Lattices, Learning with Errors and Post-Quantum Cryptography

Lecture Notes
# Contents

1. **The Learning with Errors Problem: Introduction and Basic Cryptography**
   - 1.1 Solving Systems of Linear Equations ........................................... 6
   - 1.2 Basic Theorems ............................................................................. 9
   - 1.3 Basic Cryptographic Applications .................................................. 9

2. **The Learning with Errors Problem: Algorithms** .................................. 16
   - 2.1 An algebraic Algorithm: Arora-Ge ................................................. 16
   - 2.2 A Combinatorial Algorithm: Blum-Kalai-Wasserman ....................... 18
   - 2.3 A Geometric (Suite of) Algorithm(s): Lattice Reduction ............... 19

3. **Worst-case to Average-case Reduction for SIS** .................................. 20
   - 3.1 Lattices and Minkowski’s Theorem .............................................. 20
   - 3.2 Lattice Smoothing ........................................................................ 21
   - 3.3 Worst-case to Average-case Reduction for SIS ............................... 27

4. **Worst-case to Average-case Reduction for LWE** ................................ 30
   - 4.1 Decision to Search Reduction for LWE ........................................... 30
   - 4.2 Bounded Distance Decoding and LWE ......................................... 36
   - 4.3 Discrete Gaussians ....................................................................... 37
   - 4.4 From (Worst-case) BDD to (Average-case) LWE ......................... 39
   - 4.5 From (Worst-case) SIVP to (Worst-case) BDD .............................. 41

5. **Pseudorandom Functions from Lattices** .......................................... 42
   - 5.1 Pseudorandom Generator from LWE .............................................. 42
   - 5.2 GGM Construction ....................................................................... 42
   - 5.3 BLMR13 Construction .................................................................. 43
   - 5.4 BP14 Construction ....................................................................... 46

6. **Trapdoors, Gaussian Sampling and Digital Signatures** ......................... 48
   - 6.1 Lattice Trapdoors ......................................................................... 48
   - 6.2 Trapdoor Sampling ....................................................................... 49
   - 6.3 Trapdoor Functions ....................................................................... 51
   - 6.4 Digital Signatures ......................................................................... 53
   - 6.5 Discrete Gaussian Sampling .......................................................... 54
7 Identity-Based Encryption and Friends

7.1 Identity-based Encryption ................................................. 56
7.2 Recap: GPV Signatures ................................................... 59
7.3 The Dual Regev Encryption Scheme ................................... 59
7.4 The GPV IBE Scheme ..................................................... 60
7.5 The CHKP IBE Scheme .................................................. 61
7.6 The ABB IBE Scheme .................................................... 63
7.7 Application: Chosen Ciphertext Secure Public-key Encryption .......... 64
7.8 Registration-based Encryption ........................................... 65

8 Encrypted Computation from Lattices

8.1 Fully Homomorphic Encryption ....................................... 66
8.2 The GSW Scheme ....................................................... 67
8.3 How to Add and Multiply (without errors) ............................. 67
8.4 How to Add and Multiply (without errors) ............................. 68
8.5 Bootstrapping to an FHE .................................................. 68
8.6 The Key Equation ........................................................ 69
8.7 Fully Homomorphic Signatures ......................................... 72
8.8 Attribute-based Encryption .............................................. 73
8.9 Constrained PRF ........................................................... 74

9 Constrained PRFs and Program Obfuscation

9.1 Constrained PRF ............................................................ 78
9.2 Private Constrained PRFs ................................................ 78
9.3 Private Constrained PRF: Construction ............................... 79
9.4 Program Obfuscation and Other Beasts ................................ 83
9.5 Lockable Obfuscation: An Application ................................ 85
9.6 Lockable Obfuscation: Construction ................................... 85

10 Ideal Lattices, Ring-SIS and Ring-LWE

11 Oblivious Transfer and Multiparty Computation

12 Zero Knowledge Proofs

13 Quantum Computing and Lattices
These notes are based on lectures in the course CS294: Lattices, Learning with Errors and Post-Quantum Cryptography at UC Berkeley in Spring 2020.
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 $s \in \mathbb{Z}_q^n$ given

$$\{(a_i, (a_i, s) + e_i) : s \leftarrow \mathbb{Z}_q^n, a_i \leftarrow \mathbb{Z}_q^n, e_i \leftarrow \chi\}_{i=1}^m$$ (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 $\mathbb{Z}_q$;

- $\chi$ 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 $\mathcal{D}$, $a \leftarrow \mathcal{S}$ denotes that $a$ is chosen uniformly at random from the (finite) set $\mathcal{S}$.

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

$$Ae = b \mod q$$ (1.2)

given $A \in \mathbb{Z}_q^{n \times m}$ and $b \in \mathbb{Z}_q^n$. 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 \( e \) 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, \( S = [-B \ldots B]^m \) 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.

- \( S = \text{Ball}_R \), 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 \( \text{SIS}(n, m, q, B) \) 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 \( e_i \in [-B, -B + 1, \ldots, B - 1, B] \).

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

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 \((B\text{-bounded})\) solutions are very likely to exist if \((2B + 1)^m \gg q^n\), or \( m = \Omega(n \log q \log B) \). 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 \( A \) and \( b \) 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 0, that is, \( A\mathbf{e} = 0 \pmod{q} \). This variant is worst-case total as long as \((B + 1)^m > q^n\). That is, for every instance \( A \) 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 \( m \ll n \log q/\log B \), one can show again that there are likely to be no \( B\)-bounded solutions for a uniformly random \( b \) and thus, we have to find a different, sensible, way to state this problem. To
do this, we first pick a $B$-bounded vector $e$ and compute $b$ as $Ae \mod q$. In a sense, we plant the solution $e$ inside $b$. The goal now is to recover $e$ (which is very likely to be unique) given $A$ and $b$. 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 $(A, y^T = s^T A + e^T)$, let $A^\perp \in \mathbb{Z}_q^{(m-n) \times m}$ be a full-rank set of vectors in the right-kernel of $A$. That is,

$$A^\perp \cdot A^T = 0 \mod q$$

Then,

$$b := A^\perp \cdot y = A^\perp \cdot (A^T s + e) = A^\perp \cdot e \mod q$$

so $(A^\perp, b)$ is an SIS instance $\text{SIS}(m-n, m, q, B)$ 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 $e$ is unique given $(A, y)$.

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 $(A, y^T := s^T A + e^T \mod q)$ and a uniformly random distribution. One can show, through a reduction that runs in $\text{poly}(q)$ 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 $A$, then there is a poly-time algorithm that solves the search LWE problem for a (possibly different and possibly larger) uniformly random matrix $A'$. 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. $A$, then it is immediate to solve decision-LWE w.r.t. $A$. Indeed, given a SIS solution $e$ such that $Ae = 0 \mod q$, and a vector $b^T$, compute $b^T e \mod q$. If $b$ is an LWE instance, then

$$b^T e = (s^T A + x^T) e = x^T e \mod q$$

which is a “small” number (as long as $x^T$ is small enough). On the other hand, if $b$ 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 $A$ is systematic, that is of the form $A = [A' || I]$ where $A' \in \mathbb{Z}_q^{n \times (m-n)}$.

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

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

The corresponding version of LWE is called short-secret LWE where both the entries of $s$ and that of $e$ are chosen from the error distribution $\chi$. 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 $\text{sslLWE}(n, m, q, \chi)$ to $\text{LWE}(n, m, q, \chi)$ and one from $\text{LWE}(n, m, q, \chi)$ to $\text{sslLWE}(n, m+n, q, \chi)$.

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 $\{H_n\}_{n \in \mathbb{N}}$ where each $H_n$ 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 $H_n$ that is secure under $\text{SIS}(n, m, q, B)$ where $n \log q > m \log(B + 1)$. Each hash function $h_A$ is parameterized by a matrix $A \in \mathbb{Z}_q^{n \times m}$, takes as input $e \in [0, \ldots, B]^m$ and outputs\[ h_A(e) = Ae \mod q \]A collision gives us $e, e' \in [0, \ldots, B]^m$ where $Ae = Ae' \mod q$ which in turn says that $A(e - e') = 0 \mod q$. Since each entry of $e - e'$ is in $[-B, \ldots, B]$, this gives us a solution to $\text{SIS}(n, m, q, B)$.

Private-Key Encryption

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

Correctness requires that for every $sk$ generated by $\text{Gen}$ and every $m \in M$,\[ \text{Dec}(sk, \text{Enc}(sk, m)) = m \]

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 $(m_L, m_R) \in M^2$ and outputs an encryption of $m_L$ (resp. $m_R$).

Private-Key Encryption from LWE.

- $\text{Gen}(1^\lambda)$: 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\[ sk := s \leftarrow \mathbb{Z}_q^n. \]

- $\text{Enc}(sk, m)$: We will work with the message space $M := \{0, 1\}$. Larger message spaces can be handled by encrypting each bit of the message independently. The ciphertext is\[ c := (a, b) := (a, s^T a + e + m\lfloor q/2 \rfloor \mod q) \]where $a \leftarrow \mathbb{Z}_q^n$ and $e \leftarrow \chi$ is chosen from the LWE error distribution.

- $\text{Dec}(sk, c = (a, b))$: Output 0 if\[ |b - s^T a \mod q| < q/4 \]and 1 otherwise.

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


Correctness and security are immediate and left as an exercise to the reader. We left the issue of how to pick \( n, q \) and \( \chi \) 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 \( 2^\lambda \) time” (this is the meaning of the security parameter \( \lambda \) 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 \( \text{Gen} \) outputs a public key \( \text{pk} \) as well as a private key \( \text{sk} \); and second, the encryption algorithm requires only the public key \( \text{pk} \) to encrypt. Security requires that a p.p.t. adversary which is given \( \text{pk} \) (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.

- **\( \text{Gen}(1^\lambda) \):** 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 \( \text{sk} \) be a random vector \( \text{sk} := s \leftarrow \chi^n \) is chosen from the error distribution and the public key is

  \[ \text{pk} := (A, y^T := s^T A + e^T) \in \mathbb{Z}_q^{n \times n} \times \mathbb{Z}_q^n \]

  where \( A \) is a uniformly random \( n \)-by-\( n \) matrix and \( e \leftarrow \chi^n \) is chosen from the error distribution.

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

  \[ c := (a, b) := (Ar + x, y^T r + x' + m\lfloor q/2 \rfloor \mod q) \]

  where \( r, x \leftarrow \chi^n \) and \( x' \leftarrow \chi \) are chosen from the LWE error distribution.

- **\( \text{Dec}(\text{sk}, c = (a, b)) \):** Output 0 if

  \[ |b - s^T a \mod q| < q/4 \]

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

Proof. For correctness, note that the decryption algorithm computes

$$b - s^T a \mod q = s^T x + e^T r + x'$$

whose absolute value, as long as $\text{Supp}(\chi) \subseteq (-\sqrt{q/4(2n+1)}, \sqrt{q/4(2n+1)})$ 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 $\text{Enc}(pk, m)$ where $m \in \{0, 1\}$.

Hybrid 1.m. Feed the adversary with a “fake” public key $\tilde{pk}$ computed as

$$\tilde{pk} = (A, y) \leftarrow Z_{q^n}^n \times Z_q^n$$

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

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

$$\tilde{\text{Enc}}(\tilde{pk}, m) = (a, b' + m\lfloor q/2 \rfloor \mod q)$$

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

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

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

$$\begin{pmatrix} a \\ b' \end{pmatrix} + \begin{pmatrix} 0 \\ m\lfloor q/2 \rfloor \end{pmatrix} \mod q$$

where $a \leftarrow Z_q^n$ and $b' \leftarrow Z_q$.

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 $(\#\text{bits in ciphertext})/(\#\text{bits in plaintext})$ 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).

- \textbf{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 \( sk := s \leftarrow \mathbb{Z}_q^n \) is chosen uniformly at random from \( \mathbb{Z}_q \) and the public key is

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

where \( A \) is a uniformly random \( n \)-by-\( m \) matrix and \( e \leftarrow \chi^n \) 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 \( A \) is somewhat larger than in LPR.

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

\[
c := (a, b) := (Ar, y^T r + m\lfloor q/2 \rfloor \mod q)
\]

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

Note the difference from LPR where the vector \( r \) 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.

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

\[
\lvert b - s^T a \mod q \rvert < q/4
\]

and 1 otherwise.

Decryption recovers \( m\lfloor q/2 \rfloor \) plus an error \( e^T r + x' \) 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 \( \text{Supp}(\chi) \subseteq (-q/4(m+1), q/4(m+1)) \).

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.
• **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 $sk := r \leftarrow \{0,1\}^m$ is chosen uniformly at random with 0 or 1 entries and the public key is

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

where $A$ 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”.*

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

$$c := (y^T, b) := (s^T A + e^T, s^T a + x' + m \lfloor q/2 \rfloor \mod q)$$

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

• **Dec**($sk, c = (y^T, b)$): Output 0 if

$$|b - y^T r \mod q| < q/4$$

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 $\chi$ 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].
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 \((a, b := a^T s + e)\) where \(e \in S \subseteq \mathbb{Z}_q\) as a polynomial equation

\[
f_{a,b}(s) = \prod_{x \in S} (b - a^T s - x) \mod q
\]

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

\[
\{f_{a_i,b_i}(s) = 0 \mod q\}_{i=1}^m
\]

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, \(s\) is a solution. When \(m\) is large enough, we can also show that \(s\) 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

\[
(b + a \cdot u) \cdot (b + a \cdot u - 1) = 0 \mod q
\]

16
Writing it out explicitly, we get
\[ b(b - 1) + (2b - 1)a \cdot u + a^2 \cdot u^2 = 0 \mod q \]

Linearizing this involves replacing \( u \) and \( u^2 \) by independent variables \( u_1 \) and \( u_2 \) giving us
\[ p(a) = b(b - 1) + (2b - 1)a \cdot u_1 + a^2 \cdot u_2 = 0 \mod q \]  \( (2.1) \)

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 coefficient 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 \( (2.1) \). We get
\[ p'(a) = e(e - 1) + (2e - 1)(s + u_1) \cdot a + (u_2 + 2su_1 + s^2) \cdot a^2 \]
\[ = (2e - 1)(s + u_1) \cdot a + (u_2 + 2su_1 + s^2) \cdot a^2 = 0 \mod q \]

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 \( \chi \) is the discrete Gaussian distribution. Let’s now see what this does to \( \text{LWE}(n, m, q, \chi) \) where \( \chi \) 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 \( m > \binom{n}{k,s} \) 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 $x_{i,j}$ of the columns of $A$ that sum up to a fixed vector, say the unit vectors $u_i$, that is $Ax_{i,j} = u_i \mod q$. Once we find such vectors, we compute

$$b^T x_{i,j} = (s^T A + e^T)x_{i,j} = s_i + e^T x_{i,j} \mod q$$

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

In another variant, we find small-norm $x_{i,j}$ such that $Ax_{i,j} = 2^j e_i \mod q$. Upon multiplying with $b$ as before, we get

$$b^T x_{i,j} = (s^T A + e^T)x_{i,j} = s_i 2^j + e^T x_{i,j} \mod q$$

As long as $|e^T x_{i,j}| \ll q$, this allows us to "decode" $s_i$ with many fewer copies, essentially $O(\log q)$ of them, using the following decoding algorithm: use $s_i 2^\lceil \log(q/2) \rceil + e^T x_{i,j}$ 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 $A$ into $\alpha$ groups of size $\beta := n/\alpha$ each.

- For each column $a_i$ of $A$ we put it into one of $q^\beta$ buckets depending on what $a_i[1\ldots\beta]$ is.
- Notice that the difference of any two vectors, one in bucket labeled $w \in \mathbb{Z}_q^\beta$ and the other in bucket labeled $w - v \in \mathbb{Z}_q^\beta$, starts with $v$ in the first $\beta$ positions.
- This gives us many vectors whose first $\beta$ 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 $i \in [1\ldots\alpha]$.

The result is a linear combination with Hamming weight $2^\alpha$ of the columns of $A$ which sum to any given target vector. The process needs $q^\beta$ 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 $A$ 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
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 Kuper-
berg’s sub-exponential time quantum algorithm for the dihedral hidden subgroup problem (Kuper-
berg) 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 

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

and

\[
\mathcal{L}_y := \{s^T A : s \in \mathbb{Z}_q^n \} \oplus \mathbb{Z}_q^n \oplus \{0, y\}
\]

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

Let’s look at the case where \( \chi \) is a \( B \)-bounded distribution. We argue that:

- \( \mathcal{L}_y \) has a short vector, in fact a vector of \( \ell_2 \) norm \( \tilde{O}(B) \) (where \( \tilde{O} \) hides \( \text{poly}(m) \) factors) since \( e \in \mathcal{L}_y \).
- \( \mathcal{L} \) does not have any short vectors. The shortest vector of \( \mathcal{L} \) has \( \ell_2 \)-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 \( \mathcal{L}_y \) 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 \( e \). 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 \( \text{poly}(m) \), 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.
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 $\mathcal{L}(B)$ and its fundamental parallelepiped as follows:

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

As additive groups, $\mathcal{P}(B) = \mathbb{R}^n / \mathcal{L}(B)$. (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 parallelepiped. 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 $B$.

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,

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

where $B(\mathbf{0}, r)$ 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 $\lambda_i(\mathcal{L})$:

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

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

**Computational Problems.** The $\gamma$-approximate shortest vector problem (SVP) is to find a non-zero vector $v \in L(B)$ given a basis $B$ such that $\|v\| \leq \gamma \cdot \lambda_1(L)$. The $\gamma$-approximate shortest independent vectors problem (SIVP) is to find $n$ linearly independent vectors $v_1, \ldots, v_n$ such that $\|v_i\| \leq \gamma \cdot \lambda_n(L)$.

### 3.2 Lattice Smoothing

**Lattice Duality**

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

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

Indeed, each dual lattice vector $x$ corresponds to a linear function $\phi_x : L \to \mathbb{Z}$ 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 $L = L(B)$ for a basis matrix $B \in \mathbb{R}^{n \times n}$, then $L^*$ is generated by the columns of $B^{-T}$, its transposed inverse. Indeed, the pairwise inner products of the basis vectors and the dual basis vectors is captured in the matrix $B \cdot (B^{-T})^T = I$.

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 $L$, $\lambda_1(L^*) \cdot \lambda_1(L) \leq n$.

**Proof.** We know from Minkowski that

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

Multiplying the two, we get

$$\lambda_1(L) \cdot \lambda_1(L^*) \leq n$$

as desired. \(\square\)
In fact, using far more advanced tools, we can show something stronger, namely that \( \lambda_1(L^*) \cdot \lambda_n(L) \leq n \). 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 \( L \), \( \lambda_n(L^*) \cdot \lambda_1(L) \geq 1 \).

**Proof.** Let \( x \in L \) be the shortest non-zero vector. Let \( v_1, \ldots, v_n \in L^* \) be linearly independent. At least one of the \( v_i \) has a non-zero inner product with \( x \), say \( \langle v_i, x \rangle > 0 \). Since the inner products of lattice vectors and dual vectors are integers, \( \langle v_i, x \rangle \geq 1 \). Therefore,

\[
\lambda_1(L) = \|x\| \geq 1/\|v_i\| \geq 1/\lambda_n(L^*) .
\]

\[\square\]

**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 \( \rho_s \) 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

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

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, \( \int_{\mathbb{R}^n} |f(x)| dx < \infty \).

**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

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

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

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

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

\[
\hat{\rho}_s(x) = s^n \cdot \rho_{1/s}(x)
\]
Proof. We provide a proof in one dimension.

\[
\hat{\rho}_s(y) = \int_{\mathbb{R}^n} \rho_s(x) e^{-2\pi i \langle x, y \rangle} dx
\]

\[
= \int_{\mathbb{R}^n} e^{-\pi ||x||^2 / s^2} e^{-2\pi i \langle x, y \rangle} dx
\]

\[
= e^{-\pi s^2 ||y||^2} \int_{\mathbb{R}^n} e^{-\pi ||(x/s + iy)||^2} dx
\]

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

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

\[\Box\]

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 \( f: \mathbb{R}^n \rightarrow \mathbb{C} \) that is \( \mathcal{L} \)-periodic, that is, \( f(x+y) = f(x) \) for all \( x \in \mathbb{R}^n \) and \( y \in \mathcal{L} \), we have its Fourier series \( \hat{f}: \mathcal{L}^* \rightarrow \mathbb{C} \) defined as

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

We will state the Fourier inversion formula below without proof.

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

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 \( f: \mathbb{R}^n \rightarrow \mathbb{C} \), and any full-rank lattice \( \mathcal{L} \), we have

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

**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

\[
\phi(x) = \sum_{z \in \mathcal{L}} f(x + z)
\]
Clearly $\phi$ is periodic over $\mathcal{L}$, therefore $\hat{\phi}$ is defined over $\mathcal{L}^\ast$. For any $y \in \mathcal{L}^\ast$, we have

$$\hat{\phi}(y) = \det(\mathcal{L}^\ast) \int_{x \in \mathcal{P}(\mathcal{L})} \phi(x)e^{-2\pi i (x, y)} \, dx$$

$$= \det(\mathcal{L}^\ast) \int_{x \in \mathcal{P}(\mathcal{L})} \left( \sum_{z \in \mathcal{L}} f(x + z) \right) e^{-2\pi i (x, y)} \, dx$$

$$= \det(\mathcal{L}^\ast) \sum_{z \in \mathcal{L}} \int_{x \in \mathcal{P}(\mathcal{L})} f(x + z) e^{-2\pi i (x, y)} \, dx$$

$$= \det(\mathcal{L}^\ast) \sum_{z \in \mathcal{L}} \left( \int_{x \in \mathbb{R}^n} f(x) e^{-2\pi i (x, y)} \, dx \right)$$

$$= \det(\mathcal{L}^\ast) \hat{f}(y)$$

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

Now use Fourier inversion for $\phi$ to show that

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

Smoothing Lemma and Proof

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

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

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

$$\phi_s(x) = \sum_{y \in \mathcal{L}^\ast} \hat{\phi}_s(y) e^{2\pi i (x, y)}$$

$$= \det(\mathcal{L}^\ast) \cdot (1/s^n) \sum_{y \in \mathcal{L}^\ast} \hat{\rho}_s(y) e^{2\pi i (x, y)}$$

$$= s^n \cdot \det(\mathcal{L}^\ast) \cdot (1/s^n) \sum_{y \in \mathcal{L}^\ast} \rho_{1/s}(y) e^{2\pi i (x, y)}$$

$$= \det(\mathcal{L}^\ast) \cdot \left( 1 + \sum_{y \in \mathcal{L}^\ast \setminus \{0\}} \rho_{1/s}(y) \cdot e^{2\pi i (y, x)} \right)$$

24
where the first equality is by the definition of Fourier inversion, the second by the definition of $\phi_s$ 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 $\phi_s$ from the uniform distribution over the parallelepiped whose density function is $U_{P(B)}(x) = 1 / \det(L) = \det(L^*)$.

$$
\Delta(\phi_s, U_{P(B)}) = \int_{P(B)} |\phi_s(x) - U_{P(B)}(x)| \, dx
= \det(L^*) \int_{P(B)} \left| \sum_{y \in L^* \setminus \{0\}} \rho_1/s(y) \cdot e^{2\pi i \langle y, x \rangle} \right| \, dx
= \det(L^*) \cdot \det(L) \cdot \max_{x \in P(B)} \left| \sum_{y \in L^* \setminus \{0\}} \rho_1/s(y) \cdot e^{2\pi i \langle y, x \rangle} \right|
\leq \sum_{y \in L^* \setminus \{0\}} \rho_1/s(y) := \rho_1/s(L^* \setminus \{0\})
$$

(3.1)

In other words, we established $\rho_1/s(L^* \setminus \{0\})$ as the quantity that governs the variation (or statistical) distance between the continuous Gaussian reduced modulo $P(B)$ and the uniform distribution over $P(B)$. 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 $\rho_1/s(L^*)$ by something close to 1. First of all, the heaviest vector is the zero vector that gets a weight of 1. Secondly, if $\lambda_1(L^*) \gtrsim (1/s\sqrt{2\pi}) \cdot \omega(\sqrt{\log n})$, 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 $\lambda_1$ 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 $L$ be a rank-$n$ lattice. The number of lattice vectors of length at most $r$ is at most $\left( 1 + \frac{2r}{\lambda_1(L)} \right)^n$.

**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

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

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

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

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

**Lemma 14.** Let \( L \) be a rank-\( n \) lattice. \( \sum_{y \in L} \rho_s(y) = 1 + 2^{-O(n)} \) 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

\[
\sum_{y \in L} \rho_s(y) = \int_0^1 |\{ y \in L : \rho_s(y) \geq t \}| dt
\]

\[
= \int_0^1 |\{ y \in L : e^{-\pi ||y||^2/s^2} \geq t \}| dt
\]

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

\[
= \frac{2\pi}{s^2} \int_0^{\infty} \left| \{ y \in L : ||y|| \leq r \} \right| r e^{-\pi r^2/s^2} dr
\]

\[
\leq \frac{2\pi}{s^2} \left( \int_0^{\lambda_1} + \int_{\lambda_1}^{\infty} \right) \left| \{ y \in L : ||y|| \leq r \} \right| r e^{-\pi r^2/s^2} dr
\]

\[
\leq (1 - e^{-\pi \lambda_1^2/s^2}) + \frac{2\pi}{s^2} \int_{\lambda_1}^{\infty} \left| \{ y \in L : ||y|| \leq r \} \right| r e^{-\pi r^2/s^2} dr
\]

\[
\leq 1 + \frac{2\pi}{s^2} \int_{\lambda_1}^{\infty} \left( \frac{3r}{\lambda_1} \right)^n r e^{-\pi r^2/s^2} dr
\]

\[
\leq 1 + \frac{2\pi C^n}{s^2 \lambda_1} \int_{\lambda_1}^{\infty} r^{n+1} e^{-\pi r^2/s^2} dr
\]

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

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

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 \( L^* \) and the function is \( \rho_{1/s} \), we get that the sum \( \sum_{y \in L^*} \rho_{1/s}(y) \) is exponentially close to 1 as long as

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

Indeed, if \( s \) is so large, we have \( \lambda_1(L^*) \geq 1/\lambda_n(L) \geq \frac{C}{s} \cdot \sqrt{\frac{n}{2\pi e}} \) 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 $T = \mathbb{R}/\mathbb{Z}$, $\text{SIS}_T$. $\text{SIS}_T$ is exactly as in $\text{SIS}$, except that you are given a matrix $A \in T^{n \times m}$ and you are asked to find a small integer linear combination that sums to zero. That is, find $x \in \mathbb{Z}^m$ such that $Ax = 0$ and $\|x\|$ is “small”.

How would such a reduction look like? On the one hand, the reduction has to generate a uniformly random $\text{SIS}_T$ instance from a given lattice $L$; therefore, the SIS instance “forgets” the lattice $L$ 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 $L$. 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 $L$.

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

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

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

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

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

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

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

Assume now that there is a $\text{SIS}_T$ algorithm that gives us a non-zero integer vector $x \in \mathbb{Z}^m$ such that $Ax = 0$ (mod 1). Then we know that $B^{-1}Vx \in \mathbb{Z}^n$ and therefore, $Vx \in L(B)$ is a lattice vector. Now, since $X \equiv V$ (mod $P(B)$), we know that $Xx \in L(B)$ is also a lattice vector.

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

$$A = Q + E \pmod{1}$$
where \( qQ \in \mathbb{Z}^{n \times m} \) and \( \|E\|_\infty \leq 1/2q \).

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

\[
0 = Qx = (A - E)x = B^{-1}(V - BE)x = B^{-1}(X - BE)x \pmod{1}
\]

and therefore, \( (X - BE)x \) is a lattice vector. We would, in analogy to before, show that these are short lattice vectors.

\[
||Xx - BEx|| \leq ||Xx|| + ||BEx|| \leq s||x||\sqrt{n} + \frac{||x||}{q} \cdot \max_i ||b_i||_2
\]

So, this does not give us short vectors, rather it reduces the length of the longest vector in the basis by a factor of \( q/||x||_1 \geq q/m \) (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 \( x \) which is somehow cleverly designed so that \( (X - BE)x \) is the zero vector. What does the SIS algorithm see? It possibly sees \( V = X \pmod{P(B)} \) but (a) it never sees \( X \) itself; and (b) given \( V \), there are multiple possible values of \( X \), which is a consequence of smoothing-type arguments. In other words, the adversary is trying to force \( (X - BE)x \) to be \( 0 \), but it does not know what \( X \) 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 $\Lambda^\perp(A) := \{x \in \mathbb{Z}^n : Ax = 0 \text{ (mod } q)\}$). However, even viewing through these lens, what we have demonstrated is an algorithm that finds vectors of length related to $\lambda_n$, and not $\lambda_1$. 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.
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 \( s \) is just as easy as solving it with a uniformly random secret \( s \). That is, it is easy to re-randomize the secret \( s \). The key observation is that \( A \) is public and that everything here is additive.

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

\[
    b^T := b_{wc}^T + s_r^T A
\]

for a uniformly random vector \( s_r \leftarrow \mathbb{Z}_q^n \). Now, note that

\[
    b^T := (s_{wc} + s_r)^T A + e^T
\]

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

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 $D_0$ and $D_1$. Such an algorithm is said to be a $(T, \varepsilon)$-distinguisher if it runs in time $T$ and has a (distinguishing) advantage of $\varepsilon$:

$$|\Pr[x \leftarrow D_0; \text{Dist}(x) = 1] - \Pr[x \leftarrow D_1; \text{Dist}(x) = 1]| \leq \varepsilon$$

Equivalently,

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

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

Proof. Our approach to solve search 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 $s$, that is, $s = (s_1, \ldots, s_n)$. Consider the algorithm which, on input $(A, s^T A + e^T)$, for each $i \in [m]$, guesses the $i$th coordinate of $s$ as described in Algorithm 1 below. First of all, the algorithm partitions the columns of $A$ into $n \cdot q \cdot \tilde{O}(m/\varepsilon^2)$ parts – $n$ for the number of coordinates of $s$; $q$ 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 $s$

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

- Let $g_i := j$.
- For $\ell = 1, \ldots, L = \tilde{O}(1/\varepsilon^2)$:
  - Choose a fresh block of the search LWE challenge, call it $(A_\ell, b_\ell)$.
  - Sample a random vector $c_\ell \leftarrow \mathbb{Z}_q^m$, and let $C_\ell \in \mathbb{Z}_q^{n \times m}$ be the matrix whose $i$-th row is $c_\ell$, and whose other entries are all zero.
  - Let $A'_\ell := A_\ell + C_\ell$, and $b'_\ell = b_\ell + g_i \cdot c_\ell$.
  - Run the distinguisher $\mathcal{D}$ on input $(A'_\ell, b'_\ell)$ and let the output of $\mathcal{D}$ be called $d_\ell$.
- If $\text{maj}(d_1, \ldots, d_L) = 1$ (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 $(A'_\ell, b'_\ell)$ given to $\mathcal{D}$ are fresh LWE samples, since

$$b'_\ell = b_\ell + s_i \cdot c_\ell = s^T A_\ell + e^T_\ell + s_i \cdot c^T_\ell \quad \text{(expanding $b'_\ell$)}$$

$$= (s^T A_\ell + s_i \cdot c^T_\ell) + e^T_\ell \quad \text{(rearranging)}$$

$$= s^T (A_\ell + C_\ell) + e^T_\ell \quad \text{(by construction of $C_\ell$)}$$

$$= s^T A'_\ell + e^T_\ell. \quad \text{(by definition of $A'_\ell$)}$$
On the other hand, if the guess $g_i$ is wrong, i.e. $s_i \neq g_i$, then the inputs $(A'_\ell, b'_\ell)$ given to $D$ are uniformly random, since

$$b'_\ell = b_\ell + g_i \cdot c_\ell = s^T A_\ell + e_\ell^T + g_i \cdot c_\ell^T = (s^T A'_\ell + g_i \cdot c_\ell^T) + e_\ell^T = s^T A'_\ell + e_\ell^T + (g_i - s_i) \cdot c_\ell,$$

and the term $(g_i - s_i) \cdot c_\ell$ 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) $c_\ell$ is random and independent of $A'_\ell, s$ and $e_\ell$.

It follows that $D$ will output 1 with probability at least $1/2 + \varepsilon$, in the case that $s_i = g_i$. Since we run $D$ 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 $D$ 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 $s$ correctly. Therefore, applying Algorithm 1 to each coordinate of $s$ will, with overwhelming probability, correctly output all coordinates $s_1, \ldots, s_n$ of $s$.

**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 $\text{poly} \log q$: 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 $q / \text{poly}(n)$. 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 $\text{poly} \log q$ 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 $s$) 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 $D_0, D_1, \ldots, D_k$ where $D_i$ produces

$$ (a, (a, s) + e + r \cdot 2^i \pmod{q}) $$
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 \( 1/poly(n) \) advantage, there is a \( j \in [k] \) such that it distinguishes between \( \mathcal{D}_{j-1} \) and \( \mathcal{D}_j \) with advantage at least \( 1/k \cdot 1/poly(n) \). 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 \((a, b)\), create a sample
  \[ (a', b') = (a + r \cdot 2^{j-1} \cdot u_1, b) \]
  where \( u_1 \) 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
  \[ (a', b') = \langle a', s \rangle + e + r \cdot 2^j \pmod{q} \]
  where \( a' \) and \( r \) are uniformly random and independent. On the other hand, if the LSB of \( s_1 \) is 1, this looks like
  \[ (a', b') = \langle a', s \rangle + e + r \cdot 2^{j-1} \pmod{q} \]
  where again, \( a' \) 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 \( s \), up to a certain point. We first transform the given LWE sample \((a, b)\) 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 \((a, b) \rightarrow (a, b - \langle a, \text{LSB}(s_1) \cdot u_1 \rangle)\).
  From then on, to recover the \( k \)-th least significant bit, we do:
  \[ (a', b') = (a + r \cdot 2^{j-k} \cdot u_1, b) \]
  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
  \[ (a, b + r \cdot 2^j + e' \pmod{q}) \]
  where \((a, b)\) is an LWE sample with noise rate \( \alpha q \) and \( e' \) is a fresh Gaussian with noise rate about \( \alpha q \).
  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 \( q/2j' \approx 1/\alpha = poly(n) \).
  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).

34
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 \[?\]. The rough idea is as follows: look at an LWE input \((A, s^T A + e^T)\) where \(s \in \{0,1\}^n\). Suppose \(A\) were decomposable into \(BC\) where \(B \in \mathbb{Z}_q^{n \times k}\) and \(C \in \mathbb{Z}_q^{k \times m}\) are uniformly random. The reader should think of \(k \approx n/\log q = H_\infty(s)\), the min-entropy of the vector \(s\). Then, \(s^T A + e^T = s^T BC + e^T = (s^T B)C + e^T\). In other words, one can think of this as an LWE input w.r.t. the public matrix \(C\) with the secret being \(s^T B\). The key point is that multiplication by \(B\) extracts randomness from \(s\) and makes \(s^T B\) (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 \(s\) w.r.t. \(A\) looks statistically close to an LWE input with a uniformly random secret \(s' := B^T s\) 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 \(H_\infty(s)/\log q \geq \lambda\) for some security parameter \(\lambda\).)

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

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

\[s^T A + e^T = s^T (BC + N) + e^T = (s^T B)C + (s^T N + e^T)\]

\(s^T B\) 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 \(s\), ruining the LHL; and (2) it makes the error distribution wonky. A cheap way to get around this problem is to ensure that \(|s^T N|\) is small, say \(\text{poly}(n)\), for example by ensuring that \(s\) is binary and \(N\) has \(\text{poly}(n)\)-bounded entries, and using the so-called noise flooding trick, setting \(e^T\) to be a Gaussian with a superpolynomially larger standard deviation. This ensures that \(s^T N + e^T\) looks statistically like a fresh Gaussian, independent of \(s^T N\). 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. \[?\]. Nevertheless, the following question is still open:

**Open Problem 4.1.** For which distributions of the secret \(s\) 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 [?]. 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 [?] in the context of fully homomorphic encryption.

The rough idea is as follows: Assume that you are given LWE samples \((A, b)\) with a 0-1 secret relative to a matrix \(A \mod q\). 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

\[
\left(\left\lfloor \frac{p}{q} A \right\rfloor, \left\lfloor \frac{p}{q} b \right\rfloor \right)
\]

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

\[
(p/q)b = (p/q) \cdot (s^T A + e^T + qz^T) = s^T A' + s^T \{p/qA\} + (p/q)e^T + pqz^T
\]

where \(z\) 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. \((p/q)e^T\) is a Gaussian with parameter \(ap\) if \(e\) is Gaussian with parameter \(aq\). Assuming \(p\) is quasipolynomially large, one can use the noise-flooding lemma to “absorb” the error \(s^T \{p/qA\}\) 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 Stehlé [?]. 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 \(B \in \mathbb{Z}^{m \times m}\) of a lattice \(\mathcal{L}(B)\) and a target vector \(t \in \mathbb{Z}^m\) such that \(D(t, \mathcal{L}(B)) \leq c \cdot \lambda_1(\mathcal{L}(B))\), and the goal is to find the lattice vector that is closest to \(t\).

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

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

(Note that \(q \mathbb{Z}^m \subseteq \Lambda(A) \subseteq \mathbb{Z}^m\).) It is not hard to show that the minimum distance of \(\Lambda(A)\) for a uniformly random matrix \(A \in \mathbb{Z}_q^{n \times m}\) 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 \(s\) (which defines the closest vector) is uniquely determined given \(s^T A + e^T\).
4.3 Discrete Gaussians

As we saw in the last lecture, the Gaussian function
\[ \rho_s(x) := e^{-\pi||x||^2/s^2} \]
from \( \mathbb{R}^n \) to \( \mathbb{R} \) can be turned into a probability distribution over \( \mathbb{R}^n \) by normalizing with \( \int_{\mathbb{R}^n} \rho_s(x) dx = s^n \). Henceforth, we will call this the \((n\text{-dimensional})\) Gaussian distribution \( N_s \). Thus, \( N_s(x) = s^n \cdot e^{-\pi||x||^2/s^2} \).

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 \( x \notin \mathcal{L} \) and the values \( D_{\mathcal{L},s}(x) = \rho_s(L) \) for every \( x \in \mathcal{L} \). Here, \( \rho_s(L) := \sum_{v \in \mathcal{L}} \rho_s(v) \).

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 \( \mathcal{L} + \mathbf{c} = \{v + \mathbf{c} : v \in \mathcal{L}\} \) 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 \( \rho_{s,c}, N_{s,c} \) and \( D_{\mathcal{L},s,c} \); for example, \( \rho_{s,c}(x) := e^{-\pi||x-c||^2/s^2} \), and so on.

When \( s \) exceeds the smoothing parameter of the lattice \( \eta(\mathcal{L}) \), 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 \( s \geq \eta(\mathcal{L}) \),
\[ \rho_s(\mathcal{L} + \mathbf{c}) \in [1 - 2\varepsilon, 1 + 2\varepsilon] \cdot \rho_s(\mathcal{L}) \]
Proof. Let \( \mathbf{c}' \) denote the shortest vector in the lattice coset \( \mathcal{L} + \mathbf{c} \). Then,
\[ \rho_s(\mathcal{L} + \mathbf{c}) = \rho_{s,-\mathbf{c}}(\mathcal{L}) \]
\[ = \det(\mathcal{L}^*) \cdot \rho_{s,-\mathbf{c}}(\mathcal{L}^*) \]
\[ = \det(\mathcal{L}^*) \cdot \sum_{\mathbf{z} \in \mathcal{L}^*} \rho_{s,-\mathbf{c}}(\mathbf{z}) \]
\[ = \det(\mathcal{L}^*) \cdot \sum_{\mathbf{z} \in \mathcal{L}^*} e^{2\pi i(\mathbf{c},\mathbf{z})} \rho_{1/s}(\mathbf{z}) \]
\[ = \det(\mathcal{L}^*) \cdot \left( 1 + \sum_{\mathbf{z} \in \mathcal{L}^* \setminus \{0\}} e^{2\pi i(\mathbf{c},\mathbf{z})} \rho_{1/s}(\mathbf{z}) \right) \]
\[ \in [1 - \varepsilon, 1 + \varepsilon] \cdot \det(\mathcal{L}^*) \]
The claim follows.\( \square \)

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 $v$ from the discrete Gaussian $D_{\mathcal{L}, s}$ and a vector $w$ from the continuous Gaussian $N_r$ and adding them together, where $s, r \geq \eta_{\varepsilon}(\mathcal{L}) \cdot \sqrt{2}$ (where $\varepsilon$ 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 $t = \sqrt{r^2 + s^2}$.

$$Y(x) = \sum_{v \in \mathcal{L}} \Pr[v] \cdot \Pr[x - v]$$

$$= \frac{1}{\rho_s(\mathcal{L}) \cdot t^n} \sum_{v \in \mathcal{L}} \rho_s(v) \cdot \rho_x(x - v)$$

$$= \frac{1}{\rho_s(\mathcal{L}) \cdot t^n} \sum_{v \in \mathcal{L}} e^{-\pi ||v||^2/s^2} \cdot e^{-\pi ||x-v||^2/r^2}$$

$$= \frac{1}{\rho_s(\mathcal{L}) \cdot t^n} \sum_{v \in \mathcal{L}} e^{-\pi \left( ||v||^2/(t^2/r^2s^2) - 2(x,v)/r^2 + ||x||^2/r^2 \right)}$$

$$= \frac{1}{\rho_s(\mathcal{L}) \cdot t^n} \sum_{v \in \mathcal{L}} e^{-\pi \left( ||v||^2/(t^2/r^2s^2) - 2(x,v)/r^2 + ||x||^2/s^2 - (2x,t)/t^2 \right)}$$

$$= \frac{\rho_t(x) \cdot t^n \cdot \rho_{rs/t^2s^2} \cdot x(\mathcal{L})}{\rho_s(\mathcal{L})}$$

$$\in [1 - \varepsilon, 1 + \varepsilon] \cdot \frac{\rho_t(x) \cdot t^n \cdot \rho_{rs/t^2s^2} \cdot x(\mathcal{L})}{\rho_s(\mathcal{L})}$$

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

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

$$Y(x) \in [1 - 3\varepsilon, 1 + 3\varepsilon] \cdot N_t(x)$$

from which it follows that the statistical distance between the two distributions in question is at most $3\varepsilon$.

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 $\hat{s} = s^n \rho_{1/s}$, and the second the definition of the smoothing parameter and the fact that $s \geq \eta_{\varepsilon}(\mathcal{L})$. Thus,

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

\[\square\]
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 \( D_{L, s} \) given \( s \) and some basis \( B \) of \( L \). Clearly, this is hard to do if \( s < \frac{1}{\sqrt{n}} \cdot \max \| b_i \| \) as it will then give us a way to make the vectors of \( B \) shorter, a computationally hard problem. However, one can hope that for significantly larger \( s \), this is possible. Indeed, Gentry, Peikert and Vaikuntanathan [?], following an algorithm of Klein [?], show such a (polynomial-time) algorithm with \( s \geq \omega(\sqrt{\log n}) \cdot \max \| b_i \| \) (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 \text{SimpleDGS} \( \) that samples from the discrete Gaussian \( D_{L, s} \) where \( s \geq 2^n \cdot \max \| b_i \| \). The algorithm simply samples a vector \( v \leftarrow N_s \) from the continuous Gaussian distribution with parameter \( s \) and “rounds” it modulo the parallelepiped \( P(B) \). That is, output

\[ v' = B \lfloor v \rfloor \in L(B) \]

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

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

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

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

It suffices to show that \( |2\langle w, x \rangle + ||x||^2|/s^2| \) is very small. Note that this quantity is at most \( (2||w|| ||x|| + ||x||^2)/s^2 \) by Cauchy-Schwartz. Since \( ||w|| \approx s \sqrt{n} \) is the length of the typical vectors (Exercise: Check this!) and \( s \gg 2^n \max \| b_i \| \geq 2^n ||x|| \), 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 \( L = \mathbb{Z} \) and show that for small \( s \), the rounded continuous Gaussian (our distribution) and the discrete Gaussian over \( \mathbb{Z} \) are in fact statistically far.

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 \( a_i \) are statistically close to uniformly random in \( \mathbb{Z}_q^n \) and independent.

\[ \frac{\rho_s(qL^* + c)}{\sum_c \rho_s(qL^* + c)} \]

(4.1)
Regev’s BDD to LWE Reduction

**Input:** Lattice basis \( B \in \mathbb{Z}^{n \times n} \), \( t = Bs + e \in \mathbb{Z}^n \).

(For simplicity, we will assume that \( ||e|| \) is known.)

**Output:** LWE instance \( A \in \mathbb{Z}_q^{n \times m} \), \( y \in \mathbb{Z}_m^q \).

Repeat \( m \) times:

- Let \( q \geq 2^{2n} \), where \( s \geq q \sqrt{2} \cdot \eta_\varepsilon(L^*) \) and \( r \geq \sqrt{2} \cdot ||x|| \cdot \eta_\varepsilon(L^*) \).
- Sample a vector \( v_i \leftarrow D_{L^*,s} \).
- Compute \( a_i := (B^*)^{-1}v_i = B^Tv_i \pmod{q} \) and \( b_i := t^Tv_i + e_i' \pmod{q} \) where \( e_i' \leftarrow N_r \).

Run the LWE algorithm on input \((A, b)\) where the columns of \( A \) are the \( a_i \), and output what it outputs.

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

\[
\sum_c \rho_s(qL^* + c) \in [1 - 2\varepsilon, 1 + 2\varepsilon] \cdot \rho_s(qL^*) \cdot q^n
\]

and

\[
\rho_s(qL^* + c) \in [1 - 2\varepsilon, 1 + 2\varepsilon] \cdot \rho_s(qL^*)
\]

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 \( 4\varepsilon \). \(\square\)

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

**Proof.** For the reduction and the proof, we will assume that \( ||e|| \) is known. This assumption can be removed with more care; we refer to [?] for more details.

Start by noting that

\[
b_i = t^Tv_i + e_i' \pmod{q} \\
= (s^TB^T + e^T)B^Ta_i + e_i' \pmod{q} \\
= s^TB^B^Ta_i + e^Tv_i + e_i' \pmod{q} \\
= s^Ta_i + e_i \pmod{q}
\]

where the second equality follows from the definition of \( t := Bs + e \) and that of \( a_i \), and \( e_i := e^Tv_i + e_i' \).

It remains to analyze the distribution of \( e_i \).
First, $e'_i$ is distributed like $e^T w_i$ where $w_i$ is a continuous Gaussian with parameter $\sqrt{2} \eta_\epsilon (\mathcal{L}^*)$. Thus,

$$e'_i = e^T (v + w) = e^T w'$$

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

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

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 [17]. We contrast this with Regev’s quantum reduction from SIVP to BDD [15].

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 $z \in \mathcal{L}$ and add a random point $e$ from a ball of radius $c \cdot \gamma$.
- Run the BDD solver with input $t := z + e$.
- If the BDD solver produces a vector $z' = z$, output NO (“large $\lambda_1$”) else output NO (“small $\lambda_1$”).

On the one hand, if $\lambda_1 (\mathcal{L}) > \gamma$, then the distance of $t$ 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 $z$. On the other hand, if $\lambda_1 (\mathcal{L}) \leq 1$, the (uniform distribution on the) balls centered at $z$ and $z + u$ where $||u|| = \lambda_1 (\mathcal{L})$ 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.
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

$$G_A(s, e) = s^T A + e^T$$

is a pseudorandom generator with two caveats:

- It is a family of PRGs indexed by $A$. 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 $A$).

- 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 $e$, roughly $m \log(\alpha q)$ to sample from a Gaussian of standard deviation $\alpha q$. 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 $G$ is an $n \times m$ matrix (where $m \geq n \log q$) with the property that $G \cdot \{0, 1\}^m \supseteq \mathbb{Z}_q^n$. That is, for every vector $v \in \mathbb{Z}_q^n$, there is a 0-1 vector $w$ such that $Gw = v \pmod{q}$. For example, the matrix $G \in \mathbb{Z}_2^{2 \times 6}$ is the following matrix:

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

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,

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

More generally, let $g$ 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, $G = g \otimes I_n$ is the tensor product of $g$ with the $n \times n$ identity matrix $I_n$. (If $m > n \lceil \log q \rceil$, pad this with a block of the zero matrix.)

We will denote the inverse mapping by $G^\leftarrow$. That is, $G^\leftarrow(v) = w$ if (a) $w$ has 0 or 1 entries; and (b) $Gw = v \pmod{q}$. Note that there could be many such $w$ that satisfy these properties, so $G^\leftarrow$ 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 $A$ are random small, and $s$ is random, is as secure as LWE. Note that (a) we showed that “Normal Form LWE” where $A$ is random and $s$ is random small, is as secure as LWE (in Lecture 1 and lecture 4) and (b) if both $A$ and $s$ have small entries, the problem is easy, as it is essentially just linear regression, a convex optimization problem.

Assume that $A \leftarrow \{0, 1\}^{N \times m}$ and $s \leftarrow \mathbb{Z}_q^N$ are uniformly random. The Flipped LWE problem asks to distinguish between $(A, s^T A + e^T)$ from a truly random pair from the same domains. Note that $s$ 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$(n, m, q, \chi)$.

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

$$(A, b = (G^T s)^T G^\leftarrow(A) + e)$$

pick a random $s' \in \mathbb{Z}_q^N$ and compute $b' = b + s'^T G^\leftarrow(A)$. Pass $(G^\leftarrow(A), b')$ to the flipped LWE adversary.
First, notice that $G^{-}(A)$ 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 $G$ matrix by adding more powers of two.

Secondly, notice that
\[
b' = b + s'^T G^{-}(A) = (G^T s + s')^T G^{-}(A) + e
\]
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.

\[\square\]

Construction

Both constructions we show will follow the following general template. The PRF family will be indexed by a secret seed $s \in \mathbb{Z}_{q^n}$, and a sequence of public matrices $A = (A_0, A_1, \ldots)$. On input $x \in \{0, 1\}^\ell$, the function will be defined as
\[
PRF_{s, A}(x) = s^T A_x + e_x^T \quad (\text{mod } q)
\]
where $A_x$ is defined as some function (depending on the construction) of $A$ and $x \in \{0, 1\}^\ell$.

The first problem that one encounters with this framework is where does the error $e_x^T$, 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 [?]. That is, we will define
\[
PRF_{s, A}(x) = \lfloor s^T A_x + e_x^T \rfloor_p \quad (\text{mod } p)
\]
where $\lfloor \cdot \rfloor_p : \mathbb{Z}_q \to \mathbb{Z}_p$ refers to a function that, on input $x \in \mathbb{Z}_q$ outputs the multiple of $p$ that is closest to it. That is,
\[
\lfloor x \rfloor_p = \left \lfloor \frac{p}{x} \right \rfloor
\]
where $\lfloor \cdot \rfloor$ refers to the function that rounds to the nearest integer.

In the BLMR construction, the public parameters are $A := (A_0, A_1)$ where both matrices are drawn at random from $\mathbb{Z}_q^{n \times n}$ and $A_x$ is defined as a subset product. We are now ready to define the BLMR construction. The construction sets
\[
A_x = G^{-}(A_{x_1}) \cdot G^{-}(A_{x_2}) \ldots G^{-}(A_{x_\ell}) = \prod_{i=1}^\ell G^{-}(A_i)
\]
and therefore,
\[
PRF_{s, A_0, A_1}(x) = \lfloor s^T A_x \rfloor_p \quad (\text{mod } p)
\]

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 $\text{NC}^2$. 

44
(Approximate) Key Homomorphism. The PRF has the attractive feature that $\text{PRF}_s(x) + \text{PRF}_{s'}(x)$ (where both PRFs use the same two public matrices $A_0$ and $A_1$) is approximately equal to $\text{PRF}_{s+s'}(x)$. 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 $\ell$. 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 $\text{PRF}_i(x)$ for $i = 0, \ldots, \ell$ as follows.

$$\text{PRF}_i(x) = [s^T A x]_p$$

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

Note that $\text{PRF}^{(0)}$ is exactly the PRF we defined with $s_0 = s$. On the other hand, $\text{PRF}^{(\ell)}$ is a random function. The proof goes via a hybrid argument that switches from $\text{PRF}^{(0)}$ to $\text{PRF}^{(\ell)}$ in $\ell$ steps. We will now show that each such switch is computationally indistinguishable to the adversary. For simplicity, we show this for $\text{PRF}^{(0)}$ versus $\text{PRF}^{(1)}$.

- First, consider

$$\text{PRF}^{(0)}(x_1 \ldots x_\ell) = [s^T A x]_p = [s^T G^{-}(A_{x_1}) \cdot \prod_{i=2}^{\ell} G^{-}(A_{x_i})]_p$$

- We first show that this distribution is statistically close to

$$[(s^T G^{-}(A_{x_1}) + e_{x_1}) \cdot \prod_{i=2}^{\ell} G^{-}(A_{x_i})]_p$$

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

- Next, observe that this is computationally indistinguishable from

$$s_{x_1}^T \prod_{i=2}^{\ell} G^{-}(A_{x_i})$$

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

5.4 BP14 Construction

The only difference between the BLMR13 and BP14 constructions is in the definition of $A_x$. Let $x = x_1 x_2 \ldots x_\ell$. BP14 defines $A_x$ recursively as follows. $A_\varepsilon = I_{m \times m}$ (where $\varepsilon$ is the empty string) and

$$A_{b \varepsilon} = G^{-} (A_b \cdot A_{b \varepsilon})$$

Thus,

$$A_x = G^{-} (A_{x_1} \cdot G^{-} (A_{x_2} \ldots G^{-} (A_{x_\ell})))$$

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

$$\lfloor s^T G A_x \rfloor_p = \lfloor s^T A_{x_1} \cdot A_{x_2} \ldots \ell \rfloor_p$$

to

$$\lfloor (s^T A_{x_1} + e_{x_1}) \cdot A_{x_2} \ldots \ell \rfloor_p$$

by a statistical argument similar to the above. However, now, the norm of $A_{x_2} \ldots \ell$ is polynomial in $N$, independent of $\ell$ 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 $\text{NC}^1$ and is based on LWE with polynomial modulus.

The computation in $\text{NC}^1$ is satisfied by the BLMR construction (and by a construction of [?] using “synthesizers”), and the polynomial modulus is satisfied by the a direct construction based on GGM (also in [?]). We refer to [?] 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 $\text{SIS}(n, m, q, B)$ where $n \log q > m \log(B + 1)$. Each hash function $h_A$ is parameterized by a matrix $A \in \mathbb{Z}_q^{n \times m}$, takes as input $e \in [0, \ldots, B]^m$ and outputs

$$h_A(e) = Ae \mod q$$

A collision gives us $e, e' \in [0, \ldots, B]^m$ where $Ae = Ae' \mod q$ which in turn says that $A(e - e') = 0 \mod q$. Since each entry of $e - e'$ is in $[-B, \ldots, B]$, this gives us a solution to $\text{SIS}(n, m, q, B)$. 
We will work with the $\ell_\infty$ 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 
$$ \Lambda^\perp(A) = \{ z \in \mathbb{Z}^m : Az = 0 \pmod{q} \} $$

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

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

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

Note that the rank of $T$ 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 $T$ to be large is the crucial thing here. This is related to why $\Lambda^\perp(A)$ 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 $T$ is that each of its columns is a homogenous SIS solution with respect to $A$.

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 $A$. 

48
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 $\varepsilon$ as long as $H_{\infty}(P) \geq n \log q + 2 \log(1/\varepsilon)$:

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

where $A \leftarrow \mathbb{Z}_{q}^{n \times m}$ is uniformly random, $e \leftarrow X$ is drawn from the probability distribution $P$ and $u \leftarrow \mathbb{Z}_{q}^n$ 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 $A \in \mathbb{Z}_q^{n \times m}$ together with a single short vector $t \in \mathbb{Z}^m$ such that $At = 0 \pmod{q}$. We begin our journey into trapdoors by describing this simple procedure.

1. Pick a uniformly random matrix $A' \in \mathbb{Z}_q^{n \times (m-1)}$.
2. Pick a uniformly random vector $t \in \{0,1\}^{m-1}$.
3. Define

$$A = [A' || -A't] \quad \text{and} \quad t = \begin{bmatrix} t \\ 1 \end{bmatrix}$$

as the matrix and trapdoor vector, respectively.

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

More generally, if we let $\|t\|_{\infty} \leq B$, 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 $A$ 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 $A \in \mathbb{Z}_q^{n \times (m^*+\ell)}$ together with $\ell$ 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 ($\ell$) 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 $T \in \mathbb{Z}_q^{n \times n \log q}$ is a matrix with short columns such that $AT = G \pmod{q}$ where $G$ is the gadget matrix that we constructed and used in the last lecture.
To jog our memory, we defined
\[ g := \begin{bmatrix} 1 & 2 & 4 & \ldots & 2^{\lfloor \log q \rfloor - 1} \end{bmatrix} \quad \text{and} \quad G := I \otimes g \]
where \( I \) is the \( n \times n \) identity matrix. In other words, \( G \) is the block diagonal \( n \times (n \lfloor \log q \rfloor) \) matrix with \( g \) 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 \( b^T = s^T A + e^T \mod q \), we do
\[ b^T T = (s^T A + e^T) T = s^T G + e^T T \mod q \]
In other words, we just transformed an LWE sample relative to \( A \) into an LWE sample relative to \( G \), with a slight increase in error. Now, if we have a trapdoor (in the sense of Definition ??) for \( G \) (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 \( T_g \in \mathbb{Z}^{\lfloor \log q \rfloor \times \lfloor \log q \rfloor} \). We will then see that \( T_G = I \otimes T_g \).

Indeed,
\[ G \cdot T_G = (I \otimes g) \cdot (I \otimes T_g) = I \otimes (gT_g) = 0 \mod q \]
Here is the trapdoor for \( g \):
\[
T_g = \begin{bmatrix}
2 & 2 & \ldots \\
-1 & -1 & \ldots \\
\vdots & \vdots & \\
2 & -1 & 2
\end{bmatrix}
\]
Let us check.

- \( T_g \) has short columns. Indeed \( ||T_g||_\infty = 2 \).
- \( gT_g = 0 \mod q \).
- The determinant of \( T_g \) 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 \( T_g \in \mathbb{Z}^{\lfloor \log q \rfloor \times \lfloor \log q \rfloor} \).

We will then see that \( T_G = I \otimes T_g \). Here is the trapdoor for \( g \):
\[
T_g = \begin{bmatrix}
2 & 2 & \ldots \\
-1 & -1 & \ldots \\
\vdots & \vdots & \begin{array}{c}	ext{bits}(q) \end{array} \\
2 & -1 & \end{bmatrix}
\]

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 \( g \) 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 $B \in \mathbb{Z}_q^{n \times m^*}$ where $m^* = n \log q + \lambda$ (as before). Set

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

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

$$A \cdot \begin{bmatrix} -R \\ I \end{bmatrix} = G \quad \text{(mod } q)$$

and since $||R||_\infty \leq 1$, we have an inhomogenous trapdoor! Furthermore, $A$ 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 $A$.

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

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

Thus,

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

Finally, multiplying this on the right by $\begin{bmatrix} I & 0 \\ -G^{-1}(B) & T_G \end{bmatrix}$, we get

$$[B||BR + G] \cdot \begin{bmatrix} I \\ -G^{-1}(B) \end{bmatrix} \cdot \begin{bmatrix} I \\ 0 \\ T_G \end{bmatrix} = [B||G] \cdot \begin{bmatrix} I \\ -G^{-1}(B) \end{bmatrix} = 0 \quad \text{(mod } q)$$

Thus, the lattice trapdoor

$$T_A = \begin{bmatrix} I + RG^{-1}(B) & -RT_G \\ -G^{-1}(B) & T_G \end{bmatrix}$$

We already saw that $AT_A = 0 \pmod{q}$. The $\ell_\infty$ norm of $T_A$ is $O(m)$. Finally, since $T_A$ 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 $\mathcal{F}_n = \{f_i : \{0,1\}^n \rightarrow \{0,1\}^m\}$ 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 \( i, t \) and \( y \in \{0,1\}^m \), outputs \( x \in \{0,1\}^n \) or a special symbol \( \perp \). We require that if \( y \in \text{Image}(f_i) \), then \( x \) is an inverse, namely \( f_i(x) = y \).

Injective Trapdoor Function

The function
\[
f_A(s,e) = s^T A + e^T \pmod{q}
\]
where \( A \in \mathbb{Z}_q^n \), \( s \in \mathbb{Z}_q^n \) and \( e \leftarrow \chi^m \) is a one-way family of functions, under LWE. Given the trapdoor \( T \), one inverts this as follows.
\[
(s^T A + e^T)T = e^T T \pmod{q}
\]
Now, since the latter quantity has absolute value at most \( q/4 \), it is \( e^T T \) (over the integers). The \( \text{mod-}q \) has no effect, and this is the key observation. Now, multiplying the latter by \( T^{-1} \) (the inverse of \( T \) over the reals) recovers \( e \). Here, it is very important that \( T \) had full rank over the reals; otherwise, \( T^{-1} \) would not exist.

Surjective Trapdoor Function

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

One way to do this is the following. On input \( v \in \mathbb{Z}_q^n \), find some \( w \in \mathbb{Z}_q^m \) such that \( Aw = v \pmod{q} \). Consider outputting
\[
T \cdot \{T^{-1}w\}
\]
where \( \{x\} \) denotes the fractional part of \( x \in \mathbb{R} \). Why does this work?

• First of all,
\[
AT \cdot \{T^{-1}w\} = AT \cdot (T^{-1}w - \lfloor T^{-1}w \rfloor) = v - 0 = v \pmod{q}
\]
so we have an inverse.

• Secondly,
\[
\|T \cdot \{T^{-1}w\}\|_\infty \leq m \cdot \|T\|_\infty \leq m^2
\]
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 \( e' \in \{0,1\}^m \) such that \( Ge' = v \pmod{q} \). Now, we claim that
\[
\begin{bmatrix} R \\ I \end{bmatrix} \cdot e' \text{ is a required inverse. Indeed,}
\]
\[
A \cdot \begin{bmatrix} R \\ I \end{bmatrix} \cdot e' = G \cdot e' = v \pmod{q}
\]
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 $A$ and $T$. The public key is $A$ and the secret key is $T$.

- To sign a message $m$, first map it into the range of the function, e.g., by hashing it. That is, compute $v = H(m)$. The signature is an inverse of $v$ under the function $g_A$. That is, a short vector $e$ such that $Ae = v \pmod{q}$. This is guaranteed by the surjectivity of the function $g_A$.

- Verification, given a message $m$, public key $A$ and signature $e$, consists of checking that $Ae = H(m) \pmod{q}$ and that $\|e\|_\infty \leq m^2$.

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 $T$ leaks information about $T$. Collecting this leakage over sufficiently many (polynomially many) signature queries enables an adversary to find $T$, allowing her to forge signatures at will going forward.

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

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

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

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 $g_A$. The inverter is a “pre-image sampler”; that is, it is given the trapdoor $T$ and produces a “random” pre-image. More precisely, we need the following distributions to be statistically close (computational indistinguishability is fine, but we will achieve statistical closeness):

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

That is, the following processes produce statistically close outputs: (a) first sample $e$ from a discrete Gaussian, and deterministically set $v$ to be $Ae \pmod{q}$; and (b) sample $v$ uniformly and use the pre-image sampler to produce an inverse of $v$ under $g_A$ 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,

$$\Lambda_v(A) := \{ e \in \mathbb{Z}^m : Ae = v \pmod{q} \}$$

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

$$\left( A \leftarrow \mathbb{Z}_q^{n \times m}, \{ e_i \leftarrow D_{\mathbb{Z}_q^{m},s}, v := Ae \pmod{q} \}_{i=1}^{\text{poly}(\lambda)} \right)$$

$$\approx_s \left( A \leftarrow \mathbb{Z}_q^{n \times m}, \{ e \leftarrow \text{PreSamp}(A,T), v \leftarrow \mathbb{Z}_q^m \}_{i=1}^{\text{poly}(\lambda)} \right)$$

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

$$\left( e \leftarrow D_{\mathbb{Z}_q^{m},s}, v := Ae \pmod{q} \right)$$

$$\approx_s \left( e \leftarrow \text{PreSamp}(A,T), v \leftarrow \mathbb{Z}_q^m \right)$$

**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 $\text{Round}(N_s)$ is at least $1/s^3$.

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

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

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 $\text{negl}(\lambda)$-smoothing parameter of $\mathbb{Z}$ which is $\omega(\sqrt{\log \lambda})$.

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

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

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.

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 $\varepsilon = \text{negl}(\lambda)$, 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 $\text{negl}(\lambda)$-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 $O(t) = \text{poly}(\lambda)$.

**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 \(\mu\), 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.

- \textbf{Setup}(1^\lambda): is a probabilistic algorithm that generates a master public key \(MPK\) and a master secret key \(MSK\).
• $\text{Enc}(MPK, ID, \mu)$: is a probabilistic algorithm that generates a ciphertext $C$ of a message $\mu$ (for simplicity, we will encrypt bits but that is largely irrelevant) w.r.t. identity $ID$.

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

• $\text{Dec}(SK_ID, C)$: 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 administrative assistant). For example, all messages are tagged with a keyword $ID = \text{CS294}$, 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

- **KeyGen** $(1^\lambda)$: Generate a random matrix $A \in \mathbb{Z}_q^{n \times m}$ and its trapdoor $T$ by running $\text{TrapSamp}$.

- **Sign** $(\mu)$: first compute $v = H(\mu) \in \mathbb{Z}_q^n$ where $H$ is treated as a random oracle in the analysis. Then, use Gaussian sampling (via the GPV algorithm) to compute a Gaussian solution $e \in \mathbb{Z}_m$ to the equation

$$ Ae = v \pmod{q} $$

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

$$ \{ e \in \mathbb{Z}_m^m : Ae = 0 \pmod{q} \} $$

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

$$ \Lambda^\perp_v(A) = \{ e \in \mathbb{Z}_m^m : Ae = v \pmod{q} \} $$

Note that the distribution of $e$ is $D_{\Lambda^\perp_v(A),\sigma}$ where $\sigma \approx \|T\| \cdot \omega(\sqrt{\log n})$. (The $\omega(\sqrt{\log n})$ is so that the sampling algorithm can achieve negligible statistical distance from a true discrete Gaussian.)

- **Verify** $(A, e, \mu)$: check that (1) $e$ is short, that is $\|e\| \leq \|T\| \cdot \omega(\sqrt{n \log n})$; and (2) $Ae = H(\mu)$ (mod $q$).

The key question now is how to we build an encryption algorithm whose public key is $v$ (which will be treated as $H(ID)$) and the corresponding private key is $e$ 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 $A \in \mathbb{Z}_q^{n \times m}$ and a random vector $v \in \mathbb{Z}_q^n$. The private key is a short vector $e$ such that $Ae = v \pmod{q}$.

$$ pk = (A, v) \quad sk = e $$

- **Enc** $(pk, \mu)$: pick an LWE secret $s \in \mathbb{Z}_q^n$ and output

$$ (c_1^T, c_2) := \left( s^T A + x^T, s^T v + x' + m \lfloor q/2 \rfloor \right) $$

as the ciphertext. We will call this ciphertext the dual Regev encryption of $\mu$ relative to $A$ and $v$. 

59
Dec\((sk, (c_1^T, c_2))\): Compute
\[
\hat{\mu} := \text{Round}(c_2 - c_1^T e)
\]
where \(\text{Round}(\alpha)\) outputs 1 if \(|\alpha - q/2| \leq q/4\) and 0 otherwise.

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

### 7.4 The GPV IBE Scheme

- **Setup\((1^\lambda)\):** Pick the right \(n = n(\lambda)\) for a security level of \(\lambda\) bits. Generate a matrix \(A \in \mathbb{Z}_q^{n \times m}\) and its trapdoor \(T \in \mathbb{Z}_m^{m \times m}\) by running the trapdoor sampling algorithm.

\((A, T) \leftarrow \text{TrapSamp}(1^n)\)

(The parameters \(m\) and \(q\) are picked internally by the trapdoor sampling algorithm.) The master public key is \(\text{mpk} = A\) and the master secret key is \(\text{msk} = T\).

- **KeyGen\((\text{msk}, ID)\):** Compute \(v := H(ID) \in \mathbb{Z}_q^n\) where \(H : \{0, 1\}^* \rightarrow \mathbb{Z}_q^n\) is a hash function (which, in the security analysis, will be treated as a random oracle.) Generate a short vector

\[e \leftarrow \text{DGSamp}(A, T, v)\]

by running the discrete Gaussian sampling algorithm. Recall that \(Ae = v \pmod{q}\). Output the secret key \(sk_{ID} = e\).

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

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

**Proof of (Full) Security**

We will come up with alternate algorithms called **Setup\(^*\)**, **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.

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

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

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

- **Enc\(^*(\text{MPK}\^*, \text{ID}\^*, \mu)\):** return the dual Regev encryption of \(\mu\) relative to \(A^*\) and \(v^*\).
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 $A$ as the master public key and the dual-Regev public key $v^*$ 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 $\text{Setup}^*$ 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 $A$, can you generate a trapdoor for $[A||B]$ where $B$ is an arbitrary matrix?

The Scheme

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

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

  where $\ell$ is the length of the identities. The master public key is

  $$\text{mpk} = (A_{i,b})_{i \in [\ell], b \in \{0,1\}}, v$$

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

  $$\text{msk} = (T_{A_0}, T_{A_1})$$

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

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

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

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

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

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


Proof of (Selective) Security

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

- \texttt{Setup}*(ID*, 1\(^\lambda\)): sample random \(v^*\). sample \(\ell\) random matrices \(B_1, \ldots, B_\ell\) and set
  \[
  A_{i, ID_i^*} = B_i
  \]
  sample \(\ell\) matrices \(B'_1, \ldots, B'_\ell\) together with their trapdoors and set
  \[
  A_{i, 1-ID_i^*} = B'_i
  \]

  MPK* consists of all the \(A_{i,b}\) and \(v^*\). MSK* consists of the trapdoors of all \(A_{i,1-ID_i^*}\).

- \texttt{KeyGen}*(ID): We know that \(ID \neq ID^*\). Therefore, I know the trapdoor of the matrix
  \[
  A_{ID} := [A_{1,ID_1} \mid \ldots \mid A_{\ell,ID_\ell}]
  \]
  (do you see why?)

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

One can also prove full security with a more sophisticated proof. In one sentence, the idea is to set up \(A_{i,b}\) 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 \(A_{ID}\), you can generate a trapdoor for \(A_{ID||ID'} = [A_{ID}||A_{ID'}]\). 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 $A_0$, a matrix $R$ with small entries, and a trapdoor for $G$, can you generate a trapdoor for $[A_0 || A_0 R + \alpha \cdot G]$ for an arbitrary integer $\alpha \neq 0 \pmod{q}$?

How about for $\alpha = 0 \pmod{q}$, that is, $[A_0 || A_0 R]$?

The Scheme

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

  $$A_0, A_1 \in \mathbb{Z}_q^{n \times m}$$

  The master public key is

  $$\text{mpk} = A_0, A_1, v$$

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

  $$\text{msk} = T_{A_0}$$

  We will never use the trapdoor for $A_1$.

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

  $$A_{ID} := [A_0 || A_1 + h(ID) \cdot G]$$

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

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

- **Enc$(\text{mpk}, ID, \mu)$**: Run the dual Regev encryption algorithm with $pk := (A_{ID}, v)$ and message $\mu$ and output the resulting ciphertext.

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

ABB: Proof of Selective Security

As before, we will come up with a bunch of alternate algorithms called Setup*, 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.

- **Setup*(ID*, 1^\lambda)**: sample random $v^*$. sample a random matrix $A_0$ and a matrix $R$ with small entries. Set

  $$A_1 := [A_0 || A_0 R - h,ID^*)G]$$

  MPK* consists of $A_0, A_1$ and $v^*$. MSK* consists of $R$ (and the trapdoor for $G$.)
• KeyGen*(ID): We know that \( ID \neq ID^* \). Therefore, I know the trapdoor of the matrix

\[
A_{ID} := [A_0 || A_1 + h(ID)G] = [A_0 || A_0R + (h(ID) - h(ID^*)G]
\]

(do you see why?)

• Enc*(MPK*, ID*, \( \mu \))\: given a dual Regev encryption of \( \mu \) relative to \( A_0 \) and \( v^* \), compute a dual Regev encryption of \( \mu \) relative to

\[
A_{ID^*} = [A_0 || (A_0R - h(ID^*)G) + h(ID^*)G] = [A_0 || A_0R]
\]

and \( v^* \). (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 [?]. 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 \( \text{Dec}(SK, c) \) 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 \( 1/2 + \text{negl}(\lambda) \).

Here is the construction.

• KeyGen(1^\lambda): run lBE.Setup(1^\lambda) to get an MPKIbe and an MSKIbe. The public key \( PK \) of the CCA scheme is MPKIbe and the secret key \( SK \) is MSKIbe.

• Enc(PK, \( \mu \)): pick a random string \( ID \). Run lBE.Enc(PK = MPKIbe, ID, \( \mu \)) and output \( ID \) together with the resulting ciphertext.

• Dec(SK, (ID, c)): use \( SK = MSKIbe \) to create \( SK_{ID} \) and run the IBE decryption algorithm \( \mu = lBE.Dec(SK_{ID}, c) \).
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 [?, ?]. 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.
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 $\text{Enc}(x_1), \ldots, \text{Enc}(x_k)$ and produce an encryption of any polynomial-time computable function of them, that is, $\text{Enc}(f(x_1, \ldots, x_k))$ where $f$ is any function with a $\text{poly}(\lambda)$-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.

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

**Correctness** says that

$$\text{Dec}(sk, \text{Eval}(ek, f, \text{Enc}(sk, \mu_1), \ldots, \text{Enc}(sk, \mu_k))) = f(\mu_1, \ldots, \mu_k)$$
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 \( (\mu_i)_{i \in \text{poly}(\lambda)} \) and \( (\mu'_i)_{i \in \text{poly}(\lambda)} \) are computationally indistinguishable. (the fact that the encryption scheme is homomorphic is a functionality requirement, and does not change the notion of security.)

\[
\left( \text{Enc}(sk, \mu_1), \ldots, \text{Enc}(sk, \mu_{p(\lambda)}) \right) \approx_c \left( \text{Enc}(sk, \mu'_1), \ldots, \text{Enc}(sk, \mu''_{p(\lambda)}) \right)
\]

A final and important property is **compactness**, that is, \(|c_f| = \text{poly}(\lambda)\), independent of the circuit size of \( f \). (Weak 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 \( s = \begin{bmatrix} s' \\ -1 \end{bmatrix} \) where \( s' \in \mathbb{Z}_q^n \).
- **Enc**: output \( A + \mu G \) where \( A \) is a random matrix such that
  \[
  s^T A \approx 0 \pmod{q}
  \]
  Here is one way to do it: choose a random matrix \( A' \) and let
  \[
  A := \begin{bmatrix}
  A' \\
  (s')^T A' + e'
  \end{bmatrix}
  \]
- **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

\[
C = A + \mu I
\]

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

Now,

\[
s^T C = \mu s^T
\]
• ADD($C_1, C_2$) outputs $C_1 + C_2$. This is an encryption of $\mu_1 + \mu_2$ since
$$s^T(C_1 + C_2) = (\mu_1 + \mu_2)s^T$$
Eigenvalues add.

• MULT($C_1, C_2$) outputs $C_1C_2$. This is an encryption of $\mu_1\mu_2$ since
$$s^T(C_1C_2) = \mu_1s^TC_2 = \mu_1\mu_2s^T$$
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

$$C = A + \mu G$$

where $s^TA \approx 0$. Think of $G$ as an error correcting artifact for the message $\mu$.

Now,

$$s^TC \approx \mu s^TG$$

which is the approximate eigenvalue equation.

• ADD($C_1, C_2$) outputs $C_1 + C_2$. This is an encryption of $\mu_1 + \mu_2$ since
$$s^T(C_1 + C_2) \approx (\mu_1 + \mu_2)s^TG$$

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

• MULT($C_1, C_2$) outputs $C_1G^{-}(C_2)$. This is an encryption of $\mu_1\mu_2$ since
$$s^T(C_1G^{-}(C_2)) = (s^TC_1)G^{-}(C_2) \approx (\mu_1s^TG)G^{-}(C_2) = \mu_1(s^TC_2) \approx \mu_1\mu_2s^TG$$

where the first $\approx$ is because $G^{-}(C_2)$ is small and the second $\approx$ because $\mu_1$ 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 $\mu$ and $\mu'$ respectively, in the sense that their decryption noise has $\ell_\infty$ 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 $\tilde{C}$:

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

What you get out is an encryption of $\mu \text{ NAND } \mu'$. How did this happen (and what did happen?) First of all, note that $\tilde{C}$ is a fresh encryption of $sk$. Secondly, assume that the 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 $\mu \text{ NAND } \mu'$; therefore, putting together this discussion, the output of the homomorphic evaluation of the circuit is an encryption of $\mu \text{ NAND } \mu'$ under $sk$.

Once we can implement 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 $\text{BootNAND}_{C,C'}$ and homomorphically evaluate it on an encryption $\tilde{C}$ 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:

\[ s^T A_i \approx \mu_i s^T G \]

and rewrite it as

\[ s^T (A_i - \mu_i G) \approx 0 \]  \hspace{1cm} (8.1)

Let \( A_f \) be the homomorphically evaluated ciphertext for a function \( f \). We know that

\[ s^T A_f \approx f(\mu) s^T G \]

or

\[ s^T (A_f - f(\mu) G) \approx 0 \]  \hspace{1cm} (8.2)

We will generalize this to arbitrary matrices \( A_1, \ldots, A_\ell \) – not necessarily ones that share the same eigenvector.

First, we know that \( A_f \) is a function of \( A_1, \ldots, A_\ell \) and \( f \) (but not \( \mu_1, \ldots, \mu_\ell \)). Henceforth, when we say \( A_f \), 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 \( A_1, \ldots, A_\ell \),

\[ [A_1 - \mu_1 G] \cdots [A_\ell - \mu_\ell G] \mathbf{H}_{f,\mu} = A_f - f(\mu) G \]

where \( \mathbf{H}_{f,\mu} \) is a matrix with small coefficients. We call this the key lattice equation.
To see this for addition, notice that

$$[A_1 - \mu_1 G || A_2 - \mu_2 G] \left[ \begin{array}{c} I \\ H_{+,\mu_1,\mu_2} \end{array} \right] = A_1 + A_2 - (\mu_1 + \mu_2)G = A_+ - (\mu_1 + \mu_2)G$$

and for multiplication,

$$[A_1 - \mu_1 G || A_2 - \mu_2 G] \left[ \begin{array}{c} G^{-1}(A_2) \\ \mu_1 I \\ H_{\times,\mu_1,\mu_2} \end{array} \right] = A_1 G^{-1}(A_2) - \mu_1 \mu_2 G = A_\times - \mu_1 \mu_2 G$$

By composition, we get that

$$[A_1 - \mu_1 G || A_2 - \mu_2 G] \cdots [A_\ell - \mu_\ell G] H_{f,\mu} = A_f - f(\mu)G$$

where $H_{f,\mu}$ 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 $A_i$ and $A_f$, there exists such a small matrix $H$; but if $A_f$ is arbitrary, it is hard to find.

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

- The ciphertexts are the matrices $A_i$ and we picked them such that
  $$s^T A \approx \mu s^T G$$

- Homomorphic evaluation is computing $A_f$ starting from $A_1, \ldots, A_\ell$.

- Correctness of homomorphic eval follows from the key equation: We know that
  $$s^T[A_1 - \mu_1 G] \cdots [A_\ell - \mu_\ell G] \approx 0$$

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

$$s^T[A_f - f(\mu)G] = s^T[A_1 - \mu_1 G] \cdots [A_\ell - \mu_\ell G] H_{f,\mu} \approx 0$$

as well meaning that $A_f$ is an encryption of $f(\mu)$. Note that no one needs to know or compute the matrix $H$; 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 $\sigma_f$ 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 $\sigma_f$ that binds the output of a computation $f(\mu)$ with the computation itself $f$. Here is the definition:

1. $PK, f \rightarrow PK_f$.
2. $(\mu_1, \sigma_1), \ldots, (\mu_\ell, \sigma_\ell) \rightarrow (f(\mu), \sigma_f)$.  
   // Both the operations above are as expensive as computing $f$.
3. $\text{Verify}(PK_f, f(\mu), \sigma_f) = 1$.
4. For any $f$ and any $y \neq f(\mu)$, no PPT adversary can produce a (fake) signature $\sigma'$ such that $\text{Verify}(PK_f, y, \sigma') = 1$ (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 $B, A_1, \ldots, A_\ell$. (This scheme can sign $\ell$ bits). $SK$ is trapdoor of $B$.
- Signature $\sigma_i$ for a message $\mu_i$ is a short $R_i$ such that $BR_i = A_i - \mu_iG$.  
  // (Can you see how the signing algorithm works?)
- $PK_f$ is $A_f$.
- To homomorphically compute on the signatures, start from the key equation:

$$[A_1 - \mu_1G|\ldots|A_\ell - \mu_\ellG] H_{f,\mu} = A_f - f(\mu)G$$

Notice that the way we constructed signatures,

$$B \begin{bmatrix} R_1 \\ \vdots \\ R_\ell \end{bmatrix} H_{f,\mu} = \underbrace{A_f - f(\mu)G}_{PK_f}$$

$\sigma_f$ is thus the homomorphic signature of $f(\mu)$ under $PK_f$.

- Why can’t an adversary cheat? Suppose an adversary produces a signature $\sigma'$ that verifies for the message $y \neq f(\mu)$ w.r.t. $PK_f$. So,

$$B\sigma' = A_f - yG$$

Subtracting the last two equations, we get

$$B(\sigma' - \sigma_f) = (f(\mu) - y)G$$

So, $\sigma' - \sigma_f$ is an inhomogenous trapdoor for $B$, 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, $\mu = ID$.)
- **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 $\mu$ (attributes are in the clear) and uses $SK_f$ to decrypt a ciphertext $C$ if $f(\mu) = 1$ (true). If $f(\mu) = 0$, Dec simply outputs $\bot$.

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 $A, A_1, \ldots, A_\ell$ and a vector $v$ and these form the $MPK$. The $MSK$ is the trapdoor for $A$.
- **Enc** computes
  
  \[ s^T[A||A_1 - \mu_1 G|| \ldots ||A_\ell - \mu_\ell G] \]
  
  (plus error, of course, and we will consider that understood.) Finally, the message is encrypted as $s^Tv + e + m\lceil q/2 \rceil$.
- **Let’s see how Dec might work.** You (and in fact anyone) can compute
  
  \[ s^T[A||A_1 - \mu_1 G|| \ldots ||A_\ell - \mu_\ell G] \left[ \begin{array}{cc} I & 0 \\ 0 & H_{f,\mu} \end{array} \right] = s^T[A||A_f - f(\mu)G] \]
  
  using the key equation.

  If you had a short $r$ that maps $[A||A_f - G]$ to $v$, that is
  
  $[A||A_f - G] r = v$

  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 [?].
- One might wonder if the attributes $\mu$ 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 [?] 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

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

(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 $\text{PRF}(K, x^*)$ 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 [?] for details and extensions.

- The scheme has public parameters $B_0, B_1$ and $A_1, \ldots, A_k$ where $k$ is an upper bound on the description length of any function $f$ that will be constrained. The PRF key is $s$.

- To define the PRF, consider the universal function $U$:

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

The key equation applied to the universal circuit $U$ now tells us that

$$[A_1 - f_1G|| \ldots ||A_k - f_kG||B_{x_1} - x_1G|| \ldots ||B_{x_l} - x_lG] \ H_{U,f,x}$$

$$= A_{U,x} - U(f, x)G$$

$$= A_{U,x} - f(x)G$$

Here, $A_{U,x}$ (which we will simply denote as $A_x$) is a result of the GSW homomorphic evaluation on the matrices $A_1, \ldots, A_k, B_{x_1}, \ldots, B_{x_l}$.

- The PRF is defined to be $[s^T A_x]$ 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 [?] for details and extensions.

- The scheme has public parameters $B_0, B_1$ and $A_1, \ldots, A_k$ where $k$ is an upper bound on the description length of any function $f$ that will be constrained. The PRF key is $s$.

- To define the PRF, consider the universal function $U$:

$$\forall f \text{ where } |f| \leq k, x \in \{0,1\}^l : \ U(f, x) = f(x)$$
The key equation applied to the universal circuit $U$ now tells us that

$$[A_1 - f_1 G] \cdots [A_k - f_k G] || B_{x_1} - x_1 G] \cdots || B_{x_\ell} - x_\ell G] H_{U,f,x}$$

$$= A_{U,x} - U(f,x) G$$

$$= A_{U,x} - f(x) G$$

Here, $A_{U,x}$ (which we will simply denote as $A_x$) is a result of the GSW homomorphic evaluation on the matrices $A_1, \ldots, A_k, B_{x_1}, \ldots, B_{x_\ell}$.

- The PRF is defined to be $[s^T A_x]$ on every input $x$.
- The constrained key for a function $f$ is

$$s^T [A_1 - f_1 G] \cdots || A_k - f_k G] || B_0 || B_1 - G]$$

On input $x$, the constrained eval proceeds as follows. First you can get

$$[A_1 - f_1 G] \cdots || A_k - f_k G] || B_{x_1} - x_1 G] \cdots || B_{x_\ell} - x_\ell G]$$

for any $x$ of your choice. Second, using the key equation, multiplying this on the right by $H_{U,f,x}$:

$$s^T [A_x - f(x) G]$$

from which one computes $[s^T A_x] := \text{PRF}(K, x)$ if $f(x) = 0$ (true).

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

- The scheme has public parameters $B_0, B_1$ and $A_1, \ldots, A_k$ where $k$ is an upper bound on the description length of any function $f$ that will be constrained. The PRF key is $s$.

- To define the PRF, consider the universal function $U$:

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

The key equation applied to the universal circuit $U$ now tells us that

$$[A_1 - f_1 G] \cdots || A_k - f_k G] || B_{x_1} - x_1 G] \cdots || B_{x_\ell} - x_\ell G] H_{U,f,x}$$

$$= A_{U,x} - U(f,x) G$$

$$= A_{U,x} - f(x) G$$

Here, $A_{U,x}$ (which we will simply denote as $A_x$) is a result of the GSW homomorphic evaluation on the matrices $A_1, \ldots, A_k, B_{x_1}, \ldots, B_{x_\ell}$.

- The PRF is defined to be $[s^T A_x]$ on every input $x$.
- The constrained key for a function $f$ is

$$s^T [A_1 - f_1 G] \cdots || A_k - f_k G] || B_0 || B_1 - G]$$

On input $x$, you can compute

$$s^T [A_x - f(x) G]$$
• For security: suppose an adversary managed to compute

\[ s^T A_x + e \]

for some \( x \) where \( f(x) = 1 \). We can ourselves compute

\[ s^T (A_x - G) + e' \]

using constrained evaluation. Put together, these reveal \( s^T G \) plus error, and therefore \( s \), breaking LWE.
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

$$\forall x \text{ s.t. } f(x) = 1 : \text{PRF}(K, x)$$

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 $\text{PRF}(K, x^*)$ 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 $\text{PRF}_K(\cdot)$, 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 $\text{PRF}_K(x)$. She also uses $K\{f\}$ to compute something, and she knows that the output matches $\text{PRF}_K(x)$ 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?
9.3 Private Constrained PRF: Construction

Dual-BLMR

We will start with the following PRF (that we will call dual-BLMR).

$$\text{PRF}_{S_{i,b}}(x_1 x_2 \ldots x_\ell) = [S_{1,x_1} S_{2,x_2} \ldots S_{\ell,x_\ell} A]_p \pmod q$$

where $S_{i,b}$ are (secret) square matrices with random small entries, and $A$ is a (public) uniformly random matrix mod $q$.

(This is closely related to the BLMR construction which if you recall looks as follows:

$$\text{PRF}_{S}(x_1 x_2 \ldots x_\ell) = [S A_{1,x_1} G^{-1}(A_{2,x_2}) \ldots G^{-1}(A_{\ell,x_\ell})]_p \pmod q$$

where the $A_{i,x_i}$ are random matrices. This is actually a slight generalization of BLMR using a secret matrix $S$ and 2$\ell$ 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.

$$S_{1,0} A + E_{1,0}, \quad S_{2,0} A + E_{2,0}, \ldots, \quad S_{\ell,0} A + E_{\ell,0}$$

This certainly hides the $S_{i,b}$ (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.

$$B_{1,0} := S_{1,0} A + E_{1,0}, \quad B_{2,0} := S_{2,0} A + E_{2,0}, \ldots, \quad B_{\ell,0} := S_{\ell,0} A + E_{\ell,0}$$

$$B_{1,1} := S_{1,1} A + E_{1,1}, \quad B_{2,1} := S_{2,1} A + E_{2,1}, \ldots, \quad B_{\ell,1} := S_{\ell,1} A + E_{\ell,1}$$

It’s not clear what you get by multiplying, say, $B_{1,0}$ with $B_{2,0}$. What we need is to enable some sort of homomorphic multiplication. Let’s take inspiration from GSW13.

$$B_{1,0} := S_{1,0} A + E_{1,0}, \quad B_{2,0} := S_{2,0} A + E_{2,0}, \ldots, \quad B_{\ell,0} := S_{\ell,0} A + E_{\ell,0}$$

$$B_{1,1} := S_{1,1} A + E_{1,1}, \quad B_{2,1} := S_{2,1} A + E_{2,1}, \ldots, \quad B_{\ell,1} := S_{\ell,1} A + E_{\ell,1}$$
Here is how GSW enables multiplication.

\[
B_{1,0} \cdot A^{-1}(B_{2,0}) = S_{1,0}A + E_{1,0} \cdot A^{-1}(S_{2,0}A + E_{2,0}) \\
= S_{1,0}A \cdot A^{-1}(S_{2,0}A + E_{2,0}) + E_{1,0}A^{-1}(B_{2,0}) \\
\approx S_{1,0} \cdot (S_{2,0}A + E_{2,0}) \\
= S_{1,0}S_{2,0}A + S_{1,0}E_{2,0} \approx 0 \\
\approx S_{1,0}S_{2,0}A
\]

where \(A^{-1}(B)\) is a matrix \(R\) with small random entries such that \(AR = B \pmod{q}\).

Continuing along these lines, you can compute the PRF on all inputs if you can compute \(A^{-1}(B_{i,b})\). Since computing \(A^{-1}(\cdot)\) requires the trapdoor of \(A\), the owner of the PRF key can precompute all these matrices

\[
D_{i,b} := A^{-1}(S_{i,b}A + E_{i,b})
\]

and release them.

\[
D_{1,0} := A^{-1}(S_{1,0}A + E_{1,0}), \quad D_{2,0} := A^{-1}(S_{2,0}A + E_{2,0}), \quad \ldots, \quad D_{\ell,0} := A^{-1}(S_{\ell,0}A + E_{\ell,0}) \\
D_{1,1} := A^{-1}(S_{1,1}A + E_{1,1}), \quad D_{2,1} := A^{-1}(S_{2,1}A + E_{2,1}), \quad \ldots, \quad D_{\ell,1} := A^{-1}(S_{\ell,1}A + E_{\ell,1})
\]

How about security? This is tricky because we would like to argue that each \(S_{i,b}A + E_{i,b}\) is pseudorandom, invoking LWE. But the constrained key depends on the trapdoor for \(A\) in the presence of which LWE is not hard.
In general, each location $i$ of a branching program for a circuit, improved later by Chen-Vaikuntanathan-Wee’18.

Let’s now incorporate matrix BPs into the constrained key. The construction is due to Canetti and Chen (2017), and we think of $A_\ell = A$.

We have that

$$A_0 D_{1,0} D_{2,0} \cdots D_{\ell,x_\ell} \approx S_{1,x_1} S_{2,x_2} \cdots S_{\ell,x_\ell} A \pmod{q}$$

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 $2\ell$ matrices $M_{i,b} \in \{0,1\}^{w \times w}$ (where think of $w$ as a constant) and a vector $v \in \{0,1\}^{1 \times w}$ where $\ell \geq k$:

$$v = (1 \ 0 \ \ldots \ 0), \quad M_{1,0} \ M_{2,0} \ \ldots \ M_{\ell,0} \ M_{1,1} \ M_{2,1} \ \ldots \ M_{\ell,1}$$

In general, each location $i$ (corresponding to a pair of full-rank matrices $M_{i,0}$ and $M_{i,1}$) 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

$$u := v M_{1,x_1} M_{2,x_2} \cdots M_{\ell,x_\ell}$$

$u$ is either $(1 \ 0 \ \ldots \ 0)$ or $(0 \ 1 \ \ldots \ 0)$. If $u[1] \neq 0$, 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-$\ell$ matrix branching program for $f$ first.

The constrained key is

$$\hat{v} A_0, \quad D_{1,0} := A_0^{-1} (\hat{S}_{1,0} A_1 + E_{1,0}), \quad D_{2,0} := A_1^{-1} (\hat{S}_{2,0} A_2 + E_{2,0}), \ \ldots, \quad D_{\ell,0} := A_{\ell-1}^{-1} (\hat{S}_{\ell,0} A_\ell + E_{\ell,0})$$

$$D_{1,1} := A_0^{-1} (\hat{S}_{1,1} A_1 + E_{1,1}), \quad D_{2,1} := A_1^{-1} (\hat{S}_{2,1} A_2 + E_{2,1}), \ \ldots, \quad D_{\ell,1} := A_{\ell-1}^{-1} (\hat{S}_{\ell,1} A_\ell + E_{\ell,1})$$
where $\hat{S}_{i,b} = M_{i,b} \otimes S_{i,b}$ is a tensor product of the two matrices and $\hat{v} = \mathbf{v} \otimes \mathbf{I}$.

The key property of tensor products is the following associative property.

$$(A \otimes B) \cdot (C \otimes D) = AC \otimes BD$$

where $\otimes$ is the tensor product and $\cdot$ is matrix multiplication.

(Note that the $\hat{S}_{\ell,i}$ are now $nw \times nw$ matrices and $A_i$ have to be corresponding larger, i.e., $nw \times m$ for a large enough $m$.)

We have that

$$\hat{v} A_0 D_{1,x_1} D_{2,x_2} \cdots D_{\ell,x_\ell} \approx \hat{v} S_{1,x_1} S_{2,x_2} \cdots S_{\ell,x_\ell} A_\ell \mod q = \left( (v \prod M_{i,x_i}) \otimes (\prod S_{i,x_i}) \right) A_\ell$$

We know that $v \prod M_{i,x_i}$ is either $(1\ 0\ 0\ \ldots\ 0)$ or $(0\ 1\ 0\ \ldots\ 0)$. So the entire product is close to $\prod S_{i,x_i} A_\ell^{(1)}$ if $f(x) = 1$ or $\prod S_{i,x_i} A_\ell^{(2)}$ if $f(x) = 0$. where

$$A_\ell := \begin{bmatrix} A_\ell^{(1)} \\ A_\ell^{(2)} \\ \vdots \\ A_\ell^{(w)} \end{bmatrix}$$

So, letting $A_\ell^{(1)} := A$ in the definition of the PRF finishes the construction.

We will give this monster a compact name:

$$\hat{v} A_0, D_{1,0} := A_0^{-1}(\hat{S}_{1,0} A_1 + E_{1,0}), \quad D_{2,0} := A_1^{-1}(\hat{S}_{2,0} A_2 + E_{2,0}), \quad \ldots, \quad D_{\ell,0} := A_{\ell-1}^{-1}(\hat{S}_{\ell,0} A_\ell + E_{\ell,0})$$
$$D_{1,1} := A_0^{-1}(\hat{S}_{1,1} A_1 + E_{1,1}), \quad D_{2,1} := A_1^{-1}(\hat{S}_{2,1} A_2 + E_{2,1}), \quad \ldots, \quad D_{\ell,1} := A_{\ell-1}^{-1}(\hat{S}_{\ell,1} A_\ell + E_{\ell,1})$$

Call this a GGH15 chain (after the inventors Gentry, Gorbunov and Halevi) for the program $f$, secrets $S_{i,b}$ and the final matrices $A_\ell^{(1)} := A^{(1)}$ and $A_\ell^{(2)} := A^{(2)}$.

On input $x$, we can compute

$$\approx S_x A^{(2-f(x))}$$

where now and henceforth $S_x := \prod S_{i,x_i}$.

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.

$$\hat{v} A_0, D_{1,0} := A_0^{-1}(\hat{S}_{1,0} A_1 + E_{1,0}), \quad D_{2,0} := A_1^{-1}(\hat{S}_{2,0} A_2 + E_{2,0}), \quad \ldots, \quad D_{\ell,0} := A_{\ell-1}^{-1}(\hat{S}_{\ell,0} A_\ell + E_{\ell,0})$$
$$D_{1,1} := A_0^{-1}(\hat{S}_{1,1} A_1 + E_{1,1}), \quad D_{2,1} := A_1^{-1}(\hat{S}_{2,1} A_2 + E_{2,1}), \quad \ldots, \quad D_{\ell,1} := A_{\ell-1}^{-1}(\hat{S}_{\ell,1} A_\ell + E_{\ell,1})$$

1. Observe first that

$$\hat{S} A + E$$

is pseudorandom by LWE where $\hat{S} = M \otimes S$ where $S$ is a random small matrix and $M$ is any full-rank matrix. (Can you see what happens if $M$ 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 $A$, 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 $A_\ell$ 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 $\hat{S}_{\ell,b}A_\ell + E_{\ell,b}$ by uniformly random and independent matrices $U_{\ell,b}$. This is by an invocation of LWE.

- Second, replace $A_{\ell-1}^{-1}(U_{\ell,b})$ by a Gaussian matrix $D_{\ell,b}$. 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 $A_{\ell-1}$! 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 $O$ 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,

$$\forall A, \exists S, \forall \pi : \{0, 1\}^* \rightarrow \{0, 1\} \text{ and } \forall \text{ programs } P :$$

$$\Pr[A(O(P) = \pi(P))] \approx \Pr[S^P(1^n) = \pi(P)]$$

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 $\alpha, \beta$ 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 \( \alpha \) 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 \( \alpha \) and arbitrary \( \beta \) 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 \((\text{KeyGen}, \text{Enc}, \text{Dec})\). Modify it to \((\text{KeyGen}', \text{Enc}', \text{Dec}')\) that works as follows.

- **KeyGen’** generates a \((pk, sk)\) pair from KeyGen and lets
  
  \[ pk' = (pk, \text{LO}(F_{f_{sk}, \alpha, sk})) \] and \[ sk' = (sk, \alpha) \]

  for a random \( \alpha \). Here, \( f_{sk} \) takes a bunch of ciphertexts and decrypts them using \( sk \). That is, if you feed an encryption of \( \alpha \) 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 \( 2\lambda \) matrices \( A^{(j,b)} \) with \( j \in [\lambda] \) and \( b \in \{0,1\} \) such that

  \[ \sum_j A^{(j,\alpha_j)} = 0 \pmod q \]
• generate \( \lambda \) GGH15 chains, one for the function \( f_j \) that outputs the \( j \)-th bit of \( f \), and the final matrix pair \((A^{(j,0)}, A^{(j,1)})\). All GGH15 chains share the same \( S \) matrices.

For correctness, note that we can compute

\[
\approx S_x A^{(j,f_j(x))}
\]

for all \( j \). Sum them up to get

\[
\approx S_x \sum_j A^{(j,f_j(x))}
\]

This sum is \( \approx 0 \) 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 \( \alpha \) is random (or has min-entropy) can be used to show using Leftover hash lemma that the matrices \( A^{(j,b)} \) are truly random (as if there were no additive constraint on them.)

2. Now, we have \( \ell \) GGH15 chains with the same \( S \) matrices but random and independent \( A \) 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
CHAPTER 11

Oblivious Transfer and Multiparty Computation
Zero Knowledge Proofs
Quantum Computing and Lattices