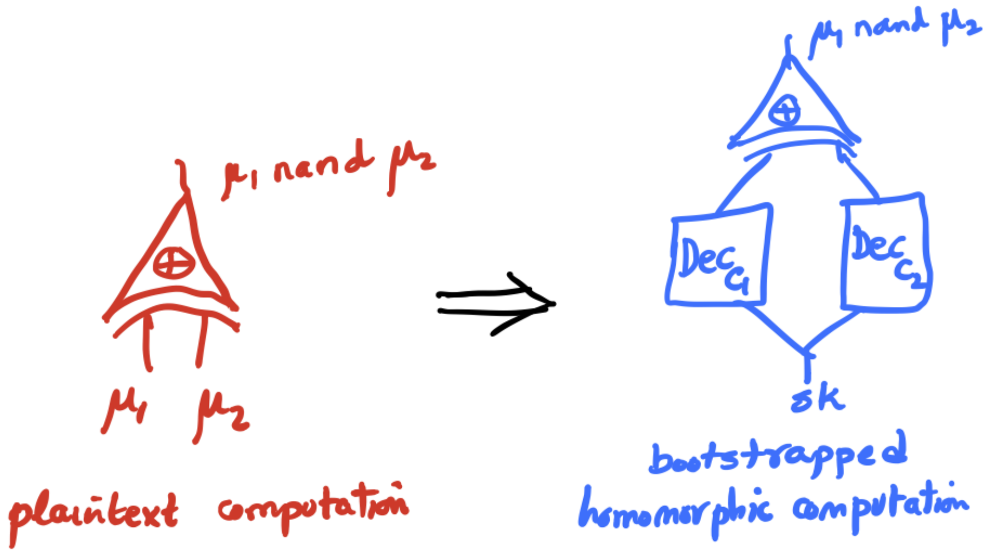
Circular Security

Is it OK to publish a circular encryption? Does the IND-CPA security of the scheme hold when the adversary additionally gets such an encryption? First of all, the IND-CPA security of the underlying encryption scheme (GSW in this case) alone does not tell us anything about what happens in this scenario. Indeed, you can construct an IND-CPA secure encryption scheme whose security *breaks completely* given such a circular encryption. (I will leave it as an exercise.)

Secondly, and quite frustratingly, we do know how to show that the Regev encryption scheme is circular-secure assuming LWE, but showing that the GSW scheme is circular-secure is one of my favorite open problems in lattice-based cryptography.

8.6 The Key Equation

Let us abstract out the mathematics behind GSW into a *key lattice equation* which will guide us through constructing the rest of the primitives in this lecture.



Recall the approximate eigenvector relation:

$$\mathbf{s}^T \mathbf{A}_i \approx \mu_i \mathbf{s}^T \mathbf{G}$$

and rewrite it as

$$\mathbf{s}^T(\mathbf{A}_i - \mu_i \mathbf{G}) \approx \mathbf{0} \tag{8.1}$$

Let $\mathbf{A}_f$ be the homomorphically evaluated ciphertext for a function f. We know that

$$\mathbf{s}^T \mathbf{A}_f \approx f(\boldsymbol{\mu})\mathbf{s}^T \mathbf{G}$$

or

$$\mathbf{s}^T(\mathbf{A}_f - f(\boldsymbol{\mu})\mathbf{G}) \approx \mathbf{0} \tag{8.2}$$

We will generalize this to arbitrary matrices $\mathbf{A}_1, \ldots, \mathbf{A}_\ell$ – not necessarily ones that share the same eigenvector.

First, we know that $\mathbf{A}_f$ is a function of $\mathbf{A}_1, \ldots, \mathbf{A}_\ell$ and f (*but not* $\mu_1, \ldots, \mu_\ell$). Henceforth, when we say $\mathbf{A}_f$, we will mean a matrix obtained by the GSW homomorphic evaluation procedure. (That is, homomorphic addition of two matrices is matrix addition; homomorphic multiplication is matrix multiplication after bit-decomposing the second matrix).

Second, and very crucially, we can show that for any sequence of matrices $\mathbf{A}_1, \ldots, \mathbf{A}_\ell$,

$$[\mathbf{A}_1 - \mu_1\mathbf{G} || \ldots || \mathbf{A}_\ell - \mu_\ell \mathbf{G}] \; \mathbf{H}_{f,\boldsymbol{\mu}} = \mathbf{A}_f - f(\boldsymbol{\mu})\mathbf{G}$$

where $\mathbf{H}_{f,\boldsymbol{\mu}}$ is a matrix with small coefficients. We call this the key lattice equation.

To see this for addition, notice that

$$[\mathbf{A}_1 - \mu_1\mathbf{G}||\mathbf{A}_2 - \mu_2\mathbf{G}] \underbrace{\begin{bmatrix} \mathbf{I} \\ \mathbf{I} \end{bmatrix}}_{\mathbf{H}_{+,\mu_1,\mu_2}} = \mathbf{A}_1 + \mathbf{A}_2 - (\mu_1+\mu_2)\mathbf{G} = \mathbf{A}_+ - (\mu_1+\mu_2)\mathbf{G}$$

and for multipication,

$$[\mathbf{A}_1 - \mu_1\mathbf{G}||\mathbf{A}_2 - \mu_2\mathbf{G}] \underbrace{\begin{bmatrix} \mathbf{G}^-(\mathbf{A}_2) \\ \mu_1\mathbf{I} \end{bmatrix}}_{\mathbf{H}_{\times,\mu_1,\mu_2}} = \mathbf{A}_1\mathbf{G}^-(\mathbf{A}_2) - \mu_1\mu_2\mathbf{G} = \mathbf{A}_\times - \mu_1\mu_2\mathbf{G}$$

By composition, we get that

$$[\mathbf{A}_1 - \mu_1\mathbf{G}||\mathbf{A}_2 - \mu_2\mathbf{G}||\dots||\mathbf{A}_\ell - \mu_\ell\mathbf{G}]\ \mathbf{H}_{f,\boldsymbol{\mu}} = \mathbf{A}_f - f(\boldsymbol{\mu})\mathbf{G}$$

where $\mathbf{H}_{f,\mu}$ is a matrix with small entries (roughly proportional to $m^{O(d)}$ where d is the circuit depth of f).

An Advanced Note: Given arbitrary matrices $\mathbf{A}_i$ and $\mathbf{A}_f$, there exists such a small matrix $\mathbf{H}$; but if $\mathbf{A}_f$ is arbitrary, it is hard to find.

Let's re-derive FHE from the key equation:

- The ciphertexts are the matrices $\mathbf{A}_i$ and we picked them such that

$$\mathbf{s}^T\mathbf{A} \approx \mu\mathbf{s}^T\mathbf{G}$$

- Homomorphic evaluation is computing $\mathbf{A}_f$ starting from $\mathbf{A}_1, \dots, \mathbf{A}_\ell$.
- Correctness of homomorphic eval follows from the key equation: We know that

$$\mathbf{s}^T[\mathbf{A}_1 - \mu_1\mathbf{G}||\dots||\mathbf{A}_\ell - \mu_\ell\mathbf{G}] \approx \mathbf{0}$$

  by the equation above that characterizes ciphertexts. Therefore, by the key equation,

$$\mathbf{s}^T[\mathbf{A}_f - f(\boldsymbol{\mu})\mathbf{G}] = \mathbf{s}^T[\mathbf{A}_1 - \mu_1\mathbf{G}||\dots||\mathbf{A}_\ell - \mu_\ell\mathbf{G}]\ \mathbf{H}_{f,\boldsymbol{\mu}} \approx \mathbf{0}$$

  as well meaning that $\mathbf{A}_f$ is an encryption of $f(\boldsymbol{\mu})$. Note that no one needs to know or compute the matrix $\mathbf{H}$; it only appears in the analysis.

8.7 Fully Homomorphic Signatures

We will use the key equation quickly in succession to derive three applications. The first is fully homomorphic signatures (FHS). Here is a first take in defining what one might want from an FHS scheme: a way to take a bunch of messages $\mu_1, \ldots, \mu_\ell$ together with their signatures $\sigma_1, \ldots, \sigma_\ell$ that verify under a public key PK and compute a signature σ_f of the message $f(\mu_1, \ldots, \mu_\ell)$ that verifies under PK (for any function f).

However, this is meaningless. You could produce signatures for constant functions $f_\alpha(x) = \alpha$ and thereby forge the signature on *any message* whatsoever.

Rather, what we need from an FHS is that it produces a signature σ_f that *binds* the output of a computation $f(\boldsymbol{\mu})$ with the computation itself f. Here is the definition:

1. $PK, f \to PK_f$.
2. $(\mu_1, \sigma_1), \ldots, (\mu_\ell, \sigma_\ell) \to (f(\boldsymbol{\mu}), \sigma_f)$.
   // Both the operations above are as expensive as computing f.
3. $\mathsf{Verify}(PK_f, f(\boldsymbol{\mu}), \sigma_f) = 1$.
4. For any f and any $y \neq f(\boldsymbol{\mu})$, no PPT adversary can produce a (fake) signature σ' such that $\mathsf{Verify}(PK_f, y, \sigma') = 1$ (except with negligible probability.)

Why is this useful? An application is (online-offline) verifiable delegation of computation.

Here is a basic construction using the key equation.

- PK is $\mathbf{B}, \mathbf{A}_1, \ldots, \mathbf{A}_\ell$. (This scheme can sign ℓ bits). SK is trapdoor of $\mathbf{B}$.
- Signature σ_i for a message μ_i is a short $\mathbf{R}_i$ such that $\mathbf{B}\mathbf{R}_i = \mathbf{A}_i - \mu_i \mathbf{G}$.
  // (Can you see how the signing algorithm works?)
- PK_f is $\mathbf{A}_f$.
- To homomorphically compute on the signatures, start from the key equation:

  $$[\mathbf{A}_1 - \mu_1\mathbf{G} || \ldots || \mathbf{A}_\ell - \mu_\ell \mathbf{G}]\ \mathbf{H}_{f,\boldsymbol{\mu}} = \mathbf{A}_f - f(\boldsymbol{\mu})\mathbf{G}$$

  Notice that the way we constructed signatures,

  $$\mathbf{B}\ \underbrace{[\mathbf{R}_1 || \ldots || \mathbf{R}_\ell]\ \mathbf{H}_{f,\boldsymbol{\mu}}}_{\sigma_f} = \underbrace{\mathbf{A}_f}_{PK_f} - f(\boldsymbol{\mu})\mathbf{G}$$

  σ_f is thus the homomorphic signature of $f(\boldsymbol{\mu})$ under PK_f.
- Why can't an adversary cheat? Suppose an adversary produces a signature σ' that verifies for the message $y \neq f(\boldsymbol{\mu})$ w.r.t. PK_f. So,

  $$\mathbf{B}\sigma' = \mathbf{A}_f - y\mathbf{G}$$

  Subtracting the last two equations, we get

  $$\mathbf{B}(\sigma' - \sigma_f) = (f(\boldsymbol{\mu}) - y)\mathbf{G}$$

  So, $\sigma' - \sigma_f$ is an inhomogenous trapdoor for $\mathbf{B}$, constructing which breaks SIS.

8.8 Attribute-based Encryption

Attribute-based encryption (ABE) generalizes IBE in the following way.

- Setup produces MPK, MSK.
- Enc uses MPK to encrypt a message m relative to attributes $(\mu_1, \ldots, \mu_\ell) \in \{0,1\}^\ell$.

  (In an IBE scheme, $\boldsymbol{\mu} = ID$.)

- KeyGen uses MSK to generate a secret key SK_f for a given Boolean function $f : \{0,1\}^\ell \to \{0,1\}$.

  (IBE is the same as ABE where f is restricted to be the point (delta) function $f_{ID'}(ID) = 1$ iff $ID = ID'$.)

- Dec gets $\boldsymbol{\mu}$ (attributes are in the clear) and uses SK_f to decrypt a ciphertext C if $f(\boldsymbol{\mu}) = 1$ (true). If $f(\boldsymbol{\mu}) = 0$, Dec simply outputs $\perp$.

Here is an ABE scheme (called the BGG+ scheme) using the key equation. It's best to view this as a generalization of the Agrawal-Boneh-Boyen IBE scheme.

- KeyGen outputs matrices $\mathbf{A}, \mathbf{A}_1, \ldots, \mathbf{A}_\ell$ and a vector $\mathbf{v}$ and these form the MPK. The MSK is the trapdoor for $\mathbf{A}$.
- Enc computes

$$\mathbf{s}^T[\mathbf{A}||\mathbf{A}_1 - \mu_1\mathbf{G}|| \ldots ||\mathbf{A}_\ell - \mu_\ell\mathbf{G}]$$

  (plus error, of course, and we will consider that understood.) Finally, the message is encrypted as $\mathbf{s}^T\mathbf{v} + e + m\lfloor q/2 \rfloor$.

- Let's see how Dec might work. You (and in fact anyone) can compute

$$\mathbf{s}^T[\mathbf{A}||\mathbf{A}_1 - \mu_1\mathbf{G}|| \ldots ||\mathbf{A}_\ell - \mu_\ell\mathbf{G}] \begin{bmatrix} \mathbf{I} & \mathbf{0} \\ \mathbf{0} & \mathbf{H}_{f,\boldsymbol{\mu}} \end{bmatrix} = \mathbf{s}^T[\mathbf{A}||\mathbf{A}_f - f(\boldsymbol{\mu})\mathbf{G}]$$

  using the key equation.

  If you had a short $\mathbf{r}$ that maps $[\mathbf{A}||\mathbf{A}_f - \mathbf{G}]$ to $\mathbf{v}$, that is

$$[\mathbf{A}||\mathbf{A}_f - \mathbf{G}]\ \mathbf{r} = \mathbf{v}$$

  you can decrypt and find m. (Can you fill in the blanks?)

Two notes:

- The security definition mirrors IBE exactly, and the security proof of this scheme mirrors that of the ABB IBE scheme that we did in the last lecture. I will leave it to you as an exercise. The reference is the work of [BGG+14].

- One might wonder if the attributes $\boldsymbol{\mu}$ need to be revealed. The answer is "NO", in fact one can construct an attribute-hiding ABE scheme (also called a predicate encryption scheme). There are two flavors of security of such a scheme, the weaker one can be realized using LWE [GVW15] and the stronger one implies indistinguishability obfuscation, a *very* powerful cryptographic primitive which we don't know how to construct from LWE yet. More in the next lecture.

8.9 Constrained PRF

A constrained PRF is a special type of PRF where the owner of the PRF key K can construct a special key K_f which enables anyone to compute

$$\forall x \text{ s.t. } f(x) = 0 : \quad \mathsf{PRF}(K, x)$$

(here, arbitrarily and for convention, we will set 0 to mean `true`.) The PRF values at all other values should remain hidden given K_f, the *constrained key.*

We will only consider single-key CPRFs here, that is the adversary gets to see the constrained key for a single function f of her choice. Constructing many-key CPRFs from LWE is another one of my favorite open problems!

More generally, the adversary can get a single constrained key together with oracle access to the PRF (as usual). Her job is to compute $\mathsf{PRF}(K, x^*)$ for some x^* where (a) $f(x^*) = 1$ (false) and (b) she did not make an oracle query on x^*.

Here is a construction of a constrained PRF using the key equation. See [BV15] for details and extensions.

- The scheme has public parameters $\mathbf{B}_0, \mathbf{B}_1$ and $\mathbf{A}_1, \ldots, \mathbf{A}_k$ where k is an upper bound on the description length of any function f that will be constrained. The PRF key is $\mathbf{s}$.

- To define the PRF, consider the universal function $\mathcal{U}$:

  $$\forall f \text{ where} |f| \leq k, x \in \{0,1\}^\ell : \quad \mathcal{U}(f, x) = f(x)$$

  The key equation applied to the universal circuit $\mathcal{U}$ now tells us that

  $$\begin{aligned} &[\mathbf{A}_1 - f_1\mathbf{G} || \ldots || \mathbf{A}_k - f_k\mathbf{G} || \mathbf{B}_{x_1} - x_1\mathbf{G} || \ldots || \mathbf{B}_{x_\ell} - x_\ell\mathbf{G}] \; \mathbf{H}_{\mathcal{U},f,x} \\ &= \mathbf{A}_{\mathcal{U},x} \; - \; \mathcal{U}(f,x)\mathbf{G} \\ &= \mathbf{A}_{\mathcal{U},x} - f(x)\mathbf{G} \end{aligned}$$

  Here, $\mathbf{A}_{\mathcal{U},x}$ (which we will simply denote as $\mathbf{A}_x$) is a result of the GSW homomorphic evaluation on the matrices $\mathbf{A}_1, \ldots, \mathbf{A}_k, \mathbf{B}_{x_1}, \ldots, \mathbf{B}_{x_\ell}$.

- The PRF is defined to be $\lfloor \mathbf{s}^T \mathbf{A}_x \rceil$ on every input x.

  Note that the PRF has to be defined *independent of which function the key will later be constrained with.* Indeed, this definition of the PRF does not depend on f at all.

Here is a construction of a constrained PRF using the key equation. See [BV15] for details and extensions.

- The scheme has public parameters $\mathbf{B}_0, \mathbf{B}_1$ and $\mathbf{A}_1, \ldots, \mathbf{A}_k$ where k is an upper bound on the description length of any function f that will be constrained. The PRF key is $\mathbf{s}$.
- To define the PRF, consider the universal function $\mathcal{U}$:

  $$\forall f \text{ where} |f| \leq k, x \in \{0,1\}^\ell : \quad \mathcal{U}(f,x) = f(x)$$

  The key equation applied to the universal circuit $\mathcal{U}$ now tells us that

  $$\begin{aligned} & [\mathbf{A}_1 - f_1\mathbf{G}||\ldots||\mathbf{A}_k - f_k\mathbf{G}||\mathbf{B}_{x_1} - x_1\mathbf{G}||\ldots||\mathbf{B}_{x_\ell} - x_\ell\mathbf{G}] \ \mathbf{H}_{\mathcal{U},f,x} \\ & = \mathbf{A}_{\mathcal{U},x} \ - \ \mathcal{U}(f,x)\mathbf{G} \\ & = \mathbf{A}_{\mathcal{U},x} - f(x)\mathbf{G} \end{aligned}$$

  Here, $\mathbf{A}_{\mathcal{U},x}$ (which we will simply denote as $\mathbf{A}_x$) is a result of the GSW homomorphic evaluation on the matrices $\mathbf{A}_1, \ldots, \mathbf{A}_k, \mathbf{B}_{x_1}, \ldots, \mathbf{B}_{x_\ell}$.
- The PRF is defined to be $\lfloor \mathbf{s}^T\mathbf{A}_x \rceil$ on every input x.
- The constrained key for a function f is

  $$\mathbf{s}^T[\mathbf{A}_1 - f_1\mathbf{G}||\ldots||\mathbf{A}_k - f_k\mathbf{G}||\mathbf{B}_0||\mathbf{B}_1 - \mathbf{G}]$$

  On input x, the constrained eval proceeds as follows. First you can get

  $$[\mathbf{A}_1 - f_1\mathbf{G}||\ldots||\mathbf{A}_k - f_k\mathbf{G}||\mathbf{B}_{x_1} - x_1\mathbf{G}||\ldots||\mathbf{B}_{x_\ell} - x_\ell\mathbf{G}]$$

  for any x of your choice. Second, using the key equation, multiplying this on the right by $\mathbf{H}_{\mathcal{U},f,x}$:

  $$\mathbf{s}^T[\mathbf{A}_x - f(x)\mathbf{G}]$$

  from which one computes $\lfloor \mathbf{s}^T\mathbf{A}_x \rceil := \mathsf{PRF}(K,x)$ if $f(x) = 0$ (true).

Here is a construction of a constrained PRF using the key equation. See [BV15] for details and extensions.

- The scheme has public parameters $\mathbf{B}_0, \mathbf{B}_1$ and $\mathbf{A}_1, \ldots, \mathbf{A}_k$ where k is an upper bound on the description length of any function f that will be constrained. The PRF key is $\mathbf{s}$.
- To define the PRF, consider the universal function $\mathcal{U}$:

  $$\forall f \text{ where} |f| \leq k, x \in \{0,1\}^\ell : \quad \mathcal{U}(f,x) = f(x)$$

  The key equation applied to the universal circuit $\mathcal{U}$ now tells us that

  $$\begin{aligned} & [\mathbf{A}_1 - f_1\mathbf{G}||\ldots||\mathbf{A}_k - f_k\mathbf{G}||\mathbf{B}_{x_1} - x_1\mathbf{G}||\ldots||\mathbf{B}_{x_\ell} - x_\ell\mathbf{G}] \ \mathbf{H}_{\mathcal{U},f,x} \\ & = \mathbf{A}_{\mathcal{U},x} \ - \ \mathcal{U}(f,x)\mathbf{G} \\ & = \mathbf{A}_{\mathcal{U},x} - f(x)\mathbf{G} \end{aligned}$$

  Here, $\mathbf{A}_{\mathcal{U},x}$ (which we will simply denote as $\mathbf{A}_x$) is a result of the GSW homomorphic evaluation on the matrices $\mathbf{A}_1, \ldots, \mathbf{A}_k, \mathbf{B}_{x_1}, \ldots, \mathbf{B}_{x_\ell}$.

- The PRF is defined to be $\lfloor \mathbf{s}^T \mathbf{A}_x \rceil$ on every input x.
- The constrained key for a function f is

$$\mathbf{s}^T[\mathbf{A}_1 - f_1\mathbf{G}||\dots||\mathbf{A}_k - f_k\mathbf{G}||\mathbf{B}_0||\mathbf{B}_1 - \mathbf{G}]$$

  On input x, you can compute

$$\mathbf{s}^T[\mathbf{A}_x - f(x)\mathbf{G}]$$

- For security: suppose an adversary managed to compute

$$\mathbf{s}^T\mathbf{A}_x + \mathbf{e}$$

  for some x where $f(x) = 1$. We can ourselves compute

$$\mathbf{s}^T(\mathbf{A}_x - \mathbf{G}) + \mathbf{e}'$$

  using constrained evaluation. Put together, these reveal $\mathbf{s}^T\mathbf{G}$ plus error, and therefore $\mathbf{s}$, breaking LWE.

CHAPTER 9

Constrained PRFs and Program Obfuscation

9.1 Constrained PRF

A constrained PRF is a special type of PRF where the owner of the PRF key K can construct a special key K_f which enables anyone to compute

$$\forall x \text{ s.t. } f(x) = 1 : \quad \mathsf{PRF}(K, x)$$

The PRF values at all other values should remain hidden given K_f, the *constrained key*.

We will only consider single-key CPRFs here, that is the adversary gets to see the constrained key for a single function f of her choice. Constructing many-key CPRFs from LWE is another one of my favorite open problems!

More generally, the adversary can get a single constrained key together with oracle access to the PRF (as usual). Her job is to compute $\mathsf{PRF}(K, x^*)$ for some x^* where (a) $f(x^*) = 0$ (false) and (b) she did not make an oracle query on x^*.

9.2 Private Constrained PRFs

A (single key) private constrained PRF is a constrained PRF where, in addition, the constrained key hides the function that it constrains.

In other words, given a constrained key (denoted as $K\{f\}$) and oracle access to $\mathsf{PRF}_K(\cdot)$, it is computationally hard to determine what f is. There are of course settings where this is impossible to achieve. (Can you think of one?)

There are of course settings where this is impossible to achieve. For example, assume that an adversary gets $K\{f\}$ and wants to check if $f(x) = 0$ or 1 for some x in her mind. This can be done: she makes an oracle query to x and gets $\mathsf{PRF}_K(x)$. She also uses $K\{f\}$ to compute something, and she knows that the output matches $\mathsf{PRF}_K(x)$ iff $f(x) = 1$. This reveals some information, so it is not reasonable to expect to hide all information about f given $K\{f\}$ and oracle access to the PRF.

How then shall we define private constrained PRFs?

Security is defined in terms of an indistinguishability game where the adversary produces two functions f_0, f_1 and gets oracle access to PRF but only for those x for which $f_0(x) = f_1(x)$.

9.3 Private Constrained PRF: Construction

Dual-BLMR

We will start with the following PRF (that we will call dual-BLMR).

$$\mathsf{PRF}_{\{\mathbf{S}_{i,b}\}}(x_1 x_2 \dots x_\ell) = \lfloor \mathbf{S}_{1,x_1}\mathbf{S}_{2,x_2}\dots\mathbf{S}_{\ell,x_\ell}\mathbf{A}\rceil_p \pmod q$$

where $\mathbf{S}_{i,b}$ are (secret) square matrices with random small entries, and $\mathbf{A}$ is a (public) uniformly random matrix mod q.

(This is closely related to the BLMR construction which if you recall looks as follows:

$$\mathsf{PRF}_{\mathbf{S}}(x_1 x_2 \dots x_\ell) = \lfloor \mathbf{S}\mathbf{A}_{1,x_1}\mathbf{G}^{-}(\mathbf{A}_{2,x_2})\dots\mathbf{G}^{-}(\mathbf{A}_{\ell,x_\ell})\rceil_p \pmod q$$

where the $\mathbf{A}_{i,x_i}$ are random matrices. This is actually a slight generalization of BLMR using a secret *matrix* $\mathbf{S}$ and 2ℓ public matrices.)

We'd like to generate constrained keys for dual-BLMR.

Constrained Key for the Identity Function

For starters, let's try generating the constrained key for the identity function. That is, given the constrained key, one should be able to compute the PRF on *all* inputs. (We want the construction to be non-trivial in the sense that the constrained key should not reveal the PRF key).

Here is a try.

$$\begin{array}{llll} \mathbf{S}_{1,0}\mathbf{A}+\mathbf{E}_{1,0}, & \mathbf{S}_{2,0}\mathbf{A}+\mathbf{E}_{2,0}, & \dots, & \mathbf{S}_{\ell,0}\mathbf{A}+\mathbf{E}_{\ell,0} \\ \mathbf{S}_{1,1}\mathbf{A}+\mathbf{E}_{1,1}, & \mathbf{S}_{2,1}\mathbf{A}+\mathbf{E}_{2,1}, & \dots, & \mathbf{S}_{\ell,1}\mathbf{A}+\mathbf{E}_{\ell,1} \end{array}$$

This certainly hides the $\mathbf{S}_{i,b}$ (the PRF key) but it's not clear how to compute the PRF output from it.

When you are stuck, you start by naming things. So, let's do it.

$$\begin{array}{llll} \mathbf{B}_{1,0} := \mathbf{S}_{1,0}\mathbf{A}+\mathbf{E}_{1,0}, & \mathbf{B}_{2,0} := \mathbf{S}_{2,0}\mathbf{A}+\mathbf{E}_{2,0}, & \dots, & \mathbf{B}_{\ell,0} := \mathbf{S}_{\ell,0}\mathbf{A}+\mathbf{E}_{\ell,0} \\ \mathbf{B}_{1,1} := \mathbf{S}_{1,1}\mathbf{A}+\mathbf{E}_{1,1}, & \mathbf{B}_{2,1} := \mathbf{S}_{2,1}\mathbf{A}+\mathbf{E}_{2,1}, & \dots, & \mathbf{B}_{\ell,1} := \mathbf{S}_{\ell,1}\mathbf{A}+\mathbf{E}_{\ell,1} \end{array}$$

It's not clear what you get by multiplying, say, $\mathbf{B}_{1,0}$ with $\mathbf{B}_{2,0}$. What we need is to enable some sort of homomorphic multiplication. Let's take inspiration from GSW13.

$$\begin{array}{llll} \mathbf{B}_{1,0} := \mathbf{S}_{1,0}\mathbf{A}+\mathbf{E}_{1,0}, & \mathbf{B}_{2,0} := \mathbf{S}_{2,0}\mathbf{A}+\mathbf{E}_{2,0}, & \dots, & \mathbf{B}_{\ell,0} := \mathbf{S}_{\ell,0}\mathbf{A}+\mathbf{E}_{\ell,0} \\ \mathbf{B}_{1,1} := \mathbf{S}_{1,1}\mathbf{A}+\mathbf{E}_{1,1}, & \mathbf{B}_{2,1} := \mathbf{S}_{2,1}\mathbf{A}+\mathbf{E}_{2,1}, & \dots, & \mathbf{B}_{\ell,1} := \mathbf{S}_{\ell,1}\mathbf{A}+\mathbf{E}_{\ell,1} \end{array}$$

Here is how GSW enables multiplication.

$$\begin{aligned}
\mathbf{B}_{1,0} \cdot \mathbf{A}^{-1}(\mathbf{B}_{2,0}) &= \mathbf{S}_{1,0}\mathbf{A} + \mathbf{E}_{1,0} \cdot \mathbf{A}^{-1}(\mathbf{S}_{2,0}\mathbf{A} + \mathbf{E}_{2,0}) \\
&= \mathbf{S}_{1,0}\mathbf{A} \cdot \mathbf{A}^{-1}(\mathbf{S}_{2,0}\mathbf{A} + \mathbf{E}_{2,0}) + \underbrace{\mathbf{E}_{1,0}\mathbf{A}^{-1}(\mathbf{B}_{2,0})}_{\approx 0} \\
&\approx \mathbf{S}_{1,0} \cdot (\mathbf{S}_{2,0}\mathbf{A} + \mathbf{E}_{2,0}) \\
&= \mathbf{S}_{1,0}\mathbf{S}_{2,0}\mathbf{A} + \underbrace{\mathbf{S}_{1,0}\mathbf{E}_{2,0}}_{\approx 0} \\
&\approx \mathbf{S}_{1,0}\mathbf{S}_{2,0}\mathbf{A}
\end{aligned}$$

where $\mathbf{A}^{-1}(\mathbf{B})$ is a matrix $\mathbf{R}$ with small random entries such that $\mathbf{AR} = \mathbf{B} \pmod q$.

Continuing along these lines, you can compute the PRF on all inputs if you can compute $\mathbf{A}^{-1}(\mathbf{B}_{i,b})$. Since computing $\mathbf{A}^{-1}(\cdot)$ requires the trapdoor of $\mathbf{A}$, the owner of the PRF key can precompute all these matrices

$$\mathbf{D}_{i,b} := \mathbf{A}^{-1}(\mathbf{S}_{i,b}\mathbf{A} + \mathbf{E}_{i,b})$$

and release them.

$$\begin{array}{llll}
\mathbf{D}_{1,0} := \mathbf{A}^{-1}(\mathbf{S}_{1,0}\mathbf{A} + \mathbf{E}_{1,0}), & \mathbf{D}_{2,0} := \mathbf{A}^{-1}(\mathbf{S}_{2,0}\mathbf{A} + \mathbf{E}_{2,0}), & \ldots, & \mathbf{D}_{\ell,0} := \mathbf{A}^{-1}(\mathbf{S}_{\ell,0}\mathbf{A} + \mathbf{E}_{\ell,0}) \\
\mathbf{D}_{1,1} := \mathbf{A}^{-1}(\mathbf{S}_{1,1}\mathbf{A} + \mathbf{E}_{1,1}), & \mathbf{D}_{2,1} := \mathbf{A}^{-1}(\mathbf{S}_{2,1}\mathbf{A} + \mathbf{E}_{2,1}), & \ldots, & \mathbf{D}_{\ell,1} := \mathbf{A}^{-1}(\mathbf{S}_{\ell,1}\mathbf{A} + \mathbf{E}_{\ell,1})
\end{array}$$

How about security? This is tricky because we would like to argue that each $\mathbf{S}_{i,b}\mathbf{A} + \mathbf{E}_{i,b}$ is pseudorandom, invoking LWE. But the constrained key depends on the trapdoor for $\mathbf{A}$ in the presence of which LWE is *not* hard.

So, let's modify the construction a bit more.

$$\mathbf{A}_0, \begin{array}{llll} \mathbf{D}_{1,0} := \mathbf{A}_0^{-1}(\mathbf{S}_{1,0}\mathbf{A}_1 + \mathbf{E}_{1,0}), & \mathbf{D}_{2,0} := \mathbf{A}_1^{-1}(\mathbf{S}_{2,0}\mathbf{A}_2 + \mathbf{E}_{2,0}), & \dots, & \mathbf{D}_{\ell,0} := \mathbf{A}_{\ell-1}^{-1}(\mathbf{S}_{\ell,0}\mathbf{A}_\ell + \mathbf{E}_{\ell,0}) \\ \mathbf{D}_{1,1} := \mathbf{A}_0^{-1}(\mathbf{S}_{1,1}\mathbf{A}_1 + \mathbf{E}_{1,1}), & \mathbf{D}_{2,1} := \mathbf{A}_1^{-1}(\mathbf{S}_{2,1}\mathbf{A}_2 + \mathbf{E}_{2,1}), & \dots, & \mathbf{D}_{\ell,1} := \mathbf{A}_{\ell-1}^{-1}(\mathbf{S}_{\ell,1}\mathbf{A}_\ell + \mathbf{E}_{\ell,1}) \end{array}$$

where A_i are *distinct* random matrices (chosen during the generation of the constrained key), and we think of $\mathbf{A}_\ell = \mathbf{A}$.

We have that

$$\mathbf{A}_0\mathbf{D}_{1,x_1}\mathbf{D}_{2,x_2}\dots\mathbf{D}_{\ell,x_\ell} \approx \mathbf{S}_{1,x_1}\mathbf{S}_{2,x_2}\dots\mathbf{S}_{\ell,x_\ell}\mathbf{A} \pmod q$$

Thus, we have a construction of a PRF together with a constrained key (and an algorithm to generate it) that can evaluate the PRF at all inputs while (plausibly) hiding the original PRF key.

We now have two questions: (a) allow more expressive constraint functions and (b) show security! We will do these in turn.

Interlude: Matrix Branching Programs

This is a convenient model of computation for us as we are already working with matrices! In a matrix branching program computing a Boolean function f on k input bits, we have 2ℓ matrices $\mathbf{M}_{i,b} \in \{0,1\}^{w\times w}$ (where think of w as a constant) and a vector $\mathbf{v} \in \{0,1\}^{1\times w}$ where $\ell \geq k$:

$$\mathbf{v} = (1\ 0\ \dots 0), \begin{array}{cccc} \mathbf{M}_{1,0} & \mathbf{M}_{2,0} & \dots & \mathbf{M}_{\ell,0} \\ \mathbf{M}_{1,1} & \mathbf{M}_{2,1} & \dots & \mathbf{M}_{\ell,1} \end{array}$$

In general, each location i (corresponding to a pair of **full-rank** matrices $\mathbf{M}_{i,0}$ and $\mathbf{M}_{i,1}$) is indexed by an input bit, say $x_j = x_{j(i)}$. To avoid complicating matters, we will let $\ell = k$ and let $j(i) = i$.

To compute the program on an input x, you compute

$$\mathbf{u} := \mathbf{v}\mathbf{M}_{1,x_1}\mathbf{M}_{2,x_2}\dots\mathbf{M}_{\ell,x_\ell}$$

$\mathbf{u}$ is either $(1\ 0\ \dots 0)$ or $(0\ 1\ \dots 0)$. If $\mathbf{u}[1] \neq 0$, output 1 (**true**), otherwise output 0 (**false**).

We know that every function in NC1 (circuits with log depth and poly size) can be computed by poly-size matrix branching programs where each matrix is 5-by-5 permutation matrix (it's in S_5). This is Barrington's theorem. So, matrix branching programs are a pretty powerful computational model.

(Give a simple example of a matrix branching program.)

Constructing a Constrained Key

Let's now incorporate matrix BPs into the constrained key. The construction is due to Canetti and Chen (2017), improved later by Chen-Vaikuntanathan-Wee'18.

Say you want to constrain the PRF key for a constraint function f. Create a length-ℓ matrix branching program for f first.

The constrained key is

$$\hat{\mathbf{v}}\mathbf{A}_0, \begin{array}{llll} \mathbf{D}_{1,0} := \mathbf{A}_0^{-1}(\hat{\mathbf{S}}_{1,0}\mathbf{A}_1 + \mathbf{E}_{1,0}), & \mathbf{D}_{2,0} := \mathbf{A}_1^{-1}(\hat{\mathbf{S}}_{2,0}\mathbf{A}_2 + \mathbf{E}_{2,0}), & \dots, & \mathbf{D}_{\ell,0} := \mathbf{A}_{\ell-1}^{-1}(\hat{\mathbf{S}}_{\ell,0}\mathbf{A}_\ell + \mathbf{E}_{\ell,0}) \\ \mathbf{D}_{1,1} := \mathbf{A}_0^{-1}(\hat{\mathbf{S}}_{1,1}\mathbf{A}_1 + \mathbf{E}_{1,1}), & \mathbf{D}_{2,1} := \mathbf{A}_1^{-1}(\hat{\mathbf{S}}_{2,1}\mathbf{A}_2 + \mathbf{E}_{2,1}), & \dots, & \mathbf{D}_{\ell,1} := \mathbf{A}_{\ell-1}^{-1}(\hat{\mathbf{S}}_{\ell,1}\mathbf{A}_\ell + \mathbf{E}_{\ell,1}) \end{array}$$

where $\hat{\mathbf{S}}_{i,b} = \mathbf{M}_{i,b} \otimes \mathbf{S}_{i,b}$ is a tensor product of the two matrices and $\hat{\mathbf{v}} = \mathbf{v} \otimes \mathbf{I}$.

The key property of tensor products is the following associative property.

$$(\mathbf{A} \otimes \mathbf{B}) \cdot (\mathbf{C} \otimes \mathbf{D}) = \mathbf{AC} \otimes \mathbf{BD}$$

where $\otimes$ is the tensor product and $\cdot$ is matrix multiplication.

(Note that the $\hat{\mathbf{S}}_{i,b}$ are now $nw \times nw$ matrices and $\mathbf{A}_i$ have to be corresponding larger, i.e., $nw \times m$ for a large enough m.)

We have that

$$\hat{\mathbf{v}}\mathbf{A}_0\mathbf{D}_{1,x_1}\mathbf{D}_{2,x_2}\dots\mathbf{D}_{\ell,x_\ell} \approx \hat{\mathbf{v}}\hat{\mathbf{S}}_{1,x_1}\hat{\mathbf{S}}_{2,x_2}\dots\hat{\mathbf{S}}_{\ell,x_\ell}\mathbf{A}_\ell \pmod q = \Big((\mathbf{v}\prod \mathbf{M}_{i,x_i}) \otimes (\prod \mathbf{S}_{i,x_i})\Big)\mathbf{A}_\ell$$

We know that $\mathbf{v}\prod \mathbf{M}_{i,x_i}$ is either $(1\ 0\ 0\ \dots 0)$ or $(0\ 1\ 0\ \dots 0)$. So the entire product is close to $\prod \mathbf{S}_{i,x_i}\mathbf{A}_\ell^{(1)}$ if $f(x) = 1$ or $\prod \mathbf{S}_{i,x_i}\mathbf{A}_\ell^{(2)}$ if $f(x) = 0$. where

$$\mathbf{A}_\ell := \begin{bmatrix} \mathbf{A}_\ell^{(1)} \\ \mathbf{A}_\ell^{(2)} \\ \dots \\ \mathbf{A}_\ell^{(w)} \end{bmatrix}$$

So, letting $\mathbf{A}_\ell^{(1)} := \mathbf{A}$ in the definition of the PRF finishes the construction.

We will give this monster a compact name:

$$\hat{\mathbf{v}}\mathbf{A}_0, \begin{array}{llll} \mathbf{D}_{1,0} := \mathbf{A}_0^{-1}(\hat{\mathbf{S}}_{1,0}\mathbf{A}_1 + \mathbf{E}_{1,0}), & \mathbf{D}_{2,0} := \mathbf{A}_1^{-1}(\hat{\mathbf{S}}_{2,0}\mathbf{A}_2 + \mathbf{E}_{2,0}), & \dots, & \mathbf{D}_{\ell,0} := \mathbf{A}_{\ell-1}^{-1}(\hat{\mathbf{S}}_{\ell,0}\mathbf{A}_\ell + \mathbf{E}_{\ell,0}) \\ \mathbf{D}_{1,1} := \mathbf{A}_0^{-1}(\hat{\mathbf{S}}_{1,1}\mathbf{A}_1 + \mathbf{E}_{1,1}), & \mathbf{D}_{2,1} := \mathbf{A}_1^{-1}(\hat{\mathbf{S}}_{2,1}\mathbf{A}_2 + \mathbf{E}_{2,1}), & \dots, & \mathbf{D}_{\ell,1} := \mathbf{A}_{\ell-1}^{-1}(\hat{\mathbf{S}}_{\ell,1}\mathbf{A}_\ell + \mathbf{E}_{\ell,1}) \end{array}$$

Call this a **GGH15 chain** (after the inventors Gentry, Gorbunov and Halevi) for the program f, secrets $\mathbf{S}_{i,b}$ and the final matrices $\mathbf{A}_\ell^{(1)} := \mathbf{A}^{(1)}$ and $\mathbf{A}_\ell^{(2)} := \mathbf{A}^{(2)}$.

On input x, we can compute

$$\approx \mathbf{S}_x\mathbf{A}^{(2-f(x))}$$

where now and henceforth $\mathbf{S}_x := \prod \mathbf{S}_{i,x_i}$.

Proof of Constraint-Hiding

We will now sketch the proof that the scheme is constraint-hiding. We start by showing that the constrained key is pseudorandom.

$$\hat{\mathbf{v}}\mathbf{A}_0, \begin{array}{llll} \mathbf{D}_{1,0} := \mathbf{A}_0^{-1}(\hat{\mathbf{S}}_{1,0}\mathbf{A}_1 + \mathbf{E}_{1,0}), & \mathbf{D}_{2,0} := \mathbf{A}_1^{-1}(\hat{\mathbf{S}}_{2,0}\mathbf{A}_2 + \mathbf{E}_{2,0}), & \dots, & \mathbf{D}_{\ell,0} := \mathbf{A}_{\ell-1}^{-1}(\hat{\mathbf{S}}_{\ell,0}\mathbf{A}_\ell + \mathbf{E}_{\ell,0}) \\ \mathbf{D}_{1,1} := \mathbf{A}_0^{-1}(\hat{\mathbf{S}}_{1,1}\mathbf{A}_1 + \mathbf{E}_{1,1}), & \mathbf{D}_{2,1} := \mathbf{A}_1^{-1}(\hat{\mathbf{S}}_{2,1}\mathbf{A}_2 + \mathbf{E}_{2,1}), & \dots, & \mathbf{D}_{\ell,1} := \mathbf{A}_{\ell-1}^{-1}(\hat{\mathbf{S}}_{\ell,1}\mathbf{A}_\ell + \mathbf{E}_{\ell,1}) \end{array}$$

1. Observe first that

$$\hat{\mathbf{S}}\mathbf{A} + \mathbf{E}$$

is pseudorandom by LWE where $\hat{\mathbf{S}} = \mathbf{M} \otimes \mathbf{S}$ where $\mathbf{S}$ is a random small matrix and $\mathbf{M}$ is any **full-rank** matrix. (Can you see what happens if $\mathbf{M}$ is not full-rank?)

2. The trapdoor issue still rears its head, that is, the constrained key is a function of trapdoors for various matrices $\mathbf{A}$, in the presence of which LWE for those matrices does not hold!

   However, and this is the thing that saves us, observe that the trapdoor for $\mathbf{A}_\ell$ is **never used**!!

3. So, we can do a *right-to-left* proof where we replace matrices by random small matrices in two mini-steps:

   - First replace $\hat{\mathbf{S}}_{\ell,b}\mathbf{A}_\ell + \mathbf{E}_{\ell,b}$ by uniformly random and independent matrices $\mathbf{U}_{\ell,b}$. This is by an invocation of LWE.
   - Second, replace $\mathbf{A}_{\ell-1}^{-1}(\mathbf{U}_{\ell,b})$ by a Gaussian matrix $\mathbf{D}_{\ell,b}$. This is by an invocation of the GPV theorem (the same thing we used to construct digital signatures via Gaussian sampling.)
   - Now, we are at a hybrid experiment where we *never use the trapdoor for* $\mathbf{A}_{\ell-1}$! Rinse and repeat.

Of course, we need to prove more. That it is constraint-hiding even with oracle access to the PRF (in the sense that we defined) and that the constraint key does not enable evaluation of PRF on x such that $f(x) = 0$. We will leave these as a (non-trivial but doable) exercise.

An advanced comment. Before we move on, let's ask if we can use this to release more than one constrained key. Ie, can we plausibly conjecture security?

9.4 Program Obfuscation and Other Beasts

How much more useful is it to have the *code* of a program than merely *oracle* access (or input/output access) to it? This is a foundational question in cryptography, and indeed in theoretical computer science and even all of computing.

In a cryptographic context, we ask: can we transform a program into another (*obfuscated*) program which has the same input/output functionality but is no more revealing than having black-box access? This is the problem of *program obfuscation.*

On the one hand, given a program P, it is hard to even say whether it halts on input 0^n (this is the Halting problem). Indeed, any "non-trivial" property of a program is undecidable (this is Rice's theorem). So, worst-case programs seem naturally obfuscated. Yet, these are programs we do not necessarily care about. This brings into sharper focus the problem of program obfuscation, the problem of transforming *arbitrary* programs, ones that we do care about, into their obfuscated versions.

Aside from their obvious uses in software protection, program obfuscation is of fundamental importance to cryptography. Let us illustrate two of their uses.

Applications of Program Obfuscation (Informally)

In the early 1970s, the big problem in cryptography was whether there is a method of encrypting messages from A to B which does not require A and B to have met beforehand and come up with a common private key. In other words, is public-key cryptography possible?

In a landmark paper that kickstarted the field, Diffie and Hellman wondered about the following possibility. Take the encryption program of a secret-key encryption scheme and obfuscate it!

> **Essentially what is required is a one-way compiler: one which takes an easily understood program written in a high level language and translates it into an incomprehensible program in some machine language. The compiler is one-**

They didn't quite manage to make it work, and went a different route, but it's a fascinating route.

Indeed, being able to obfuscate programs (in an appropriate sense) makes nearly every cryptographic task trivial. Can you see how to achieve fully homomorphic encryption if I gave you a way to obfuscate programs?

Defining Program Obfuscation

What does it mathematically mean to obfuscate programs?

One way to define it leads to the notion of *virtual black-box obfuscation* due to Hada and Barak et al. In a nutshell, it defines an obfuscator $\mathcal{O}$ to be a probabilistic algorithm that takes programs (or circuits or Turing machines...) and converts them into other programs that are:

- *functionally equivalent* and *nearly as fast* (asymptotically!); and
- *virtual black-box.* That is, for every PPT adversary A that tries to learn a predicate of the original program given its obfuscation, there is a black-box PPT adversary S (also called the simulator) that does the same thing given oracle access. That is,

$$\forall A, \exists S, \forall \pi : \{0,1\}^* \to \{0,1\} \text{ and } \forall \text{ programs } P :$$
$$\Pr[A(\mathcal{O}(P) = \pi(P)] \approx \Pr[S^P(1^n) = \pi(P)]$$

This is a pretty strong definition and formalizes the idea that the obfuscated program should be *no more revealing* than black-box access to it. Unfortunately, it is also impossible to construct a universal program obfuscator. (can you see, perhaps informally, why?)

Defining Program Obfuscation: Take 2

Nevertheless, researchers have shown multiple paths to circumvent this (one!) impossibility. The most well-studied is to relax the *definition* to indistinguishability obfuscation. We will explore a different route today, relaxing the *class of functions* we plan to obfuscate. In particular, we will look at the following class of functions.

$$F_{f,\alpha,\beta}(x) = \begin{cases} \beta & \text{if } f(x) = \alpha \\ 0 & \text{otherwise} \end{cases}$$

where f is some function and α, β are strings (of length at least the security parameter).

We call this *lockable obfuscation*, a terminology due to Wichs-Zirdelis'17 and Goyal-Koppula-Waters'17. (A special case of this is obfuscating point functions).

(A slightly weaker version we will look at is

$$F_{f,\alpha}(x)$$

which outputs 1 if $f(x) = \alpha$ and 0 otherwise.)

We will obfuscate this class when f is pretty complex (all we need is that there is a matrix branching program that computes it) and α is a uniformly random string (really, all we need is that it has large min-entropy.) In fact, in this world, we require that the lockable obfuscation for any function f, random α and arbitrary β is *pseudorandom* or *simulatable with no other information.*

9.5 Lockable Obfuscation: An Application

As we mentioned in the last class, it is easy to construct an example of an encryption scheme which is CPA-secure but releasing an encryption of the secret key under the matching public key is completely insecure (let's try!)

However, what if the encryption algorithm is restricted to encrypting bits? That is, when we say "encrypt the secret key", we will encrypt the bits of the secret key one by one. The counterexample we just constructed falls apart. So maybe every *bit encryption scheme* is circular-secure?! For a while, we did not have counterexamples for this under plausible conjectures, but now we do, thanks to lockable obfuscation.

Take any CPA-secure encryption scheme $(\mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec})$. Modify it to $(\mathsf{KeyGen}', \mathsf{Enc}', \mathsf{Dec}')$ that works as follows.

- KeyGen' generates a (pk, sk) pair from KeyGen and lets

  $$pk' = (pk, \mathsf{LO}(F_{f_{sk},\alpha,sk})) \text{ and } sk' = (sk, \alpha)$$

  for a random α. Here, f_{sk} takes a bunch of ciphertexts and decrypts them using sk. That is, if you feed an encryption of α to the lockable obfuscation, it reveals sk. You see where this is going!

- Enc' is the same as Enc.

- Dec' is the same as Dec.

This scheme remains CPA-secure (using the security of lockable obfuscation) but it is not circular-secure.

9.6 Lockable Obfuscation: Construction

The final item for the day is a construction of lockable obfuscation using the machinery of GGH15 chains. (We will show the slightly weaker construction today, but it's easy to modify it to get the stronger version.) To do lockable obfuscation $F_{f,\alpha}$ of a function f with lock $\alpha \in \{0,1\}^{2\lambda}$, do:

- generate 2λ matrices $\mathbf{A}^{(j,b)}$ with $j \in [\lambda]$ and $b \in \{0,1\}$ such that

  $$\sum_j \mathbf{A}^{(j,\alpha_j)} = 0 \pmod{q}$$

- generate λ GGH15 chains, one for the function f_j that outputs the j-th bit of f, and the final matrix pair $(\mathbf{A}^{(j,0)}, \mathbf{A}^{(j,1)})$. All GGH15 chains *share the same* $\mathbf{S}$ *matrices.*

For correctness, note that we can compute

$$\approx \mathbf{S}_x \mathbf{A}^{(j,f_j(x))}$$

for all j. Sum them up to get

$$\approx \mathbf{S}_x \sum_j \mathbf{A}^{(j,f_j(x))}$$

This sum is ≈ 0 if each $f_j(x) = \alpha_j$, in other words if $f(x) = \alpha$.

We will only say two words about security, which we argue in two steps.

1. First, the fact that α is random (or has min-entropy) can be used to show using Leftover hash lemma that the matrices $\mathbf{A}^{(j,b)}$ are truly random (as if there were no additive constraint on them.)

2. Now, we have ℓ GGH15 chains with the same $\mathbf{S}$ matrices but random and independent $\mathbf{A}$ matrices. The same proof that we did before can be argued to show security in this case as well.

CHAPTER 10

Ideal Lattices, Ring-SIS and Ring-LWE

This chapter is adapted from notes by Noah Stephens-Davidowitz.

10.1 Hash Functions

The SIS problem yields a very simple collision-resistant hash function that is provably secure if worst-case lattice problems are hard:

$$h_{\mathbf{A}}(\mathbf{e}) = \mathbf{A}\mathbf{e} \pmod q$$

where the key $\mathbf{A} \in \mathbb{Z}_q^{n\times m}$ is uniformly random and the input is $\mathbf{e} \in \{0,1\}^m$. Recall that finding an $h_{\mathbf{A}}$ collision is equivalent to solving the SIS problem, whose definition we reproduce below.

Definition 27. *For parameters n, m, q, the (average-case, homogenous) Short Integer Solutions (SIS) problem is defined as follows. The input is a uniformly random matrix $\mathbf{A} \in \mathbb{Z}_q^{n\times m}$. The goal is to find a non-zero vector $\mathbf{e} \in \{-1,0,1\}^m$ such that $\mathbf{A}\mathbf{e} = \mathbf{0} \pmod q$.*

$h_{\mathbf{A}}$ has a lot going for it as a hash function. It is remarkably simple—a linear collision-resistant hash function! And, we saw that it is provably secure under the assumption that certain well-studied worst-case lattice problems are hard. If those two things are not enough, $h_{\mathbf{A}}$ is also worthy of study because of its close relationship with LWE, the topic of this course and an extremely important problem for cryptographers.

Unfortunately, $h_{\mathbf{A}}$ is quite inefficient, since just reading the public hash description $\mathbf{A}$ takes time roughly $nm \log q > n^2$ (where the inequality follows from the fact that we must have $m > n$ in order for $h_{\mathbf{A}}$ to be a compressing function). But, $h_{\mathbf{A}}$ is breakable in time $2^{O(m)}$ (even by brute-force search).

*Ideal*ly, we would hope for a hash function that can be broken in time $2^{O(m)}$ to run in time roughly linear in $m \approx n$. Our goal is therefore to show a variant of $h_{\mathbf{A}}$ whose running time is in fact roughly linear in n.

10.2 The cyclic shift matrix, and the ring $\mathbb{Z}[x]/(x^n - 1)$

Since just reading the key of $h_{\mathbf{A}}$ requires time greater than n^2, any attempt to speed up the computation of $h_{\mathbf{A}}$ will presumably have to first compress the key size. E.g., we could take some short uniformly random seed r (with bit length, say, $O(n)$) and set $\mathbf{A} = H(r)$ for some suitable expanding function H. If H is modeled as a random oracle, then the resulting hash function $h_{H(r)}$ retains its security. (This idea is actually quite useful in practice [**?**] in the context of LWE.) However, if H is an arbitrary function, then we do not expect to be able to compute $h_{H(r)}(\boldsymbol{e})$ in time faster than n^2. So, though this idea immediately yields a hash function with a smaller key, we need to do more work to get a faster hash function.

Even so, assuming that we are happy with a small key and quadratic runtime, I do not know how to prove the security of this approach from standard assumptions, e.g., SIS.

> **Open Problem.** Construct a hash function with a linear-size key which is as secure as SIS or LWE (in particular, without relying on the random oracle assumption.)

In order to speed up our computation, we presumably need our matrix $\mathbf{A}$ to be a very special function of the seed. To that end, we take our short random seed to be $\ell = m/n$ uniformly random vectors $\boldsymbol{a}_1, \ldots, \boldsymbol{a}_\ell \in [q]^n$, and we take the columns of our matrix A to be the vectors $\boldsymbol{a}_1, \ldots, \boldsymbol{a}_\ell$ together with all "cyclic rotations" of the $\boldsymbol{a}_i$. I.e., for $\boldsymbol{a} = (a_1, \ldots, a_n)^T \in \mathbb{Z}^n$, we define

$$\mathrm{Rot}(\boldsymbol{a}) := \begin{pmatrix} a_1 & a_n & \cdots & a_3 & a_2 \\ a_2 & a_1 & \cdots & a_4 & a_3 \\ a_3 & a_2 & \cdots & a_5 & a_4 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ a_{n-2} & a_{n-3} & \cdots & a_n & a_{n-1} \\ a_{n-1} & a_{n-2} & \cdots & a_1 & a_n \\ a_n & a_{n-1} & \cdots & a_2 & a_1 \end{pmatrix} \in \mathbb{Z}^{n \times n} \ ,$$

where each column is a simple cyclic permutation of the previous column.[1] Matrices of the form $\mathrm{Rot}(\boldsymbol{a})$ are sometimes referred to as "cyclic matrices" or "circulant matrices." We then take

$$A = (\mathrm{Rot}(\boldsymbol{a}_1), \mathrm{Rot}(\boldsymbol{a}_2), \ldots, \mathrm{Rot}(\boldsymbol{a}_\ell)) \in \mathbb{Z}^{n \times m} \ .$$

We claim that for A with this structure, we can compute $A\boldsymbol{e} \bmod q$ in time $n\ell \cdot \mathrm{polylog}(n, q) = \widetilde{O}(m)$. This is because the set of all integer cyclic matrices, $\widetilde{R} := \{\mathrm{Rot}(\boldsymbol{a}) \ : \boldsymbol{a} \in \mathbb{Z}^n\}$ is actually a very nice set with nice algebraic structure. In particular, we can write

$$\mathrm{Rot}(\boldsymbol{a}) = (\boldsymbol{a}, X\boldsymbol{a}, \ldots, X^{n-1}\boldsymbol{a}) \ ,$$

where

$$X := \begin{pmatrix} 0 & 0 & \cdots & 0 & 1 \\ 1 & 0 & \cdots & 0 & 0 \\ 0 & 1 & \cdots & 0 & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & \cdots & 1 & 0 \end{pmatrix} \in \{0,1\}^{n \times n}$$

[1] Notice that the definition of Rot does not depend at all on q. It is convenient to forget about q for now and to think of $\boldsymbol{a}$ as some arbitrary vector in $\mathbb{Z}^n$.

is the "cyclic shift" matrix. Notice that $\widetilde{R} \subset \mathbb{Z}^{n\times n}$ is a lattice in $n \times n$ dimensions with rank n and basis $I_n, X, X^2, \ldots, X^{n-1}$. Indeed, for any $\boldsymbol{a} = (a_1, \ldots, a_n)^T \in \mathbb{Z}^n$, we can write

$$\mathrm{Rot}(\boldsymbol{a}) = a_1 I_n + a_2 X + \cdots + a_n X^{n-1} .$$

This identity immediately shows us that $\widetilde{R}$ is actually closed under (matrix) multiplication (note that $X^n = I_n$) and that multiplication is commutative over $\widetilde{R}$. I.e., $\widetilde{R}$ is a commutative ring!

In fact, $\widetilde{R}$ is isomorphic to the polynomial ring $R := \mathbb{Z}[x]/(x^n - 1)$. I.e., R is the ring of polynomials in the variable x of degree at most $n-1$ and integral coefficients, with addition defined in the obvious way and multiplication defined by the distributive law together with the identity

$$x \cdot x^i = \begin{cases} x^{i+1} & i < n-1 \\ 1 & i = n-1 \end{cases} .$$

(The polynomial $x^n - 1$ is the characteristic polynomial of the cyclic shift matrix X, which is why it arises in this context.) To see that these two rings are isomorphic, one only needs to check that the map $X \mapsto x$ is a bijection that preserves addition and multiplication of basis elements.

So, there's no reason to drag these $n\times n$ matrices around, and we can instead think of $\mathrm{Rot}(\boldsymbol{a}) \in \widetilde{R}$ as the corresponding polynomial $a \in R$ of degree at most $n-1$. (I.e., we change notation slightly.) We can therefore identify our matrix $A \in [q]^{n\times m}$ with a tuple of ring elements $(a_1, \ldots, a_\ell)^T \in R_{[q]}^\ell$, and similarly the input $\boldsymbol{e} \in \{0,1\}^m$ is a tuple of ring elements $(e_1, \ldots, e_\ell)^T \in R_{\{0,1\}}^\ell$, where we use the notation R_S to represent the set of polynomials in R with coefficients in S. Therefore, our hash function is now $h_{a_1,\ldots,a_\ell}(e_1, \ldots, e_\ell) = a_1 e_1 + \cdots + a_\ell e_\ell \bmod qR$.[2] For convenience, we abbreviate this by $h_a(e)$.

Now, to gain in efficiency, we simply recall that we can multiply two elements in $R_{[q]}$ in time $n \cdot \mathrm{polylog}(n, q)$ via the fast Fourier transform. Therefore, we can compute h_a in time $\ell n \cdot \mathrm{polylog}(n, q) = m \cdot \mathrm{polylog}(n, q)$, which is a tremendous speedup over the $nm \cdot \mathrm{polylog}(q)$ running time of the original h_A. Indeed, we typically think of $q = \mathrm{poly}(n)$ and $\ell = \mathrm{polylog}(n)$, so that this running time is quasilinear in n.

Towards Ring-SIS

Of course, this is not very useful if h_a is not secure. In fact, Micciancio showed that h_a is secure *as a one-way function* (under a plausible worst-case lattice assumption) [**?**]. I.e., with certain reasonable parameters, it is difficult to invert h_a on a random input. This result is really quite remarkable, but we will not state it formally.

Unfortunately, h_a is *not* a collision-resistant hash function. To see this, it helps to define the Ring-SIS problem, which is the analogue of SIS in this setting.

Definition 28. *For a ring R, integer modulus $q \geq 2$, and integer $\ell \geq 1$, the (average-case) Ring-SIS problem is defined as follows. The input is $a_1, \ldots, a_\ell \in R_{[q]}$ sampled independently and uniformly at random. The goal is to output $e_1, \ldots, e_\ell \in R_{\{-1,0,1\}}$ not all zero such that $a_1 e_1 + \cdots + a_\ell e_\ell = 0 \bmod qR$.*

[2] Here, we have chosen to think of the e_i as ring elements as well. This is formally justified by the identity

$$\mathrm{Rot}(\boldsymbol{a}) \cdot \mathrm{Rot}(\boldsymbol{e}) = \mathrm{Rot}\big(\mathrm{Rot}(\boldsymbol{a}) \cdot \boldsymbol{e}\big) .$$

The reduction mod qR simply means that we reduce the coefficients of the result of our polynomial multiplication modulo q.

One can easily see that finding a collision in h_a is equivalent to solving Ring-SIS, just like finding a collision in h_A is equivalent to solving SIS.

Unfortunately, Ring-SIS over $\mathbb{Z}[x]/(x^n - 1)$ is not hard. The issue is that the ring $\mathbb{Z}[x]/(x^n - 1)$ has non-trivial zero divisors (i.e., it is not an integral domain). To see this, let $\widetilde{e} = 1 + x + x^2 + \cdots + x^{n-1} \in \mathbb{Z}[x]/(x^n - 1)$, and notice that $(x - 1)\widetilde{e} = x^n - 1 = 0$. (In terms of Rot and $\widetilde{R}$, this corresponds to the fact that $\mathrm{Rot}(\boldsymbol{u})$ is singular, where $\boldsymbol{u} = (1, 1, \ldots, 1)^T \neq \mathbf{0}$.) This leads to an attack.

Claim 29. *For any integer modulus $q \geq 2$ and integer $n \geq 1$, let $R := \mathbb{Z}[x]/(x^n - 1)$ and let $\widetilde{e} = 1 + x + x^2 + \cdots + x^{n-1} \in R_{-1,0,1}$. Then, $a\widetilde{e} = 0 \bmod qR$ with probability $1/q$ when $a \in R_{[q]}$ is sampled uniformly at random.*

In particular, $\widetilde{e}, 0, \ldots, 0 \in R_{\{-1,0,1\}}$ is a solution to Ring-SIS over R with probability $1/q$, and the hash function h_a can be broken efficiently with probability $1/q$.

Proof. Suppose that $a \in R_{[q]}$ is divisible by $x - 1$ modulo qR. I.e., $a = (x - 1)a' \bmod qR$. Then, $\widetilde{e}a = \widetilde{e}(x-1)a' = 0 \bmod qR$. The result follows by noting that $a \in R_{[q]}$ is divisible by $x-1$ modulo qR with probability $1/q$. (Notice that being divisible by $x - 1$ is equivalent to having coefficients that sum to zero mod q.) $\square$

If our original hash function h_A is in fact $2^{\Omega(n)}$ secure, then this result makes h_a uninteresting as a collision-resistant hash function. In particular, in order for h_a to have a chance of matching this security, we would need to take $q = 2^{\Omega(n)}$, in which case h_a is actually a slower hash function than h_A.

10.3 The ring $\mathbb{Z}[x]/(x^n + 1)$, ideal lattices, and a secure collision-resistant hash function

Recall that our attack on h_a over $\mathbb{Z}[x]/(x^n - 1)$ relied on the fact that $x^n - 1$ has a nontrivial factor over the integers, $x^n - 1 = (x - 1)(x^{n-1} + x^{n-2} + \cdots + 1)$. So, it is natural to try replacing $x^n - 1$ with an irreducible polynomial. Indeed, one can easily show that $\mathbb{Z}[x]/(p(x))$ for some polynomial $p(x) \in \mathbb{Z}[x]$ is an integral domain if and only if p is irreducible.

We strongly prefer sparse polynomials with small coefficients (both because they are easy to work with and because this ensures that our ring has nice "geometric" properties). Since $x^n - 1$ failed, we try $x^n + 1$. This is irreducible over $\mathbb{Z}$ if and only if n is a power of two.[3] So, we take $R := \mathbb{Z}[x]/(x^n + 1)$ for n some power of two. I.e., R is the ring of polynomials over $\mathbb{Z}$ of degree at most $n - 1$ with addition defined in the obvious way and multiplication defined by

$$x \cdot x^i = \begin{cases} x^{i+1} & i < n-1 \\ -1 & i = n-1 \end{cases} .$$

From the matrix perspective of the previous section, this corresponds to taking

$$\mathrm{Rot}(\boldsymbol{a}) = (\boldsymbol{a}, X\boldsymbol{a}, \ldots, X^{n-1}\boldsymbol{a}) = \begin{pmatrix} a_1 & -a_n & \cdots & -a_3 & -a_2 \\ a_2 & a_1 & \cdots & -a_4 & -a_3 \\ a_3 & a_2 & \cdots & -a_5 & -a_4 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ a_{n-2} & a_{n-3} & \cdots & -a_n & -a_{n-1} \\ a_{n-1} & a_{n-2} & \cdots & a_1 & -a_n \\ a_n & a_{n-1} & \cdots & a_2 & a_1 \end{pmatrix} \in \mathbb{Z}^{n\times n} ,$$

where

$$X := \begin{pmatrix} 0 & 0 & \cdots & 0 & -1 \\ 1 & 0 & \cdots & 0 & 0 \\ 0 & 1 & \cdots & 0 & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & \cdots & 1 & 0 \end{pmatrix} \in \{0,1\}^{n\times n} .$$

Notice that X differs in just one entry from our choice in the previous section. Matrices of the form $\mathrm{Rot}(\boldsymbol{a})$ as above are occasionally called "anti-cyclic."

As before, we define our hash function $h_a(e) = a_1e_1 + \cdots + a_\ell e_\ell \bmod qR$, where the a_i are chosen uniformly $a_i \in R_{[q]}$ and $e_i \in R_{\{0,1\}}$. But, we stress that the underlying ring has changed from $\mathbb{Z}[x]/(x^n - 1)$ to $R = \mathbb{Z}[x]/(x^n + 1)$, so that this is not the same hash function as before. (Formally, we should include the ring as a parameter in h, i.e. $h_{a,\mathbb{Z}[x]/(x^n+1)}$, to distinguish it, but we prefer to keep the notation uncluttered.) As before, finding a collision for this hash function is equivalent to solving Ring-SIS, now over this new ring, $\mathbb{Z}[x]/(x^n + 1)$.

Ring-SIS is in fact hard over this ring, under a reasonable *worst-case* complexity assumption. We will describe this complexity assumption (which will lead us to the topic of ideal lattices) but will not prove the worst-case to average-case reduction for Ring-SIS.

Remark The ring R is rather special; it is the ring of integers of the cyclotomic number field $\mathbb{Q}[x]/(x^n + 1)$. Number fields and their rings of integers are very well-studied and very interesting objects, and these notes stop short of presenting some of the beautiful mathematics that is lurking beneath the surface here. (The fact that R is such a rich mathematical object also seems relevant for the security of h_a. In particular, there are algorithmic results for related problems that exploit rather deep properties of R [**?**, **?**, **?**, **?**].)

[3] If $p > 1$ is a non-trivial odd factor of n, then $x^{n/p} + 1$ is a non-trivial factor of $x^n + 1$. If n has no odd factors, then $x^n + 1$ is the $2n$th cyclotomic polynomial—i.e., the minimal polynomial over $\mathbb{Z}$ of any primitive $2n$th root of unity.

Ideal lattices

In order to present the worst-case hardness assumption that will imply the security of our hash function, we will need to introduce a special class of lattices known as *ideal lattices.* Recall that a lattice is an additive subgroup of $\mathbb{Z}^n$. I.e., a subset of $\mathbb{Z}^n$ closed under addition and subtraction. An ideal $\mathcal{I} \subseteq R$ is an additive subgroup of a ring R that is closed under multiplication by any ring element. I.e., $\mathcal{I}$ is closed under addition and subtraction, *and* for any $y \in \mathcal{I}$ and $r \in R$, we have $ry \in \mathcal{I}$.

For our choice of ring, we can view $\mathcal{I}$ as a lattice by embedding R in $\mathbb{Z}^n$ via the trivial embedding that maps x^i to the unit vector $\boldsymbol{e}_i$. So, $\mathcal{I}$ can equivalently be viewed as a lattice $\mathcal{I} \subseteq \mathbb{Z}^n$ that is invariant under the linear transformation X. I.e., $\mathcal{I} \subseteq \mathbb{Z}^n$ is a lattice such that $(y_1, \ldots, y_n)^T \in \mathcal{I}$ if and only if $(-y_n, y_1, y_2, \ldots, y_{n-1})^T \in \mathcal{I}$. Such lattices are sometimes called "anti-cyclic," and the corresponding lattices over $\mathbb{Z}[x]/(x^n - 1)$ are often called "cyclic."

In particular, this embedding allows us to consider the geometry of an ideal $\mathcal{I}$, as a subset of $\mathbb{Z}^n$. E.g., we can define the ℓ_2 norm and the inner product over $\mathcal{I}$ by taking the ℓ_2 norm and the inner product over $\mathbb{Z}^n$.[4] We then see that ideal lattices $\mathcal{I}$ are a strange class of lattices in which non-zero lattice elements $y \in \mathcal{I}$ can be divided into groups of n linearly independent elements, $y, xy, x^2y, \ldots, x^{n-1}y$, all with the same length, $\|x^i y\| = \|x^j y\|$. In particular $\lambda_1(\mathcal{I}) = \lambda_n(\mathcal{I})$. (Notice that we move freely between the representation of R as $\mathbb{Z}^n$ and the representation of R as a polynomial ring. I.e., we can think of $y_1, y_2 \in R$ as scalars, written in plain font, as opposed to boldface vectors $\boldsymbol{y}_1, \boldsymbol{y}_2 \in \mathbb{Z}^n$. We can still talk about their norms $\|y_1\|, \|y_2\|$ and inner product $\langle y_1, y_2 \rangle$.)

Remark Ideals are very important objects in the study of rings, and they have a rich history that we do not discuss here. In fact, much of the early study of lattices was motivated by the study of the geometry of ideals, going back all the way to the seminal work of Minkowski, Dirichlet, and others in the middle of the 19th century.

SVP over ideal lattices and worst-case hardness

For our purposes, this view of ideals as lattices is useful because it allows us to extend computational lattice problems to ideals. I.e., for some fixed ring R, we can define the computational problems γ-IdealSVP, γ-IdealSIVP, γ-GapIdealSVP, etc., as the corresponding computational problems restricted to ideal lattices. In fact, the above discussion shows that γ-IdealSVP and γ-IdealSIVP are equivalent over our ring $R = \mathbb{Z}[x]/(x^n + 1)$. A slightly more sophisticated argument shows that γ-GapIdealSVP is easy over R for $\gamma > \sqrt{n}$ because the length of the shortest vector in an ideal can be approximated up to a factor of $\sqrt{n}$ by the determinant. We therefore only present a formal definition of γ-IdealSVP.

Definition 30. *For a ring R (with an associated norm $\|\cdot\|$) and approximation factor $\gamma \geq 1$, γ-IdealSVP over R is the approximate search problem defined as follows. The input is (a basis for) an ideal lattice $\mathcal{I}$ over R. The goal is to output a non-zero element $y \in \mathcal{I}$ with $\|y\| \leq \gamma\lambda_1(\mathcal{I})$.*

[4]For more general rings of integers over number fields, there is actually a different notion of geometry obtained via the "canonical embedding" of $\mathcal{I}$ into $\mathbb{C}^n$, which has very nice properties. E.g., in the canonical embedding, ring multiplication is coordinate-wise. For our very special choice of ring, $\mathbb{Z}[x]/(x^n + 1)$ for n a power of two, these two embeddings actually yield the same geometry.

With this, we can present the worst-case to average-case hardness of Ring-SIS, which was discovered independently by Peikert and Rosen [?] and by Lyubashevsky and Micciancio [?].

Theorem 31 ([?, ?])**.** *For any power of two n, integer $\ell \geq 1$, and integer modulus $q \geq 2n^2\ell$, γ-Ideal SVP over $R = \mathbb{Z}[x]/(x^n + 1)$ can be efficiently reduced to Ring-SIS over R, where $\gamma = \ell \cdot \mathrm{poly}(n)$.*

Regarding the hardness of IdealSVP

Of course, Theorem 31 is only interesting if γ-IdealSVP is hard over the ring $\mathbb{Z}[x]/(x^n + 1)$. Until very recently, our best algorithms for this problem were essentially no better than our generic algorithms for γ-SVP over general n-dimensional lattices. However, very recently, polynomial-time quantum algorithms for γ-IdealSVP with the very large approximation factor $\gamma = 2^{\widetilde{O}(\sqrt{n})}$ were discovered in a series of works [?, ?, ?, ?]. (The best known algorithms for $2^{\sqrt{n}}$-SVP run in time roughly $2^{\sqrt{n}}$, even on a quantum computer. And, our best polynomial-time algorithms for γ-SVP only achieve an approximation factor of $\gamma = 2^{\widetilde{\Theta}(n)}$. So, this is a very big improvement.)

These algorithms are not known to extend to attacks on Ring-SIS for two reasons. First, the approximation factor $\gamma = 2^{\widetilde{O}(\sqrt{n})}$ is much larger than the approximation factors that are relevant to Ring-SIS. Second, Ring-SIS is not exactly an ideal lattice problem. Instead, notice that a solution to Ring-SIS consists of a *vector* of ring elements $(e_1, \ldots, e_\ell) \in R^\ell$. Indeed, Ring-SIS is technically a lattice problem over rank ℓ *modules*. It is therefore not currently known how to efficiently reduce Ring-SIS to IdealSVP.

As a result of all of this, the status of Ring-SIS is a bit unclear at the moment. The barriers mentioned in the previous paragraph seem to be quite hard to overcome, so perhaps this new line of research will not lead to an attack. As far as we know, Ring-SIS is just as hard as SIS, and indeed, as far as we know, it could yield a collision-resistant hash function that is computable in $\widetilde{O}(n)$ time and only breakable in time $2^{\Omega(n)}$.

We will not present the worst-case to average-case reduction for Ring-SIS. It is a slight variant of the reduction that we have already seen for SIS, and is included in the posted lecture notes (`http://people.csail.mit.edu/vinodv/CS294/lecturenotes.pdf`).

The reduction

Finally, we present the worst-case to average-case reduction for Ring-SIS, which is a slight variant of the reduction that we have already seen for SIS. For simplicity, we leave out some details, sweep some technical issues under the rug, and assume that the reader is familiar with the presentation of the SIS reduction from an earlier lecture.

Proof of Theorem 31. The reduction receives as input some ideal $\mathcal{I} \subseteq R$. We may assume without loss of generality that it is also given some parameter s such that $s/2 \leq \sqrt{n}\lambda_n(\mathcal{I}) \leq s$ and some non-zero element $b \in \mathcal{I}$ in the ideal with not-too-large norm, say $\|b\| \leq 2^n\lambda_n(\mathcal{I})$. (Such an element b can be found by running the LLL algorithm, and we can simply try many different parameters $s_i = \sqrt{n}\|b\|/2^{i/2}$ for $i = 0, \ldots, 2n$ to obtain s.) As in the reduction for SIS, we will show a reduction that makes "slow progress." I.e., it finds a non-zero $b' \in \mathcal{I}$ with

$$\|b'\| \leq \ell n^2\|b\|/q + \ell n^{1.5}s \; .$$

We may repeat this procedure, say, $10n$ times to eventually find a vector of length at most, say,

$$10\ell n^{1.5}s \leq 20n^2\lambda_n(\mathcal{I}) = 20n^2\ell\lambda_1(\mathcal{I}) \; ,$$

as needed.

For simplicity, we first assume that $\mathcal{I}$ is a principal ideal generated by b. I.e., every element in $\mathcal{I}$ can be written as rb for some $r \in R$. This is definitely not true in general,[5] and we will sketch how to remove this assumption at the end of the proof.

The reduction samples $\boldsymbol{y}_1, \ldots, \boldsymbol{y}_\ell \in \mathbb{R}^n$ from the (continuous) Gaussian distribution with parameter s. Let $\boldsymbol{b} \in \mathbb{Z}^n$ be the vector of coefficients of b, and let $\boldsymbol{y}_i' \in \mathcal{I}/q$ be $\boldsymbol{y}_i$ with its coordinates in $\mathrm{Rot}(\boldsymbol{b})$ rounded to the nearest integer multiple of $1/q$. Let $y_i' \in R/q$ be the associated polynomial with coefficients given by the vector $\boldsymbol{y}_i'$. Finally, let $a_i \in R_{[q]}$ such that $ba_i = qy_i' \bmod qR$. The reduction calls the Ring-SIS oracle on input $a_1, \ldots, a_\ell$, receiving as output non-zero $e_1, \ldots, e_\ell \in R_{\{-1,0,1\}}$ such that $a_1e_1 + \cdots + a_\ell e_\ell = 0 \bmod qR$. (We might actually need to repeat this procedure many times to receive valid output from the oracle, but we ignore this here.) The reduction outputs

$$b' = y_1'e_1 + \cdots + y_\ell' e_\ell \ .$$

We first note that the input $a_1, \ldots, a_\ell \in R_{[q]}$ is statistically close to random because of our choice of $s > \sqrt{n}\lambda_n(\mathcal{I})$, just like in the reduction for SIS. So, the input to the Ring-SIS oracle is distributed correctly.

We next notice that $b' \in \mathcal{I}$. In particular, we see from the definition of a_i that

$$b' = (ba_1e_1 + \cdots + ba_\ell e_\ell)/q \bmod \mathcal{I} \ .$$

Since the right-hand side is in $\mathcal{I}$, b' is as well.

We next study the length of b'. To do this, we need the inequality $\|ye\| \le \sqrt{n}\|y\|\|e\|$ for any $e, y \in R$, which follows from the definition of polynomial multiplication together with the Cauchy-Schwarz inequality. Therefore,

$$\|b'\| \le \sum_{i=1}^{\ell} \|y_i'e_i\| \le \sqrt{n}\sum_{i=1}^{\ell} \|y_i'\|\|e_i\| \le n\sum_{i=1}^{\ell} \|\boldsymbol{y}_i'\| \le \ell n^2\|b\|/q + n\sum_{i=1}^{\ell} \|\boldsymbol{y}_i\| \ ,$$

where the last inequality follows from the fact that $\|\boldsymbol{y}_i' - \boldsymbol{y}_i\| \le \sum \|x^j b\|/q = n\|b\|/q$. And, since $\|y_i\|$ was sampled from a Gaussian with parameter s, we have $\|\boldsymbol{y}_i\| \le \sqrt{n} \cdot s$ except with negligible probability. The result follows.

So, b' is a short element in the ideal, but we must still show that it is non-zero. Notice that the oracle's input depends only on the coset $y_i \bmod \mathcal{I}$, and if $e_i \neq 0$, there is at most one value of y_i in this coset that yields $b' = 0$. (Notice that this fact is true over an integral domain, when there are no non-trivial zero divisors in our ring. If we used the ring $\mathbb{Z}[x]/(x^n - 1)$ instead, then there could potentially be many values of y_i that cause $b' = 0$, such as when $e_i = \widetilde{e}$ from our earlier attack.) Just like in the SIS case, our choice of parameter $s > \sqrt{n}\lambda_n(\mathcal{I})$ guarantees that y_i has high entropy, even conditioned on its coset. Therefore, b' will often be non-zero.

Finally, we sketch how to remove the assumption that $\mathcal{I}$ is principal. The basic idea is just to still do the reduction but to work with the lattice $\mathcal{I}'$ generated by b. The problem is that we might have $\lambda_n(\mathcal{I}') > s/\sqrt{n}$. (Indeed $\mathcal{I}'$ might not have any vectors shorter than b.) This causes issues in two steps of the proof: when we argue that a_i is uniformly random, and when we argue that b' is non-zero.

[5] Most ideals are not principal at all—i.e., there is no element b that generates the ideal. Even if our ideal is principal, our specific b will typically not be a generator.

To ensure that a_i is still uniformly random, we add to each $\boldsymbol{y}_i$ an element $\boldsymbol{v}_i \in \mathcal{I}$ that is uniformly random mod $\mathcal{I}'$. We can then "subtract out" $\boldsymbol{v}_i$ later to ensure that it does not increase the length of b'. This effectively gives us a short vector in a coset $\mathcal{I}' + v$, where v is an R-linear combination of the $\boldsymbol{v}_i$. In fact, once we have done this, we can show that the distribution over $\boldsymbol{v}_i$ of the input to the oracle depends only on the coset of $\boldsymbol{y}_i$ modulo $\mathcal{I}$. It follows that our output vector b' has high entropy and is therefore rarely zero. □

10.4 Ring-LWE basics and some properties of $\mathbb{Z}[x]/(x^n + 1)$

From Ring-SIS to Ring-LWE

Now, we do unto LWE what we just did to SIS. In particular, the problem (search) LWE asks us to find $\boldsymbol{s} \in \mathbb{Z}_q^n$ given $(A, \boldsymbol{s}^T A + \boldsymbol{e}^T \bmod q)$, where $A \in \mathbb{Z}_q^{n \times m}$ and $\boldsymbol{s} \in \mathbb{Z}_q^n$ are uniformly random and $\boldsymbol{e} \in \mathbb{Z}^m$ is chosen from some error distribution on short vectors. We will define Ring-LWE in a similarly natural way. We will see that the hardness of Ring-LWE implies more efficient public-key cryptography, and that this hardness can be based on the worst-case hardness of the worst-case ideal lattice problem γ-IdealBDD (which we will define later). Because we will rely very heavily on special properties of our specific ring $R = \mathbb{Z}[x]/(x^n + 1)$ for n a power of two, we only define Ring-LWE over this specific ring. Everything presented here can be generalized, but doing so requires quite a bit more work [**?**].[6]

Definition 32. *For integers $\ell, q \geq 2$, power of two n, and an error distribution χ over short elements in R, the (average-case, search) Ring-LWE problem is defined as follows. The input is $a_1, \ldots, a_\ell \in R_q$ sampled independently and uniformly at random together with $b_1, \ldots, b_n \in R_q$, where $b_i := a_i \cdot s + e_i \bmod qR$ for $s \in R_q$, and $e_i \sim \chi$. The goal is to output s.*

Notice that we take s to be worst-case, rather than uniformly random. This is without loss of generality, since we can trivially randomize s if necessary. Just like before, we will also need the decisional version of the problem, which asks us to distinguish the (a_i, b_i) from uniformly random and independent elements of R_q.

Basic properties

Ring-LWE inherits many of LWE's nice properties. In particular, Ring-LWE is equivalent to the planted variant of Ring-SIS, and the hardness of Ring-LWE (both search and decision) remains unchanged if we sample the secret s from the error distribution χ (at the expense of one sample). One can prove both of these facts in more-or-less the same way that we proved the corresponding facts for plain LWE, at least for appropriate choices of q.

For example, given ℓ Ring-LWE samples $(a_1, b_1), \ldots, (a_\ell, b_\ell)$ with $b_i := a_i s + e_i$, we can try to convert them into $\ell - 1$ Ring-LWE samples with the secret sampled from the error distribution as follows. We assume that one of the a_i is invertible in R_q (i.e., there exists an element $a_i^{-1} \in R_q$ such that $a_i a_i^{-1} = 1$, which happens with non-negligible probability, as shown in [**?**, Claim 2.25]).

[6] The "right" notion of Ring-LWE for more general rings has a more sophisticated definition based on the canonical embedding of a number field. In particular, the naive coefficient embedding in which the norm of a ring element is just the norm of its coefficient vector does not behave nicely for general rings. In the special case when $R = \mathbb{Z}[x]/(x^n + 1)$ for n a power of two, the canonical embedding and coefficient embedding are identical (up to scaling and rotation), so we can largely ignore these issues.

Then, $a_j a_i^{-1} b_i = a_j s + a_j a_i^{-1} e_i$, and $a_j a_i^{-1} b_i - b_j = a_j a_i^{-1} e_i + e_j$. We can therefore create the new samples $(a_j a_i^{-1}, a_j a_i^{-1} b_i - b_j)$ for all $j \neq i$, which are $\ell - 1$ valid Ring-LWE samples with secret e_i and error e_j, as needed.

Encryption

Recall that we saw both a secret-key encryption scheme and a public-key encryption scheme from plain LWE. Both of these schemes have natural analogues in the Ring-LWE world. Just like our Ring-SIS-based hash function, these schemes are remarkably efficient.

The secret-key encryption scheme is as follows. Both this scheme and the public-key scheme naturally use $R_{\{0,1\}}$ as their message space, i.e., polynomials with $\{0, 1\}$ coefficients. (Compare this to the one-bit message space that we obtained for LWE.)

- **Key generation:** The secret key is simply a uniformly random element $s \in R_q$.
- **Encryption:** To encrypt $m \in R_{\{0,1\}}$, compute (a, b) for $b := a \cdot s + e + \lfloor q/2 \rceil \cdot m \bmod qR$, where $a \in R_q$ is chosen uniformly at random and $e \sim \chi$.
- **Decryption:** To decrypt (a, b), compute $b - a \cdot s \bmod qR = \lfloor q/2 \rceil \cdot m + e \bmod qR$. Round each coefficient to either $q/2$ or zero, whichever is closest (where we assume that our representation modulo qR uses coefficients in $[q]$), and interpret 0 as 0 and $q/2$ as 1.

Clearly, this scheme is correct if and only if the coefficients of e are smaller than roughly $q/4$. Furthermore, the CPA-security of the scheme is immediate from Ring-LWE. And, this scheme is quite efficient:

- secret keys have size $n \log q$;
- encrypting n-bit messages using roughly $n \log q$-bit ciphertexts; and
- encryption and decryption run in time $n \cdot \mathrm{polylog}(n, q)$.

As far as we know, this scheme is $2^{\Omega(n)}$ secure for appropriate parameters, so that we may take n only linear in the security parameter.

As a parenthetical remark, we can achieve such short ciphertexts from LWE as well (as shown by Peikert, Vaikuntanathan and Waters.) with the following properties:

- secret keys have size $n^2 \log q$;
- encrypting n-bit messages using roughly $n \log q$-bit ciphertexts; and
- encryption and decryption run in time $n^2 \cdot \mathrm{polylog}(n, q)$.

The public-key encryption scheme is as follows.

- **Key generation:** The secret key is a short secret $s \sim \chi$. The public key is $(\widehat{a}, y)$ for $\widehat{a} \in R_q$ uniformly random and $y := \widehat{a} \cdot s + e \bmod qR$, where $e \sim \chi$.
- **Encryption:** To encrypt $m \in R_{\{0,1\}}$, compute (a, b), where $a := \widehat{a} r + x \bmod q$ and $b := yr + x' + \lfloor q/2 \rceil\, m \bmod q$ for $r, x, x' \sim \chi$.

- **Decryption:** To decrypt (a, b), compute $b - a \cdot s \bmod qR = \lfloor q/2 \rceil m + er + x' - xs \bmod q$ and again do our rounding procedure to find m.

Clearly, this scheme is correct if and only if $er + x' - xs$ is less than $q/4$. (So, we can take our error to have size roughly $\sqrt{q}/2$.) Security follows from a proof similar to the one for plain LWE in our first lecture. I.e., we use the hardness of decisional Ring-LWE with short secrets once to show that the public key can be replaced by uniformly random ring elements and then again to show that the element b in the ciphertext can also be replaced by a uniformly random ring element.

Again, we note the remarkable efficiency of this scheme. As far as we know, it is $2^{\Omega(n)}$ secure and all operations are computable in time $n \cdot \mathrm{polylog}(n, q)$. Taking $q = \mathrm{poly}(n)$ gives a public-key encryption scheme with key generation, encryption, and decryption all computable in time quasilinear in the security parameter. And, Lyubashevsky, Peikert, and Regev proved that breaking this scheme is at least as hard as a certain worst-case ideal lattice problem [**?**]—even an ideal lattice problem that is plausibly $2^{\Omega(n)}$ hard.

Reduction modulo ideals and Chinese Remainder Theorem

Recall that for an element $r \in R$ in some ring R (e.g., $R = \mathbb{Z}$), we define equivalence of $s_1, s_2 \in R$ modulo r by $s_1 = s_2 \bmod r$ if and only if there exists an $r' \in R$ with $s_1 = s_2 + r'r$. Equivalently, $s_1 = s_2 \bmod r$ if and only if there exists an *ideal* element $y \in rR := \{r' \cdot r \ : \ r' \in R\}$ in the ideal rR generated by r such that $s_1 = s_2 + y$. This is an equivalence relation because the ideal is closed under addition, which also implies that it respects addition. It respects multiplication because the ideal is closed under multiplication by any ring element. I.e., if $s_1 = s_2 + y$ for $y \in rR$, then $xs_1 = xs_2 + xy$, which implies that $xs_1 = xs_2 \bmod r$, since $xy \in rR$ also.

This immediately shows that we can also reduce modulo an arbitrary ideal $\mathcal{I}$, not just an ideal generated by a single element. I.e., we define $s_1 = s_2 \bmod \mathcal{I}$ if and only if there exists $y \in \mathcal{I}$ such that $s_1 = s_2 + y$. (This is a big part of the reason why ideals are such important objects in the study of rings, as opposed to, say, subrings.) Just like before, addition and multiplication are well defined modulo $\mathcal{I}$, and we write $R/\mathcal{I}$ for the ring of equivalence classes modulo $\mathcal{I}$.[7]

We will need something slightly more general. For an ideal $\mathcal{J} \subseteq \mathcal{I}$ (e.g, $\mathcal{J} = q\mathcal{I}$), we can again define the quotient $\mathcal{I}/\mathcal{J}$. This quotient is also a ring, and we can define multiplication by x in $\mathcal{I}/\mathcal{J}$ in the obvious way.

We can now present the Chinese Remainder Theorem over R. (A far more general theorem holds here over a very large class of rings.) We say that two ideals $\mathcal{I}$ and $\mathcal{J}$ are *coprime* if there exists $y \in \mathcal{I}, z \in \mathcal{J}$ such that $y + z = 1$.

Theorem 33 (Chinese Remainder Theorem for R)**.** *For any pairwise coprime ideals $\mathcal{I}_1, \ldots, \mathcal{I}_k \subseteq R$ over R, let $\mathcal{I} := \bigcap \mathcal{I}_j$. Then, $R/\mathcal{I}$ is isomorphic (as a ring) to the direct product*

$$\frac{R}{\mathcal{I}_1} \times \frac{R}{\mathcal{I}_2} \times \cdots \times \frac{R}{\mathcal{I}_k} \ .$$

Indeed, an isomorphism is given by the natural map

$$r \mapsto (r \bmod \mathcal{I}_1, r \bmod \mathcal{I}_2, \ldots, r \bmod \mathcal{I}_k) \ ,$$

and it can be efficiently inverted.

[7] In fact, we have already been sneakily using this notation, writing $\bmod qR$, rather than $\bmod q$.

Furthermore, in the special case when $\mathcal{I} = qR$ for a prime q, we may take the $\mathcal{I}_j$ to be the ideal generated by q and the jth irreducible factor of $x^n + 1$ modulo q. Then, the quotients $R/\mathcal{I}_j$ are actually fields of characteristic q.

In particular, turning back to the question of invertibility of a_i from Section 10.4, we see that at least for prime q, $a \in R_q$ is invertible unless $a = 0 \bmod \mathcal{I}_j$ for some j (since the quotients are fields and therefore do not have zero divisors). Since the quotient has size at least q, this happens with probability at most $1/q$. Because of the product structure guaranteed by the Chinese Remainder Theorem, we then see that a is invertible with probability at least $(1 - 1/q)^n$.

10.5 Search to decision

We will prove the following search-to-decision reduction for Ring-LWE, which was originally proven by Lyubashevsky, Peikert, and Regev [**?**]. We say that a polynomial *splits mod q* if it is the product of distinct linear factors modulo q. We say that an error distribution χ over R is *spherically symmetric* if the probability of sampling a ring element from χ depends only on its norm.

Theorem 34. *For a prime $q \geq 2$, integer $\ell \geq 2$, power of two n such that $x^n + 1$ splits mod q, and spherically symmetric error distribution χ, there is a reduction from search Ring-LWE to decision Ring-LWE that runs in time $q \cdot \mathrm{poly}(n, \ell)$*

Many new issues arise in the ring setting. Therefore, the proof is quite a bit more difficult than the relatively easy proof for plain LWE. Indeed, lurking behind this reduction is quite a bit of Galois theory. (We refer the reader to [**?**] for a much more thorough discussion.) Furthermore, the result is not entirely satisfying for at least two reasons.

First, the running time proportional to q is unfortunate, since our worst-case to average-case reduction will only work for exponentially large moduli q. Recall that we had this issue in the plain LWE case as well, but there we saw that modulus-switching techniques can be used to reduce exponential q to polynomial q (with a large loss in parameters) [**?**]. However, nothing similar is known in the Ring-LWE setting. Indeed, the only hardness results known for Ring-LWE with small q use a quantum reduction [**?**, **?**], which we will not present here.

Second, the fact that our polynomial $x^n + 1$ splits mod q is a bit worrisome, since we saw an attack on Ring-SIS when the polynomial modulus has a high-degree factor with small coefficients over the integers. *That* attack does extend to Ring-LWE, but as far as we know, there is no attack that *exploits a modulus q over which x^n+1 factors.* Indeed, the worst-case to average-case reduction in [**?**] shows that, if Ideal-SVP with appropriate parameters is hard for a quantum computer, then Ring-LWE is also hard for any sufficiently large modulus q, regardless of whether $x^n + 1$ splits modulo q.

Where we're going

Recall that our search-to-decision reduction for plain LWE worked by guessing the coordinates of the secret vector $\boldsymbol{s} \in \mathbb{Z}_q^n$ one by one. One might therefore hope to find a similar reduction for Ring-LWE that works by guessing the *coefficients* of the secret ring element $s \in R_q$ one by one. However, it is not at all clear how to do this. In the plain LWE case, we crucially used the fact that knowing a coordinate of $\boldsymbol{s}$ allows us to compute $\langle \boldsymbol{a}, \boldsymbol{s} \rangle \bmod q$ for some $\boldsymbol{a} \in \mathbb{Z}_q^n$. (Namely, the standard basis vector corresponding to the relevant coordinate.) However, knowing just one coefficient of s (or even $n-1$ coefficients of s) does not allow us to compute $a \cdot s \bmod qR$ for any non-zero $a \in R_q$.

We will need to develop a few tools in the next few subsections to correct this. The high-level structure is as follows. First, we show how to use a different coordinate system, based on the Chinese Remainder Theorem, to make multiplication coordinate-wise. This is nice because it allows us to guess a coordinate in a meaningful way. However, when we guess wrong, we will not end up with uniformly random samples. Instead, we will get Ring-LWE samples that are uniform in just one coordinate, and it is not immediately clear how to use a decision oracle to distinguish these two cases. In order to get around this, we will show the existence of very special functions that essentially allow us to "swap" coordinates. Finally, we will use a hybrid argument together with these tools to prove that hardness of search Ring-LWE implies hardness of decision Ring-LWE.

The CRT embedding (which is very different from the coefficient embedding!)

Our first task is to find a coordinate system in which multiplication is coordinate-wise. E.g., in these coordinates, the product of $(s_1, s_2, \ldots, s_n)$ with $(1, 0, 0, \ldots, 0)$ should simply be $(s_1, 0, 0, \ldots, 0)$. Indeed, since $x^n + 1$ splits modulo q, the Chinese Remainder Theorem tells us that R_q is isomorphic as a ring to the ring $\mathbb{Z}_q^n$ under coordinate-wise multiplication. So, we can in fact write ring elements $a, s \in R_q$ in a coordinate system such that $a \cdot s = (a_1 s_1, \cdots a_n s_n)$. We call this the *CRT embedding*, in contrast to the *coefficient embedding* in which we view the ring elements as polynomials. We recall that the Chinese Remainder Theorem guarantees that we can move efficiently between these two embeddings. (Indeed, this is accomplished via an invertible linear map over the field $\mathbb{Z}_q$.)

It might seem a bit silly to have gone through all of the trouble of defining Ring-LWE over polynomial rings just to end up working with $\mathbb{Z}_q^n$ under coordinate-wise multiplication! But, we stress that the error distribution looks quite different in the CRT embedding. (If we used as an error distribution that is short in the CRT embedding, the resulting Ring-LWE problem would be easy.)

To see this, let's consider the smallest non-trivial example. The polynomial x^2+1 splits modulo 13 as $x^2 + 1 = (x+5)(x-5) \bmod 13$, so an element $ax + b \in \mathbb{Z}_{13}/(x^2+1)$ has CRT representation $(5a + b, b - 5a) \in \mathbb{Z}_{13}^2$. (Check this!) Therefore, if our initial error distribution is, say, uniform over polynomials with $a, b \in \{-1, 0, 1\}$, then in the CRT embedding, our error distribution is uniform over the rather strange set $\{(0,0), \pm(5,-5), \pm(1,1), \pm(6,-4), \pm(6,-4)\}$, which in particular contains quite long elements, relative to $q = 13$. I.e., the mapping from the coefficient embedding to the CRT embedding is a linear transformation with large distortion (to the extent that one can define "distortion" over a finite vector space).

So, while we *can* equivalently define Ring-LWE in terms of $\mathbb{Z}_q^n$ (when $x^n + 1$ splits modulo q), we would end up with a much less natural error distributions. In particular, the error distributions obtained from our worst-case to average-case reduction would be rather strange and depend on q in complicated ways.

When and how $x^n + 1$ splits modulo q

We now consider when $x^n + 1$ splits modulo q and show that the factors take a nice form. Notice that x^n+1 is the minimal polynomial over $\mathbb{Z}$ of the (complex) primitive $2n$th roots of unity $e^{k\pi i/(2n)}$ for odd k. I.e., x^n+1 splits over $\mathbb{C}$ precisely because $\mathbb{C}$ contains such elements. In analogy with this, suppose that $z \in \mathbb{Z}_q^*$ is a primitive $2n$th root of unity modulo q. That is, suppose $z^{2n} = 1 \bmod q$ but

$z^k \neq 1 \bmod q$ for all $0 < k < 2n$. Then, clearly $z^n \neq 1 \bmod q$ is a square root of 1 in $\mathbb{Z}_q$. Since $\mathbb{Z}_q$ is a field, the only square roots of 1 are ± 1, so we must have $z^n = -1 \bmod q$. I.e., $z^n + 1 = 0 \bmod q$.

Furthermore, for any odd k, z^{kn} is also a primitive $2n$th root of unity. So, $z, z^3, z^5, \ldots, z^{2n-1} \in \mathbb{Z}_q^n$ are all roots of $x^n + 1$ modulo q. Indeed, they are distinct because $z^k \neq 1$ for $0 < k < 2n$. Finally, since $\mathbb{Z}_q$ is a field, there is only one non-zero polynomial over $\mathbb{Z}_q$ of degree n with these roots, and we must have $x^n + 1 = (x - z)(x - z^3)(x - z^5) \cdots (x - z^{2n-1}) \bmod q$. I.e., $x^n + 1$ splits modulo q.

So, $x^n + 1$ splits modulo a prime q if (and only if) there is an element of order $2n$ in $\mathbb{Z}_q^*$ (i.e., a primitive $2n$th root of unity modulo q). To find such a prime, we recall that $\mathbb{Z}_q^*$ is cyclic of order $q - 1$, so that it has an element of order $2n$ if and only if $2n$ divides $q - 1$. Therefore, $x^n + 1$ splits modulo a prime q if (and only if) $q = 1 \bmod 2n$. The Prime Number Theorem in arithmetic progressions guarantees that such primes exist and can be found efficiently. And, when this is the case, the factors of $x^n + 1$ modulo q can be written as $x - z^k$ for all odd $1 \leq k \leq 2n - 1$.

Some very special automorphisms τ_k

The above discussion shows a very natural way to think of the coordinates CRT embedding. Each coordinate in the CRT embedding of a polynomial $p(x)$ is simply $p(x) \bmod \mathcal{I}_i = p(z^{2i-1}) \bmod \mathcal{I}_i$ for some $i \in [n]$, where z is some fixed primitive $2n$th root of unity modulo q and $\mathcal{I}_i$ is the ideal generated by q and $x - z^{2i-1}$. It is therefore natural to order the coordinates in the CRT embedding so that the ith coordinate is $p(z^{2i-1})$. We then observe a nice symmetry of the CRT embedding. Let $k := (2i-1)^{-1}(2j-1) \bmod 2n$ (where we have used the fact that all odd numbers have an inverse modulo $2n$). Then, we see that the ith CRT coordinate of $p(x) \in R_q$ is the jth CRT coordinate of $p(x^k)$.

So, we define $\tau_k : R_q \to R_q$ for odd k such that $\tau_k(p(x)) := p(x^k)$. We see that τ_k can be viewed as a certain permutation of the coordinates in the CRT embedding. It is therefore a ring automorphism (i.e., it is a bijection respecting addition and multiplication). In fact, it also preserves norms in the coefficient embedding! I.e., $\|\tau_k(p(x))\| = \|p(x^k)\| = \|p(x)\|$, which can be seen by observing that τ_k simply permutes the coordinates of $p(x)$ (and flips some of their signs). Such maps are very rare,[8] and very useful. The next lemma extracts the specific property that we will need from them.

Lemma 35. *The maps $\tau_k : R_q \to R_q$ as described above are efficiently computable ring automorphisms preserving the norm (in the coefficient embedding). Furthermore, τ_k acts on the CRT embedding by permuting the coordinates, and for each $i, j \in [n]$, there is an efficiently computable k such that τ_k maps the ith CRT coordinate to the jth CRT coordinate.*

The reduction

We can now finally present our reduction. As we discussed above, we can guess the coordinate s_1 and replace the Ring-LWE sample (a_i, b_i) by

$$(a_i + \alpha_i v_1 \bmod qR, b + \alpha_i \sigma_1 v_1 \bmod qR)$$

[8] As we've described these maps here, they only exist for our specific choice of R_q! They can, however, be generalized to more rings if we work in the canonical embedding rather than the coefficient embedding [**?**].

where $v_1 = (1, 0, 0, \dots, 0)^T$ in the CRT embedding, $\sigma_1 \in \mathbb{Z}_q$ is our guess for the first coordinate s_1 of s in the CRT embedding, and $\alpha_i \in \mathbb{Z}_q$ is uniformly random. Clearly, when our guess σ_1 is correct, the result is still a valid Ring-LWE sample with the same secret s, and the same error. However, when σ_1 is not correct, the result is not uniformly random. Instead, the first coordinate in the CRT embedding is uniformly random, but the remaining coordinates are completely unchanged.

To fix this, we use a hybrid argument together with the special maps τ_k. In particular, we let $\mathsf{Ring\text{-}LWE}_j$ be the variant of decision Ring-LWE that asks us to distinguish Ring-LWE samples in which the first $j-1$ coordinates in the CRT embedding are replaced by uniformly random noise from Ring-LWE samples in which the first j CRT coordinates are replaced by uniformly random noise. To show the hardness of decision Ring-LWE, it suffices to show the hardness of $\mathsf{Ring\text{-}LWE}_j$ for each j.

Notice that the above argument lets us use an oracle for $\mathsf{Ring\text{-}LWE}_1$ to learn the first coordinate s_1 in the CRT embedding of the secret s of a Ring-LWE instance. More generally, we can use an oracle for $\mathsf{Ring\text{-}LWE}_j$ to find the jth coordinate s_j. So, to finish our proof, we need to show how the ability to find the jth coordinate s_j in the CRT embedding allows us to find all coordinates s_i. This is where we use the maps τ_k. In particular, Lemma 35 lets us find a k such that τ_k maps the ith coordinate to the jth coordinate in the CRT embedding. Since τ_k is a ring automorphism, it converts Ring-LWE samples with secret s to Ring-LWE samples with secret $\tau_k(s)$. Furthermore, since τ_k preserves the norm and the error distribution χ is spherically symmetric, τ_k preserves the error distribution.

So, our full reduction from search Ring-LWE to $\mathsf{Ring\text{-}LWE}_j$ behaves as follows. (We will denote ring elements by bold-face.) For each $i = 1, \dots, n$, we use our $\mathsf{Ring\text{-}LWE}_j$ oracle to find the ith coordinate s_i of $\boldsymbol{s}$ in the CRT embedding by first computing $k = (2i-1)^{-1}(2j-1) \bmod 2n$ such that τ_k maps the ith CRT coordinate to the jth CRT coordinate, as in Lemma 35.

Let $\boldsymbol{v}_j \in R_q$ be the element whose coordinates in the CRT embedding are $(0, 0, \dots, 1, 0, \dots, 0)$, where the 1 is in the jth position. For each $\sigma \in \mathbb{Z}_q$, we replace our Ring-LWE samples $(\boldsymbol{a}_\ell, \boldsymbol{b}_\ell)$ by

$$(\tau_k(\boldsymbol{a}_\ell) + \alpha_\ell \boldsymbol{v}_j, \tau_k(\boldsymbol{b}_\ell) + \sigma \alpha_\ell \boldsymbol{v}_j + \boldsymbol{u}_\ell)$$

where $\alpha_\ell \in \mathbb{Z}_q$ is uniformly random, and $\boldsymbol{u}_\ell \in R_q$ has its first $j-1$ coordinates uniformly random in the CRT embedding and last $n-j+1$ coordinates equal to zero. If $\sigma = s_i$, then the resulting distribution

$$(\tau_k(\boldsymbol{a}_\ell) + \alpha_\ell \boldsymbol{v}_j, \tau_k(\boldsymbol{a}_\ell)\tau_k(\boldsymbol{s}_\ell) + \alpha_\ell \sigma \boldsymbol{v}_j + \tau_k(\boldsymbol{e}_\ell) + \boldsymbol{u}_\ell)$$

will be exactly the YES case of $\mathsf{Ring\text{-}LWE}_j$ with secret $\tau_k(\boldsymbol{s})$—i.e., the first $j-1$ coordinates will be uniformly random and the last $n-j+1$ coordinates will correspond to valid Ring-LWE samples. Otherwise, the distribution will be exactly the NO case—i.e., the jth coordinate will also be uniformly random.

10.6 The worst-case to average-case reduction

We can now move on to the worst-case to average-case reduction for (search) Ring-LWE. Fortunately, very little of the math from the previous section is needed for this part. We will, however, assume a basic familiarity with the worst-case to average-case reduction for plain LWE, as presented in the earlier lectures or as described in [**?**] (or in [**?**] as the "classical part").

Recall that we defined the Bounded Distance Decoding problem (BDD) as the problem that asks us to find a lattice vector $\boldsymbol{y} \in \mathcal{L}$ that is closest to some target $\boldsymbol{t}$, given the promise that this vector is actually very close, $\mathcal{D}(\boldsymbol{t},\mathcal{L}) \ll \lambda_1(\mathcal{L})$. We define the analogous problem for ideal lattices over R. (We take our target $t \in \mathbb{R}[x]/(x^n+1)$, but we do not need many properties of this structure. We just need that it is a ring that contains R and that we can round from $\mathbb{R}[x]/(x^n+1)$ to R in the natural way.)

Definition 36. *For a power of two n and an approximation factor $\alpha < 1/2$, the α-IdealBDD problem over $R = \mathbb{Z}[x]/(x^n+1)$ is defined as follows. The input is (a basis for) an ideal $\mathcal{I}$ over R and a target $t \in \mathbb{R}[x]/(x^n+1)$ such that $\mathcal{D}(t,\mathcal{I}) \le \alpha\lambda_1(\mathcal{I})$. The goal is to output the (unique) element $y \in \mathcal{I}$ with $\|y-t\| \le \alpha\lambda_1(\mathcal{I})$.*

In the plain LWE case, we reduced BDD to LWE, and then we reduced GapSVP to BDD, so that we were able to prove hardness from a more standard lattice problem. We could do the same thing here, but recall that γ-IdealGapSVP for $\gamma > \sqrt{n}$. So, our worst-case problem will simply be IdealBDD. (There is actually a quantum reduction that reduces IdealSVP to IdealBDD with appropriate parameters [**?**, **?**], but we will not present this here.) In particular, we prove the following theorem. (To make the proof easier, we have chosen the rather extreme noise parameter $\sigma > n^{\omega(1)}\alpha q$, which makes the theorem vacuous unless $\alpha < n^{-\omega(1)}$. More careful analysis of essentially the same reduction gives $\sigma \ge \mathrm{poly}(n,\ell)\alpha q$ for some fixed polynomial and therefore allows for $\alpha = 1/\mathrm{poly}(n,\ell)$. Removing the dependence on ℓ takes much more work [**?**].)

Theorem 37 (Weak variant of [**?**])**.** *For any power of two n, $q \ge 2^n$, and any $\alpha < n^{-\omega(1)}$, there is an efficient reduction from α-IdealBDD over $R = \mathbb{Z}[x]/(x^n+1)$ to Ring-LWE over R with noise sampled from the discrete Gaussian $D_{R,\sigma}$ over R with parameter $\sigma > n^{\omega(1)}(n+\alpha q)$.*

The inverse ideal and discrete Gaussian samples

As in the reduction for plain LWE, a key tool will be the dual $\mathcal{L}^*$ of our worst-case lattice $\mathcal{L}$. Recall that $\mathcal{L}^*$ is defined as the set of vectors that have integral inner product with every lattice vector,

$$\mathcal{L}^* := \{\boldsymbol{w} \in \mathbb{Q}^n \ : \ \forall \boldsymbol{y} \in \mathcal{L},\ \langle \boldsymbol{w},\boldsymbol{y}\rangle \in \mathbb{Z}\} \ .$$

One can check that $\mathcal{L}^*$ is itself a (scaling of a) lattice. The important property of $\mathcal{L}^*$ that we use in the reduction is that, for a BDD target $\boldsymbol{t} \in \mathbb{R}^n$, we can write

$$\langle \boldsymbol{w},\boldsymbol{t}\rangle = \langle \boldsymbol{w},\boldsymbol{y}\rangle + \langle \boldsymbol{w},\boldsymbol{e}\rangle \ ,$$

where $\langle \boldsymbol{w},\boldsymbol{y}\rangle$ is an integer and $\langle \boldsymbol{w},\boldsymbol{e}\rangle$ is relatively small.

In the context of ideals, we will instead work with the *inverse* ideal

$$\mathcal{I}^{-1} := \{w \in \mathbb{Q}[x]/(x^n+1) \ : \ \forall y \in \mathcal{I},\ w \cdot y \in R\} \ .$$

(There is a notion of a dual ideal that is *different* than this notion, whose definition only makes sense in the canonical embedding. So, we avoid calling $\mathcal{I}^{-1}$ the "dual ideal." Again, we are relying here on the very special properties of the ring $\mathbb{Z}[x]/(x^n+1)$ for n a power of two in order to simply things.) To see that this is an ideal, we simply need to observe that (1) it is closed under addition, and (2) it is closed under multiplication by x. Both facts are immediate from the relevant definitions. E.g., $(xw) \cdot y = w \cdot (xy) \in R$ for any $w \in \mathcal{I}^{-1}$.[9] Furthermore, we have the ring analogue of the above identity,

$$wt = wy + we \ ,$$

where $wy \in R$ and we is short.

As in the plain LWE case, our vectors w will be sampled from the discrete Gaussian distribution $D_{\mathcal{I}^{-1},\sigma'}$, defined by

$$\Pr_{W \sim D_{\mathcal{I}^{-1},\sigma'}}[W = w] \propto e^{-\pi\|w\|^2/\sigma'^2}$$

for all $w \in \mathcal{I}^{-1}$. (Since this is a probability distribution, we only need to define this up to the constant of proportionality.)

We will only need some very basic properties from $D_{\mathcal{I}^{-1},\sigma'}$. First, we can sample efficiently from $D_{\mathcal{I}^{-1},\sigma'}$ for $\sigma' > 2^n \cdot \lambda_n(\mathcal{I}^{-1})$. Second, for $\sigma' > \sqrt{n}q \cdot \lambda_n(\mathcal{I}^{-1})$, a sample $w \sim D_{\mathcal{I}^{-1},\sigma'}$ is statistically close to uniformly random modulo $q\mathcal{I}^{-1}$. Third, a sample $w \sim D_{\mathcal{I}^{-1},\sigma'}$ is not too long, i.e. except with negligible probability we have $\|w\| < \sqrt{n}\sigma'$. We have seen all of these properties in previous lectures, and none of them depend on the ideal structure at all.

Mapping $\mathcal{I}^{-1}/(q\mathcal{I}^{-1})$ to R_q

We noted in the previous section that $w \sim D_{\mathcal{I}^{-1},\sigma'}$ is essentially uniformly random modulo $q\mathcal{I}^{-1}$ for appropriate σ'. We would like to create Ring-LWE samples (a, b) where $a \in R_q = R/(qR)$ somehow corresponds to this coset. I.e., we would like to map $\mathcal{I}^{-1}/(q\mathcal{I}^{-1})$ to $R/(qR)$. In the plain LWE world, this was simply a matter of computing the coordinates $\mathbf{B}^{-1}\boldsymbol{w} \bmod q$ modulo q in some basis $\mathbf{B}$ for $\mathcal{L}^*$ of the dual lattice vector $\boldsymbol{w} \in \mathcal{L}^*$.

In the ring case, we must be more careful because our map must preserve the multiplicative structure of $\mathcal{I}^{-1}/(q\mathcal{I}^{-1})$ and R_q. We will need a bijective map $\theta : \mathcal{I}^{-1}/(q\mathcal{I}^{-1}) \to R_q$ that is linear (i.e., $\theta(w_1 + w_2) = \theta(w_1) + \theta(w_2) \bmod qR$ for any $w_1, w_2 \in \mathcal{I}^{-1}/(q\mathcal{I}^{-1})$) and respects multiplication by any ring element, i.e., $\theta(xw) = x\theta(w) \bmod qR$ for all $w \in \mathcal{I}^{-1}/(q\mathcal{I}^{-1})$. (Formally, this is an isomorphism of R modules.)

Lemma 38. *For $q \geq 2$ and an ideal $\mathcal{I} \subseteq R$, let $z \in \mathcal{I}$ be any element such that there exists $\widehat{w} \in \mathcal{I}^{-1}$ and $\widehat{r} \in R$ such that $\widehat{w}z + \widehat{r}q = 1$. Then, the map $\theta_z : \mathcal{I}^{-1}/(q\mathcal{I}^{-1}) \to R_q$ defined by $\theta_z(w) = zw \bmod qR$ is a linear bijection satisfying $\theta_z(xw) = x\theta_z(w) \bmod qR$ for all $w \in \mathcal{I}^{-1}/(q\mathcal{I}^{-1})$.*

Furthermore, such a z always exists and can be found efficiently given (a basis for) $\mathcal{I}$ and q, and θ_z and its inverse are efficiently computable.

Proof. It follows from the basic properties of multiplication that θ_z is linear and respects multiplication. So, we only need to prove that θ_z is a bijection. Indeed, multiplication by $\widehat{w}$ is the inverse of θ_z. I.e., for each $r \in R$, we have $\theta_z(\widehat{w}r) = z\widehat{w}r \bmod qR = r \bmod qR$, where we have used the fact that $\widehat{w}r = 1 \bmod qR$. Therefore, θ_z is surjective. To prove that it is bijective, it suffices to note that the domain and range have the same size, q^n. The fact that θ_z is efficiently computable is trivial, and the fact that it can be inverted efficiently follows from the fact that it is a linear bijection. (I.e., we can write down the matrix corresponding to θ_z, which is guaranteed to have an inverse. So, we can simply compute its inverse.)

[9] Since $\mathcal{I}^{-1} \not\subseteq R$, it is technically only a *fractional* ideal. I.e., there exists some denominator $z \in \mathbb{Z}$ and some ideal $\mathcal{J} \subseteq R$ such that $\mathcal{I}^{-1} = z^{-1}\mathcal{J}$.

For the proof that z exists and can be found efficiently, see [**?**, Lemma 2.14]. It relies on the Chinese Remainder Theorem together with the factorization of an ideal into the product (i.e., intersection) of prime ideals (i.e., ideals $\mathcal{P}$ such that for any ideal $\mathcal{J}$, either $\mathcal{P}$ and $\mathcal{J}$ are coprime or $\mathcal{J} \subseteq \mathcal{P}$). □

The reduction

Recall that in the plain LWE reduction, we sampled $\boldsymbol{w}_i \sim D_{\mathcal{L}^*,\sigma'}$ for appropriately chosen parameter $s > 0$ and created LWE samples

$$(\mathbf{B}^{-1}\boldsymbol{w}_i \bmod q, \lfloor \langle \boldsymbol{w}_i, \boldsymbol{t} \rangle \rceil + \widetilde{e}_i \bmod q) ,$$

where $\mathcal{L} \subseteq \mathbb{Z}^n$ is the input lattice to the BDD instance, $\boldsymbol{t} \in \mathbb{R}^n$ is the input target, and $\widetilde{e}_i$ is some extra noise that we add.

In the Ring-LWE world, the natural analogue is as follows. We take as input (a basis for) an ideal $\mathcal{I} \subseteq R$ over R and a target $t \in \mathbb{R}[x]/(x^n + 1)$. We sample $w_1, \ldots, w_\ell \sim D_{\mathcal{I}^{-1},\sigma'}$ from the discrete Gaussian for $\sqrt{n}q\lambda_n(\mathcal{I}^{-1}) < \sigma' \le 2\sqrt{n}q\lambda_n(\mathcal{I}^{-1})$.[10]

We then create Ring-LWE samples (a_i, b_i) with $b_i := \lfloor w_i \cdot t \rceil + \widetilde{e}_i \bmod qR$, where $\widetilde{e}_i \sim \chi$ is some additional noise and $\lfloor \cdot \rceil$ just means rounding the coefficients to integers. To create a_i, we find $z \in \mathcal{I}$ with $\theta_z : \mathcal{I}^{-1}/(q\mathcal{I}^{-1}) \to R_q$ as guaranteed by Lemma 38. We then set $a_i := \theta_z(w_i) = zw_i \bmod qR$. Notice that the fact that w_i is statistically close to uniformly random modulo $q\mathcal{I}^{-1}$ together with the fact that θ_z is a bijection immediately implies that $a_i \in R_q$ is statistically close to uniformly random.

It remains to study the distribution of b_i. We can write

$$\lfloor w_i \cdot t \rceil + \widetilde{e}_i \bmod qR = w_i y + \lfloor w_i e \rceil + \widetilde{e}_i \bmod qR ,$$

where $y \in \mathcal{I}$ is a closest lattice vector to t, and $e := y - t$, and we have used the fact that $w_i y \in R$ to write $\lfloor w_i y + w_i e \rceil = w_i y + \lfloor w_i e \rceil$. We are promised that $\|e\| \le \alpha\lambda_1(\mathcal{I})$.

We first study the error term $\lfloor w_i e \rceil + \widetilde{e}_i$. We just need to show that $\| \lfloor w_i e \rceil \| \le \sigma/n^{\omega(1)}$ is short, since $\widetilde{e}_i$ is sampled from a Gaussian with parameter σ (and it is a basic fact that a Gaussian with parameter σ is within statistical distance $O(d/\sigma)$ of the same Gaussian shifted by d). Indeed, recall from the previous lecture that $\|w_i e\| \le \sqrt{n}\|w_i\|\|e\|$ (which follows immediately from Cauchy-Schwarz). We noted earlier that $\|w_i\| \le \sqrt{n}\sigma \le 2nq\lambda_n(\mathcal{I}^{-1})$ with high probability. And, by assumption $\|e\| \le \alpha\lambda_1(\mathcal{I}) \le \alpha n/\lambda_n(\mathcal{I}^{-1})$, where the second inequality is the transference theorem that we discussed in an earlier lecture [**?**]. Putting all of this together, we see that $\| \lfloor w_i e \rceil \| \le n + \|w_i e_i\| \le n + 2\alpha n^{2.5} q$ with high probability. Since we took $\sigma > n^{\omega(1)}(n + \alpha q)$, our error is statistically close to a Gaussian with parameter σ. (I.e., the additional noise from $w_i e$ does not affect the overall distribution much.)

Next, we turn to $w_i y$. Recall from Lemma 38 that we can write $1 = \widehat{w}z + \widehat{r}q$ for some $\widehat{w} \in \mathcal{I}^{-1}$ and $\widehat{r} \in R$. Therefore,

$$w_i y = w_i y(\widehat{w}z + \widehat{r}q) = w_i z\widehat{w}y + w_i y\widehat{r}q = a_i\widehat{w}y \bmod qR ,$$

where we have used the definition of $a_i = w_i z \bmod qR$ and the fact that $w_i y \in R$, since $w_i \in \mathcal{I}^{-1}$ and $y \in \mathcal{I}$.

[10] We can try many different parameters until we find one that happens to work, using the 2^n-approximation to $\lambda_n(\mathcal{I}^{-1})$ given by the LLL algorithm to guarantee that we need only try $O(n)$ different parameters.

Finally, we note that $\widehat{w}y \in R$ is independent of i, so we simply define $s := \widehat{w}y \bmod qR$, and we notice that the (a_i, b_i) are valid Ring-LWE samples with secret s. We then simply note that $zs = y \bmod q\mathcal{I}$, so that we can recover $y \bmod q\mathcal{I}$ from s. So, our Ring-LWE oracle allows us to find $y \bmod q\mathcal{I}$. To finish, we need to find y given t and $y \bmod q\mathcal{I}$, which is equivalent to solving IdealBDD with parameter $\alpha' = \alpha/q < 2^{-n}$. This can be solved efficiently, e.g., by rounding the coordinates of the target in an LLL-reduced basis.

A note on the error

If we had not chosen the rather extreme approximation factor $\alpha = n^{-\omega(1)}$, then we would have had to study the error $w_i e$ more carefully. In fact, this error is not really "average case" in the sense that it depends fundamentally on e. For example, recall that e is a polynomial and suppose that it has some root, perhaps over the integers. Then, clearly, $w_i e$ must have the same root, so it lies in a subspace and cannot be distributed like a spherical Gaussian, regardless of the distribution of w_i. In general, the distribution of $w_i e$ will be close to a discrete Gaussian over R with some covariance matrix that depends on e. (In the canonical embedding, which we have avoided defining, each coordinate will be an independent Gaussian whose parameter is proportional to the corresponding coordinate of e.) I.e., we will *not* get the same error distribution regardless of the input IdealBDD instance.

Lyubashevsky, Peikert, and Regev offer two solutions for this [**?**]. The first is to solve this problem "in the lattice regime," by rerandomizing the target t. I.e., we reduce IdealBDD to a variant of IdealBDD in which the target is sampled from a Gaussian distribution. This still does not quite allow us to reduce to Ring-LWE with fixed spherical Gaussian error. Instead, this gives us a distribution over covariance matrices such that Ring-LWE with error given by the Gaussian whose covariance is sampled from this distribution is worst-case hard with high probability. This is enough to build cryptography, but naturally this is not used in practice.

The second solution in [**?**] is to use noise flooding in the RingLWE instance itself. I.e., we take $\widetilde{e}$ to be a spherical Gaussian that is significantly larger than $\|w_i e\|$, just like we did. However, to avoid having to take $\alpha = n^{-\omega(1)}$, we allow the noise parameter to depend on the number of samples ℓ in the Ring-LWE instance. We then get a distribution $w_i e + \widetilde{e}$ that is *not* within negligible statistical distance of a spherical Gaussian, but the two distributions will still have large overlap, even over ℓ samples.

10.7 NTRU

Finally, we mention a different elegant way to build public-key encryption using polynomial rings, such as R_q, the NTRU encryption scheme, due to Hoffstein, Pipher, and Silverman [**?**]. Historically, NTRU predates LWE by nearly a decade and Ring-LWE by about 15 years. As far as we know, it is more-or-less as secure as Ring-LWE-based schemes for most reasonable parameter settings. However, unlike Ring-LWE-based schemes, NTRU comes with no worst-case hardness guarantee. We present it here because (1) it is pretty; (2)one of the relatively few concrete assumptions known to imply public-key cryptography; and (3) people who work in lattice-based cryptography should know what NTRU is.

As before, we work over $R := \mathbb{Z}[x]/(x^n + 1)$ for power-of-two n with $R_q := R/(qR)$ for some modulus $q = \mathrm{poly}(n)$. A "typical" element in R is invertible modulo $3R$ (i.e., the polynomial $x^n + 1$

does not have low-degree factors modulo 3),[11] and we may, e.g., take q to be prime to guarantee the same modulo qR. (NTRU can be defined over any polynomial ring, and it is often actually defined over $\mathbb{Z}[x]/(x^n - 1)$. This causes some annoying issues related to those that we observed in the context of Ring-SIS. They can be overcome, but we ignore this issue by using our preferred ring.)

- **Key generation:** Sample two short polynomials $g, f \in R$. E.g., sample them uniformly at random from $R_{\{0,1\}}$. If f is not invertible modulo both qR and $3R$, we resample it. Otherwise, we denote these respective inverses by f_q^{-1} and f_3^{-1}. The public key is $h := gf_q^{-1} \bmod qR$, and the private key is f, g.

- **Encryption:** Let $m \in R_{\{-1,0,1\}}$ be some ternary message. The encryption algorithm computes the ciphertext $c := hr + 3e + m \bmod qR$, where r and e are some random short polynomials.

- **Decryption:** Given a ciphertext c, we compute $fc = 3(fe + rg) + fm \bmod qR$. As long as q is sufficiently large, this element $3(fe + rg) + fm$ should have small coefficients relative to q. I.e., by choosing our representative of $3(fe + rg) + fm \bmod qR$ to have coefficients in the interval $(-q/2, q/2]$, we can actually recover $3(fe + rg) + fm \in R$, not just its coset in R_q. This allows us to reduce the result modulo $3R$ to recover fm. Finally, we multiply by f_3^{-1} to find m, which is uniquely determined by its coset modulo $3R$.

The security of NTRU is typically proven under the assumption that the public key h is indistinguishable from random. However, there is no known reduction from a more standard computational problem to the problem of distinguishing h from random. For most choices of parameters, however, our best attack on NTRU is a lattice attack that searches for a short vector in the so-called NTRU lattice, spanned by the basis

$$\begin{pmatrix} I_n & 0 \\ \mathrm{Rot}(\boldsymbol{h}) & qI_n \end{pmatrix} , \in \mathbb{Z}^{2n \times 2n}$$

where

$$\mathrm{Rot}\begin{pmatrix} h_1 \\ h_2 \\ \vdots \\ h_n \end{pmatrix} := \begin{pmatrix} h_1 & -h_n & -h_{n-1} & \cdots & -h_2 \\ h_2 & h_1 & -h_n & \cdots & -h_3 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ h_{n-1} & h_{n-2} & h_{n-3} & \cdots & -h_n \\ h_n & h_{n-1} & h_{n-2} & \cdots & h_1 \end{pmatrix} ,$$

as in the previous lecture, and $\boldsymbol{h}$ is the coefficient vector of the public key h. Notice that the NTRU lattice contains the short vector $(\boldsymbol{f}, \boldsymbol{g}) \in \mathbb{Z}^{2n}$ corresponding to the secret key. Indeed, any short enough vector in this lattice can be used to break the NTRU encryption scheme.

[11] It's factorization into irreducible polynomials is $x^n + 1 = (x^{n/2} + x^{n/4} - 1)(x^{n/2} - x^{n/4} - 1)$ modulo 3. The fact that these polynomials are irreducible is equivalent to saying that a finite field of characteristic 3 contains a primitive $2n$th root of unity if and only if it has size 3^m for m divisible by $n/2$, i.e., that $2n$ divides $3^m - 1$ if and only if $n/2$ divides m (since the multiplicative group of a finite field is cyclic). I was frustrated by my inability to find a nice enough proof of this, so I asked on Math StackExchange and got some very nice answers [?]—three very nice proof as of the last time I checked, as well as my own rather clunky proof.

CHAPTER 11

Quantum Computing and Lattices

Quantum computing and lattices have had a close relationship ever since the work of Regev in 2004 [**?**] which showed a connection between the unique shortest vector problem (which we now know is equivalent to bounded distance decoding) to the hidden subgroup problem on dihedral groups. Since then, the relationship has been very productive.

On the one hand, lattices give us one of the most prominent ways to do "post-quantum" cryptography. For this to be meaningful, we need to be reasonably confident that there are no "fast" quantum algorithms for LWE/SIS. Many have tried and failed.

We will see one of the most important such attempts, Kuperberg's algorithm for the dihedral hidden subgroup problem and its connnection to LWE. Another example is the recent advances in quantum algorithms for the principal ideal problem (PIP) which we will not describe today.

On the other hand, lattices have given us ways to solve fundamental problems in cryptography and quantum computing including generating a verifiable stream of truly random coins, designing classical protocols which check that a quantum computer is doing its job correctly, and designing a quantum money scheme. (See recent works of Mahadev, of Brakerski, Christiano, Mahadev, Vazirani and Vidick, and of Zhandry.) We will unfortunately not have time to delve into any of these today, but please see the course website for pointers.

11.1 A Quantum Computing Primer

States. A (pure) quantum state is a *unit vector* in the Hilbert space $\mathbb{C}^N$ for some N. Here, N is the universe under consideration. For example, for a one qubit system, $N = 2$; for two qubits, $N = 4$; and for n qubits, $N = 2^n$. For every $x \in [N] = \{0,1\}^n$, we will denote the elementary states as

$$|x\rangle = \begin{pmatrix} 0 \\ 0 \\ \dots \\ 1 \\ 0 \\ \dots \\ 0 \end{pmatrix}$$

with a 1 in the x^{th} location and 0 everywhere else. In particular, when $n = 1$, we have

$$|0\rangle = \begin{pmatrix} 1 \\ 0 \end{pmatrix} \qquad \text{and} \qquad |1\rangle = \begin{pmatrix} 0 \\ 1 \end{pmatrix}$$

The vectors $|x\rangle$ form an orthonormal basis for $\mathbb{C}^N$. (Indeed, as we will see later, any orthonormal basis is just as good). A general (pure) quantum state can be written as

$$|\Psi\rangle = \sum_{x \in [N]} \alpha_x |x\rangle$$

where $\alpha_x \in \mathbb{C}$ such that $\sum_{x \in [N]} |\alpha_x|^2 = 1$. For example,

$$|+\rangle = \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle) = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 \\ 1 \end{pmatrix} \qquad \text{and} \qquad |-\rangle = \frac{1}{\sqrt{2}}(|0\rangle - |1\rangle) = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 \\ -1 \end{pmatrix}$$

The joint state of n qubits lives in the tensor product of the corresponding Hilbert spaces. If $|x\rangle \in \mathbb{C}^{N_1}$ and $|y\rangle \in \mathbb{C}^{N_2}$ then the joint state, denoted $|x, y\rangle \in \mathbb{C}^{N_1 N_2}$.

Operations. Legal operations on qubits have to turn unit vectors into unit vectors; therefore, they are unitary matrices $U \in \mathbb{C}^{N \times N}$. That is,

$$U^\dagger U = I$$

where $U^\dagger$ is the conjugate transpose of U (In case U is a real matrix, this is simply the transpose of U.)

- For a single qubit, the X gate is defined by

  $$\mathsf{X} = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}$$

  This turns $|0\rangle$ into $|1\rangle$ and vice versa. The Z gate is defined by

  $$\mathsf{Z} = \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix}$$

  This keeps $|0\rangle$ the same and turns $|1\rangle$ into $-|1\rangle$.

- For two qubits, the controlled not (CNOT) gate turns $|a, b\rangle$ into $|a, a \oplus b\rangle$. That is, if $a = 0$, it leaves everything the same, but if $a = 1$, it flips the second bit.

  This is defined by

  $$\mathsf{CNOT} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix}$$

- The quantum Fourier transform QFT_n over $\mathbb{Z}_N := \mathbb{Z}_{2^n}$ is defined by the $N \times N$ unitary matrix where the $(a,b)^{th}$ entry is $e^{2\pi i ab/N} := \omega_N^{ab}$ where ω_N denotes the primitive N^{th} root of unity. (The indices run from 0 to $N-1$.)

  For example, QFT_1 is defined by

$$\begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix}$$

  and QFT_2 is defined by

$$\frac{1}{2}\begin{pmatrix} 1 & 1 & 1 & 1 \\ 1 & i & -1 & -i \\ 1 & -1 & 1 & -1 \\ 1 & -i & -1 & i \end{pmatrix}$$

We care about unitaries on n qubits that can be implemented using a quantum circuit with $\mathsf{poly}(n)$ *gates* that each act on a constant number of qubits. We will assert, but not prove, that QFT_n can be implemented with a poly-size quantum circuit.

Measurement. When measuring a qubit $|v\rangle$ in an orthonormal basis $(|b_0\rangle, |b_1\rangle)$, you get 0 with probability $|\langle v, b_0\rangle|^2$ and 1 with probability $|\langle v, b_1\rangle|^2$. For example, let

$$|v\rangle = \mathsf{cos}(\theta)|0\rangle + \mathsf{sin}(\theta)|1\rangle$$

Measuring it in the $|0\rangle, |1\rangle$ basis gives us 0 with probability $\mathsf{cos}^2(\theta)$ and 1 with probability $\mathsf{sin}^2(\theta)$. Measuring it in the $|+\rangle, |-\rangle$ basis gives us 0 (+) with probability $\mathsf{cos}^2(\theta - \pi/4)$ and 1 ($-$) with probability $\mathsf{sin}^2(\theta - \pi/4)$.

Measuring collapses the state to one of the two basis vectors.

Measuring one qubit in a multi-qubit system does something very similar. Let us show just an example. Let $|v\rangle = \frac{1}{2}(|00\rangle + |01\rangle + |10\rangle - |11\rangle)$ be a state (this is a so-called Bell state.) Measuring the second qubit in the standard basis $(|0\rangle, |1\rangle)$ gives us a uniform bit, and the state collapses to either $\frac{1}{\sqrt{2}}(|0\rangle + |1\rangle) := |+\rangle$ or $\frac{1}{\sqrt{2}}(|0\rangle - |1\rangle) := |-\rangle$.

What happens when you measure the second qubit in the Hadamard basis, that is, $|+\rangle, |-\rangle$?

11.2 Dihedral Hidden Subgroup Problem and LWE

The Hidden Subgroup Problem

For a finite set S, let $|S\rangle$ denote the state

$$|S\rangle = \frac{1}{\sqrt{|S|}} \sum_{s \in S} |s\rangle$$

In the hidden subgroup problem, we have a known finite group G (presented explicitly), a finite set S, and (black-box access to) a function $f : G \to S$ which is constant on all (right) cosets of a *hidden* subgroup $H \leq G$. The goal is to discover H, say as a set of its generators.

Nearly all quantum algorithms that achieve superpolynomial speedup do so by solving a hidden subgroup problem, typically over Abelian groups). Examples are Simon's algorithm over Z_2^n and Shor's algorithm over Z_N where N is a composite number that we wish to factor.

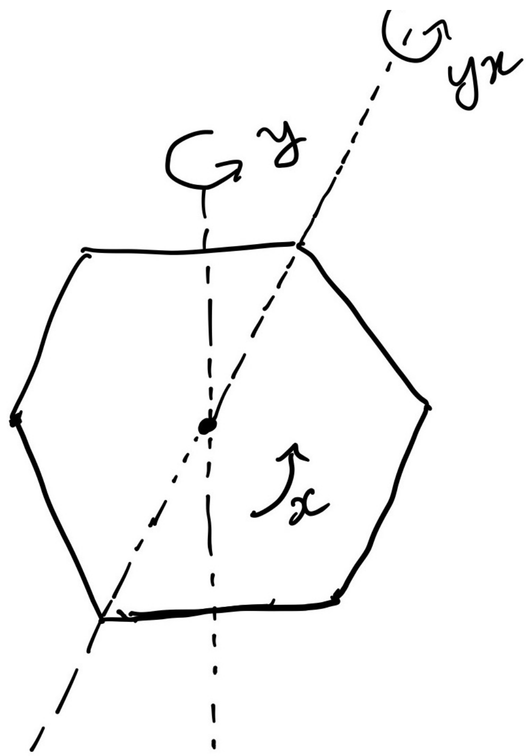


Figure 11.1: Dihedral group D_6 as a group of symmetries of a regular hexagon. y denotes reflection across the vertical axis; x denotes a counter-clockwise rotation; and yx (as the reader should check) is reflection across the slanted line.

Dihedral Group. This is a (non-Abelian) group of order $2N$ generated by two elements x and y such that

$$x^N = 1,\ y^2 = 1,\ \text{and } yxy = x^{-1}$$

We think of the dihedral group as the group of symmetries of a regular N-gon; x is a rotation and y is a reflection along the vertical axis. See the accompanying figure for an illustration.

The subgroups of D_N are either:

1. cyclic subgroups of Z_N, consisting only of rotations;

2. **subgroups of order 2 generated by some reflection** yx^s; or

3. subgroups generated by a reflection and a copy of a subgroup of Z_N.

(1) is Abelian and therefore easy. (3) can be shown to be essentially as easy as (2). If H contains a copy of a cycle C, one can reduce the problem to an HSP where $H' = H/C$ on the group $G' = D_N/C \equiv D_{N'}$. So, we will focus on (2).

As we will see, dihedral HSP is very closely related to LWE.

Dihedral HSP

Let H be (the order-2 subgroup) generated by an element $y^b x^s$ in D_N which we will denote by the pair $(b, s) \in \mathsf{Z}_2 \times \mathsf{Z}_N$. Note that with this notation, the group law can be written as

$$(b, s) \odot (b', s') = (b \oplus b', (-1)^{b'} s + s')$$

where $\oplus$ denotes mod-2 addition. Note that in the non-trivial case we are considering, $b = 1$. Finally, note that order of operation matters here as the group is non-Abelian.

The (right) cosets of H are

$$Ha := \{(0, a), (1, s + a)\}$$

for all $a \in \mathsf{Z}_N$.

Generating coset states (Coset sampling). First, create a superposition $|G\rangle$ over all elements of $G := D_N$.

$$\sum_{b' \in \{0,1\}, s' \in \mathbb{Z}_N} |b', s'\rangle$$

Tensor this with the singleton state to produce

$$\sum_{b' \in \{0,1\}, s' \in \mathbb{Z}_N} |b', s', 0\rangle$$

Compute the function $f : G \to S$ in the description of the hidden subgroup problem coherently in superposition.

$$\sum_{b' \in \{0,1\}, s' \in \mathbb{Z}_N} |b', s', f(b', s')\rangle$$

Measure the third register to get the state

$$\sum_{(b', s') \in Ha} |b', s'\rangle = |0, a\rangle + |1, s + a\rangle$$

for some (unknown) $a \in \mathsf{Z}_N$.

From LWE to (Robust) Dihedral HSP

We start by showing the relevance of the dihedral HSP by showing how to reduce LWE to it. In fact, we will crucially need a stronger version of dihedral HSP which we call robust dihedral HSP. In this variant, the function f can make an error with some small probability ε so that the coset states are "correct" with probability $1 - \varepsilon$, and are a singleton state (which is not a valid coset state) with probability ε.

Jumping ahead, we remark that we will show a subexponential algorithm for dihedral HSP in a little bit. But the algorithm seems to be not particularly noise-tolerant. Rather frustratingly (or shall we say fortunately), this is what prevents us from using the reduction this section together with Kuperberg's algorithm to come up with a subexponential quantum algorithm for LWE!

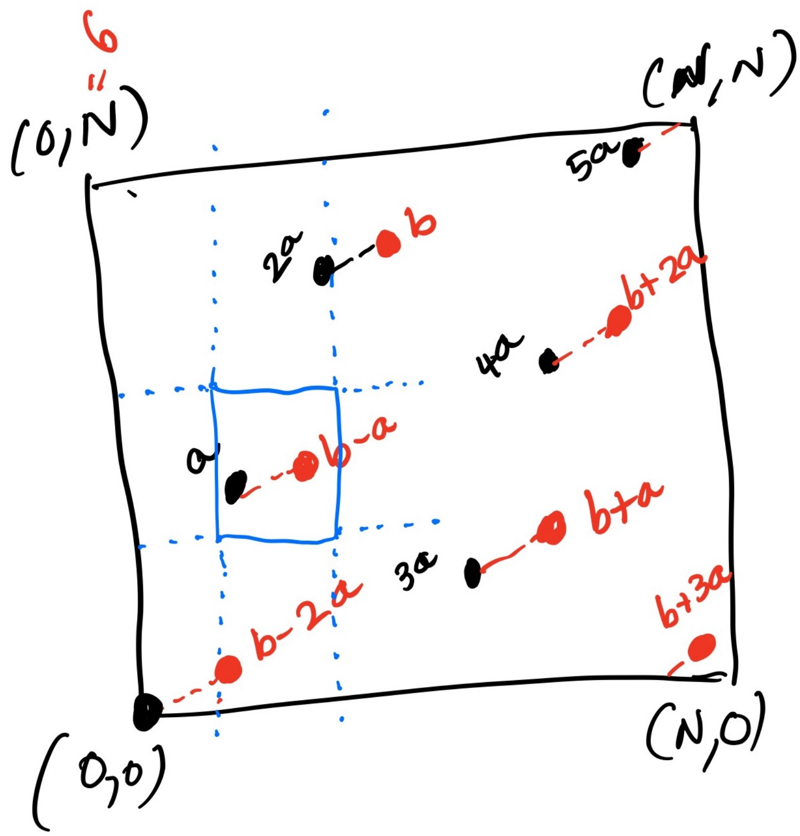
For simplicity, we will reduce from a one-dimensional version of LWE (which, for appropriate parameters, can be shown to be as hard as LWE itself) to dihedral HSP.

So, you are given

$$(\mathbf{a}, \mathbf{b} := s\mathbf{a} + \mathbf{e}) \in \mathsf{Z}_N^{1\times m} \times \mathsf{Z}_N^{1\times m}$$

for some N (which you should think of as exponential in the security parameter λ) and m (which you should think of as polynomial in λ). Your goal is to recover $s \in \mathsf{Z}_N$.

Partition the space into cubes of side-length ℓ. We will set ℓ so that $\ell\sqrt{m} < \lambda_1(L) \approx O(q)$, but so that $\ell \gg ||\mathbf{e}||$. (Note that this already places a limit on the LWE error for which we can solve it, i.e., $||\mathbf{e}|| \ll q/\sqrt{m}$.) Let ϕ be the function that maps a point in Z_N^m to its associated cube.



Create the state

$$|0\rangle \sum_{t\in\mathsf{Z}_N} |t, \phi(t\mathbf{a})\rangle + |1\rangle \sum_{t\in\mathsf{Z}_N} |t, \phi(\mathbf{b} + t\mathbf{a})\rangle = |0\rangle \sum_{t\in\mathsf{Z}_N} |t, \phi(t\mathbf{a})\rangle + |1\rangle \sum_{t\in\mathsf{Z}_N} |t - s, \phi(\mathbf{e} + t\mathbf{a})\rangle$$

Measure the second register which will give us the name of a subcube. The rest of the state will either collapse to a singleton (when there is either a lattice point or a shifted lattice point in the cube, but not both) or a superposition of two points (when there is both a lattice point and a shifted lattice point in the cube). It is easy to check that the way we set up parameters, there will never be two lattice points (resp. two shifted lattice points in the same cube).

So, in the good case, we get

$$|0\rangle|t\rangle + |1\rangle|t - s\rangle$$

Starting from our LWE sample, we can produce as many of these states as we like. As we saw a few minutes ago, these are precisely the coset states of the dihedral HSP. So, any algorithm that solves the dihedral HSP by coset sampling will give us an algorithm for LWE.

The one wrinkle in this reduction is that sometimes we get singleton states which are not valid coset states for the dihedral HSP (and we never know when we got those, so we can't throw them away.) How often do we get singleton states?

I will leave it to you to check that this happens with probability roughly $||\mathbf{e}|| \cdot n^{1.5}/N$.

Kuperberg's Algorithm for Dihedral HSP

We will show the algorithm for $N = 2^n$. This can be generalized to any N with essentially the same complexity.

Let H be (the order-2 subgroup) generated by an element $y^b x^s$ in D_N which we will denote by the pair $(b, s) \in \mathsf{Z}_2 \times \mathsf{Z}_N$. Note that with this notation, the group law can be written as

$$(b, s) \odot (b', s') = (b \oplus b', (-1)^{b'} s + s')$$

where $\oplus$ denotes mod-2 addition. Note that in the non-trivial case we are considering, $b = 1$. Finally, note that order of operation matters here as the group is non-Abelian.

The (right) cosets of H are

$$Ha := \{(0, a), (1, s + a)\}$$

for all $a \in \mathsf{Z}_N$.

Generating coset states. First, create a superposition $|G\rangle$ over all elements of $G := D_N$.

$$\sum_{b' \in \{0,1\}, s' \in \mathbb{Z}_N} |b', s'\rangle$$

Tensor this with the singleton state to produce

$$\sum_{b' \in \{0,1\}, s' \in \mathbb{Z}_N} |b', s', 0\rangle$$

Compute the function $f : G \to S$ in the description of the hidden subgroup problem coherently in superposition.

$$\sum_{b' \in \{0,1\}, s' \in \mathbb{Z}_N} |b', s', f(b', s')\rangle$$

Measure the third register to get the state

$$\sum_{(b', s') \in Ha} |b', s'\rangle = |0, a\rangle + |1, s + a\rangle$$

for some (unknown) $a \in \mathsf{Z}_N$.

Quantum Fourier Transform over Z_N. QFT gives us the state

$$\begin{aligned}
|0\rangle \sum_{x\in\mathsf{Z}_N} \omega_N^{ax}|x\rangle + |1\rangle \sum_{x\in\mathsf{Z}_N} \omega_N^{x(s+a)}|x\rangle \\
= \eta \cdot \left(\sum_{x\in\mathsf{Z}_N} |0,x\rangle + \omega_N^{xs}|1,x\rangle \right)
\end{aligned}$$

where η is a global phase (which no measurement can distinguish and can be ignored.)

Measure the second register to get a value $x \in \mathsf{Z}_N$ and the state

$$|\Psi_x\rangle := |0\rangle + \omega_N^{xs}|1\rangle$$

This can be repeated many times to generate a random $x \in \mathsf{Z}_N$ together with the state Ψ_x.

Kuperberg Sieve. We now have many copies $(x, |\Psi_x\rangle)$ and wish to find s, therefore solving the HSP. We will now focus on finding *a single bit* of s, namely its least significant bit. This can later be iterated with every single bit of s to recover the entire value.

That is, our goal will be to somehow produce the state

$$|0\rangle + (-1)^{s \bmod 2}|1\rangle$$

from which $s \bmod 2$ can be recovered by measuring in the $|+\rangle, |-\rangle$ basis.

What we will next do should remind you of an algorithm we have already seen in class. (Can you remember which one?)

We produce Q such states. We note that if we take two such states

$$|00\rangle + \omega_N^{x_1 s}|10\rangle + \omega_N^{x_2 s}|01\rangle + \omega_N^{(x_1+x_2)s}|11\rangle$$

and apply a CNOT operator (with the first qubit as the control), we get

$$|00\rangle + \omega_N^{x_1 s}|11\rangle + \omega_N^{x_2 s}|01\rangle + \omega_N^{(x_1+x_2)s}|10\rangle$$

Measure the second qubit to get

$$|0\rangle + \omega_N^{(x_1+x_2)s}|1\rangle \quad \text{or} \quad |0\rangle + \omega_N^{(x_1-x_2)s}|1\rangle$$

where the former happens if the measurement resulted in 0 and the latter if it resulted in 1.

Here is the consequence. Assume that k least significant bits of x_1 and x_2 were the same to begin with. This gives us a procedure to take two qubits and with probability $1/2$ produce a single qubit $|0\rangle + \omega_N^{xs}|1\rangle$ where k of the least significant bits of x are 0. We can continue this procedure to "clear out" more and more of the LSBs and eventually keep just the MSB of x. In this case, it is easy to check that the resulting state is $|0\rangle + \omega_N^{s \bmod 2}|1\rangle$ if the MSB of x is 1.

Each step "consumes" on average four qubits to produce a better qubit.

- If k is too small, we need many qubits to get to the end, roughly $4^{n/k}$.
- If k is too large, we may not be able to "pair up" the qubits with friends so that the least significant bits of the corresponding x match. Roughly speaking, we need about 2^k qubits to make sure, by the coupon collector bound, that *most qubits have friends*.

Fortunately, there is a point in the tradeoff space, and as the calculation suggests, the right thing to do is set $2n/k \approx k$ or $k \approx \sqrt{2n}$.

Thus, we start with producing $Q_0 = (k + n/4k) \cdot 2^k \cdot 4^{n/k}$ qubits

$$(x, |\Psi_x\rangle)$$

Each step potentially loses 2^k qubits which cannot be paired, and roughly a factor 4 because of the sieving step. So, we get in expectation $Q_1 = (Q_0 - 2^k)/4$ qubits. At the end,

$$Q_{n/k} = Q_0 \cdot 4^{-n/k} - 2^k \cdot (n/4k) \approx k2^k \gg O(1)$$

qubits remain. Since none of the operations "looked at" the MSB of the x, the resulting bits will have $MSB(x) = 0$ or 1 nearly equiprobably. In the event that $MSB(x) = 1$, we obtain the desired outcome, i.e., the LSB of s.

Setting the Parameters. Balancing k and n/k gives us a roughly

$$2^{O(\sqrt{n})} = 2^{O(\sqrt{\log N})}$$

time quantum algorithm. The memory consumption is nearly the same as time, but this has been improved subsequently by Kuperberg to use $2^{O(\sqrt{\log N})}$ time and *classical space* but only $O(\log N)$ quantum memory.

Bibliography

[BD20] Zvika Brakerski and Nico Dttling. Hardness of lwe on general entropic distributions. Cryptology ePrint Archive, Report 2020/119, 2020. https://eprint.iacr.org/2020/119.

[BGG$^+$14] Dan Boneh, Craig Gentry, Sergey Gorbunov, Shai Halevi, Valeria Nikolaenko, Gil Segev, Vinod Vaikuntanathan, and Dhinakaran Vinayagamurthy. Fully key-homomorphic encryption, arithmetic circuit ABE and compact garbled circuits. In Phong Q. Nguyen and Elisabeth Oswald, editors, *Advances in Cryptology - EUROCRYPT 2014 - 33rd Annual International Conference on the Theory and Applications of Cryptographic Techniques, Copenhagen, Denmark, May 11-15, 2014. Proceedings*, volume 8441 of *Lecture Notes in Computer Science*, pages 533–556. Springer, 2014.

[BLP$^+$13] Zvika Brakerski, Adeline Langlois, Chris Peikert, Oded Regev, and Damien Stehlé. Classical hardness of learning with errors. In Dan Boneh, Tim Roughgarden, and Joan Feigenbaum, editors, *Symposium on Theory of Computing Conference, STOC'13, Palo Alto, CA, USA, June 1-4, 2013*, pages 575–584. ACM, 2013.

[BPR12] Abhishek Banerjee, Chris Peikert, and Alon Rosen. Pseudorandom functions and lattices. In David Pointcheval and Thomas Johansson, editors, *Advances in Cryptology - EUROCRYPT 2012 - 31st Annual International Conference on the Theory and Applications of Cryptographic Techniques, Cambridge, UK, April 15-19, 2012. Proceedings*, volume 7237 of *Lecture Notes in Computer Science*, pages 719–737. Springer, 2012.

[BV11] Zvika Brakerski and Vinod Vaikuntanathan. Efficient fully homomorphic encryption from (standard) LWE. In Rafail Ostrovsky, editor, *IEEE 52nd Annual Symposium on Foundations of Computer Science, FOCS 2011, Palm Springs, CA, USA, October 22-25, 2011*, pages 97–106. IEEE Computer Society, 2011.

[BV15] Zvika Brakerski and Vinod Vaikuntanathan. Constrained key-homomorphic prfs from standard lattice assumptions - or: How to secretly embed a circuit in your PRF. In Yevgeniy Dodis and Jesper Buus Nielsen, editors, *Theory of Cryptography - 12th Theory of Cryptography Conference, TCC 2015, Warsaw, Poland, March 23-25, 2015, Proceedings, Part II*, volume 9015 of *Lecture Notes in Computer Science*, pages 1–30. Springer, 2015.

[CHK04] Ran Canetti, Shai Halevi, and Jonathan Katz. Chosen-ciphertext security from identity-based encryption. In Christian Cachin and Jan Camenisch, editors, *Advances in Cryptology - EUROCRYPT 2004, International Conference on the Theory and Applications* *of Cryptographic Techniques, Interlaken, Switzerland, May 2-6, 2004, Proceedings*, volume 3027 of *Lecture Notes in Computer Science*, pages 207–222. Springer, 2004.

[GHM+19] Sanjam Garg, Mohammad Hajiabadi, Mohammad Mahmoody, Ahmadreza Rahimi, and Sruthi Sekar. Registration-based encryption from standard assumptions. In Dongdai Lin and Kazue Sako, editors, *Public-Key Cryptography - PKC 2019 - 22nd IACR International Conference on Practice and Theory of Public-Key Cryptography, Beijing, China, April 14-17, 2019, Proceedings, Part II*, volume 11443 of *Lecture Notes in Computer Science*, pages 63–93. Springer, 2019.

[GHMR18] Sanjam Garg, Mohammad Hajiabadi, Mohammad Mahmoody, and Ahmadreza Rahimi. Registration-based encryption: Removing private-key generator from IBE. In Amos Beimel and Stefan Dziembowski, editors, *Theory of Cryptography - 16th International Conference, TCC 2018, Panaji, India, November 11-14, 2018, Proceedings, Part I*, volume 11239 of *Lecture Notes in Computer Science*, pages 689–718. Springer, 2018.

[GKPV10] Shafi Goldwasser, Yael Tauman Kalai, Chris Peikert, and Vinod Vaikuntanathan. Robustness of the learning with errors assumption. In Andrew Chi-Chih Yao, editor, *Innovations in Computer Science - ICS 2010, Tsinghua University, Beijing, China, January 5-7, 2010. Proceedings*, pages 230–240. Tsinghua University Press, 2010.

[GPV08] Craig Gentry, Chris Peikert, and Vinod Vaikuntanathan. Trapdoors for hard lattices and new cryptographic constructions. In Cynthia Dwork, editor, *Proceedings of the 40th Annual ACM Symposium on Theory of Computing, Victoria, British Columbia, Canada, May 17-20, 2008*, pages 197–206. ACM, 2008.

[GVW15] Sergey Gorbunov, Vinod Vaikuntanathan, and Hoeteck Wee. Predicate encryption for circuits from LWE. In Rosario Gennaro and Matthew Robshaw, editors, *Advances in Cryptology - CRYPTO 2015 - 35th Annual Cryptology Conference, Santa Barbara, CA, USA, August 16-20, 2015, Proceedings, Part II*, volume 9216 of *Lecture Notes in Computer Science*, pages 503–523. Springer, 2015.

[Kim20] Sam Kim. Key-homomorphic pseudorandom functions from lwe with a small modulus. Cryptology ePrint Archive, Report 2020/233, 2020. `https://eprint.iacr.org/2020/233`.

[Kle00] Philip N. Klein. Finding the closest lattice vector when it's unusually close. In David B. Shmoys, editor, *Proceedings of the Eleventh Annual ACM-SIAM Symposium on Discrete Algorithms, January 9-11, 2000, San Francisco, CA, USA*, pages 937–941. ACM/SIAM, 2000.

[Pei09] Chris Peikert. Public-key cryptosystems from the worst-case shortest vector problem: extended abstract. In Michael Mitzenmacher, editor, *Proceedings of the 41st Annual ACM Symposium on Theory of Computing, STOC 2009, Bethesda, MD, USA, May 31 - June 2, 2009*, pages 333–342. ACM, 2009.

[Reg09] Oded Regev. On lattices, learning with errors, random linear codes, and cryptography. *J. ACM*, 56(6):34:1–34:40, 2009.