CS 395T: Topics in Cryptography

Topic: Lattice-Based Cryptography

David Wu

Compilation of scribe notes from Spring 2022

Warning: These notes are a direct compilation of scribe notes from the Spring 2022 offering of the course. They have *not* been checked for completeness or correctness!

Contents

CS 395T: Topics in Cryptography January 19, 2022

Lecture 1: Introduction to Lattices

Lecturer: David Wu *Scribe: David Wu*

1.1 Why Lattices?

The focus of this course is on *lattice-based cryptography*. There are several motivations to study lattice-based cryptography:

- **Conjectured post-quantum resilience.** Many lattice-based problems are conjectured to be hard not only for classical computers, but also for quantum computers. They are one of the leading candidates for the ongoing NIST standardization efforts.
- **Security based on worst-case hardness.** Cryptography typically relies on average case hardness (i.e., a random instance of a problem drawn from some distribution of instances is hard to solve). This is a much stronger requirement than the notion of *worst-case* hardness encountered for instance in the study of NP-completeness, where a problem is "hard" if there exists *any* hard instance. Problems that are hard in the worst case can often be much easier in the average case, especially if we consider distributions over structured instances.

  A distinctive feature of lattice-based cryptography is that it allows us to base cryptography on a *worst-case* complexity assumption. Namely, a sequence of works have shown that certain lattice problems (e.g., short integer solutions or learning with errors) are hard on average as long as some closely related lattice problems (e.g., approximate shortest vector, approximate shortest independent vector) are hard in the worst case. This means that lattice-based cryptosystems are secure unless *all* instances of certain lattice problems are solvable in polynomial time.
- **Enables advanced cryptographic capabilities.** Lastly, lattices provide a rich algebraic structure and enable a broad range of advanced cryptographic capabilities. These include notions like fully homomorphic encryption, homomorphic signatures, and functional encryption. Until very recently, lattice-based cryptography has provided the *only* such realizations of these functionalities.

1.2 Lattice Definitions and Problems

We begin with some basic definitions and describe some of the key problems studied in lattice-based cryptography. Unless otherwise indicated, for a vector $\mathbf{v} \in \mathbb{R}^n$, we will write $\|\cdot\|$ to denote the ℓ_2 norm (i.e., the Euclidean norm): $\|\mathbf{v}\| = \left(\sum_{i\in[n]} v_i^2\right)^{1/2}$. Note that the choice of ℓ_p norm does not significantly impact the hardness of lattice problems [Pei08].

Definition 1.1 (Lattices)**.** An n-dimensional lattice $\mathcal{L} \subseteq \mathbb{R}^n$ is a discrete additive subspace of $\mathbb{R}^n$:

- **Discrete:** For every $\mathbf{x} \in \mathcal{L}$, there exists a neighborhood around it such that contains $\mathbf{x}$ and no other lattice point. More formally for every $\mathbf{x} \in \mathcal{L}$, there exists $\varepsilon > 0$ such that $B_\varepsilon(\mathbf{x}) \cap \mathcal{L} = \{\mathbf{x}\}$, where $B_\varepsilon(\mathbf{x}) = \{\mathbf{y} \in \mathbb{R}^n : \|\mathbf{x} - \mathbf{y}\| \leq \varepsilon\}$. In analytical terms, every subset $S \subseteq \mathcal{L}$ is an *open* set.
- **Additive subspace:** For all $\mathbf{x}, \mathbf{y} \in \mathcal{L}$, both $-\mathbf{x} \in \mathcal{L}$ and $\mathbf{x} + \mathbf{y} \in \mathcal{L}$.

An example of an n-dimensional lattice is $\mathbb{Z}^n$ (i.e., the set of n-dimensional vectors with integer coefficients). Another example is the "q-ary" lattice $q\mathbb{Z}^n$ where $q \in \mathbb{N}$ (i.e., the set of n dimensional vectors where each coordinate is an integer multiple of q).

Lattice basis. While most (non-trivial) lattices are infinite, they are *finitely-generated* by taking *integer* linear combinations of a small number of basis vectors $\mathbf{B} = [\mathbf{b}_1 \mid \cdots \mid \mathbf{b}_k]$ where each $\mathbf{b}_i \in \mathbb{R}^n$ and $\mathbf{b}_1, \ldots, \mathbf{b}_k$ are linearly independent (over $\mathbb{R}^n$). We write $\mathcal{L}(\mathbf{B})$ to denote the lattice generated by $\mathbf{B}$:

$$\mathcal{L}(\mathbf{B}) = \mathbf{B} \cdot \mathbb{Z}^k = \left\{ \sum_{i \in [k]} \alpha_i \mathbf{b}_i : \alpha_i \in \mathbb{Z} \text{ for all } i \in [k] \right\}.$$

We refer to k as the *rank* of the matrix. The basis is not *unique*. For example, the vectors $[1, 0]^\mathsf{T}$ and $[0, 1]^\mathsf{T}$ form a basis for $\mathbb{Z}^2$, as do the vectors $[7, 5]^\mathsf{T}$ and $[11, 8]^\mathsf{T}$. In fact, the choice of basis plays an important role in the hardness of many lattice problems: problems are easy if we have a "good" basis and hard if we have a "bad" basis. As we will see later in this course, oftentimes a "good" basis can be used as a trapdoor in a cryptographic construction while a "bad" basis can be used as the public key. We typically measure the quality of a basis by the norm of the vectors contained in the basis.

Minimum distance. A central quantity of interest in the study of lattices is the minimum distance (i.e., the norm of the shortest non-zero vector in a lattice).

Definition 1.2 (Minimum Distance)**.** Let $\mathcal{L}$ be an n-dimensional lattice. Then the *minimum distance* of $\mathcal{L}$, denoted by $\lambda_1(\mathcal{L})$ is the length of the shortest non-zero vector:

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

More generally, we write $\lambda_i(\mathcal{L})$ to denote the *i^{th} successive minimum of $\mathcal{L}$*: the minimum $r \in \mathbb{R}$ such that $\mathcal{L}$ contains i linearly independent basis vectors of norm at most r.

Computational problems. We now define several computational problems on lattices:

- **Shortest vector problem (SVP):** Given a basis $\mathbf{B}$ for an n-dimensional lattice $\mathcal{L} = \mathcal{L}(\mathbf{B})$, find the shortest non-zero vector $\mathbf{v} \in \mathcal{L}$: namely, find $\mathbf{v} \in \mathcal{L}$ such that $\|\mathbf{v}\| = \lambda_1(\mathcal{L})$.
- **Approximate shortest vector problem (SVP_γ):** Given a basis $\mathbf{B}$ of an n-dimensional lattice $\mathcal{L} = \mathcal{L}(\mathbf{B})$, find a non-zero vector $\mathbf{v} \in \mathcal{L}$ such that $\|\mathbf{v}\| \leq \gamma(n) \cdot \lambda_1(\mathcal{L})$.
- **Decisional approximate shortest vector problem (GapSVP_γ):** Given a basis $\mathbf{B}$ of an n-dimensional lattice $\mathcal{L} = \mathcal{L}(\mathbf{B})$, decide if $\lambda_1(\mathcal{L}) \leq 1$ or if $\lambda_1(\mathcal{L}) \geq \gamma(n)$.
- **Approximate shortest independent vectors problem (SIVP_γ):** Given a basis $\mathbf{B}$ of a full-rank n-dimensional lattice $\mathcal{L} = \mathcal{L}(\mathbf{B})$, output a set $\mathbf{B}' = \{\mathbf{b}'_1, \ldots, \mathbf{b}'_n\} \subseteq \mathcal{L}$ of linearly independent basis vectors where $\|\mathbf{b}'_i\| \leq \gamma(n) \cdot \lambda_n(\mathcal{L})$ for all $i \in [n]$.

The main problems we rely on in lattice-based cryptography are the short integer solutions problem (SIS) and the learning with errors problem (LWE). Hardness of these problems can be based on the hardness of the GapSVP and SIVP problems (with appropriately-chosen approximation factors). To date, we do not know how to base cryptography on the search versions of SVP (i.e., SVP or SVP_γ).

Complexity of GapSVP. We now state some of the known complexity results on GapSVP. A similar set of complexity results are known for the SIVP problem.

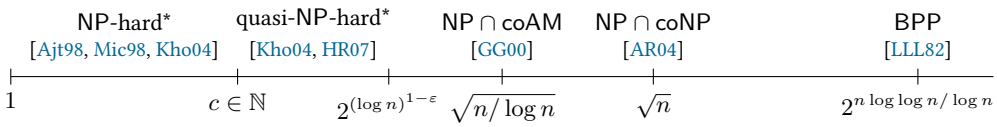


Figure 1.1: Complexity of GapSVP_γ as a function of the approximation factor γ (and lattice dimension n). We write $c > 1$ to denote any constant greater than 1 and $\varepsilon > 0$ to denote any constant greater than 0. We write "NP-hard*" to denote NP-hardness under a *randomized* reduction (specifically, a reduction that maps NO instances to NO instances with probability 1 and YES instances to YES instances with probability at least $2/3$; an algorithm for an NP-hard* problem would imply that NP = RP). We write "quasi-NP-hard*" to denote NP-hardness under a randomized quasi-polynomial time reduction (i.e., a reduction that runs in time $2^{\mathsf{polylog}(n)}$).

We now describe some open problems for understanding the complexity of lattice problems:

- **Derandomizing reductions.** The known NP-hardness results for GapSVP are under randomized reductions. Can we give a *deterministic* reduction for *some* gap? For the particular case of the ℓ_∞ norm, Dinur showed that approximating GapSVP to a *nearly polynomial* factor $n^{c/\log\log n}$ for some constant $c > 0$ is NP-hard [Din00].
- **Polynomial-time reductions.** Can we give polynomial-time (randomized) reductions for super-constant approximation factors? Existing reductions [Kho04, HR07] require super-polynomial time.

An open problem in cryptography is to reduce cryptography to an NP-hard problem. While problems like GapSVP and SIVP are NP-hard for certain approximation factors (under randomized reductions), these regimes are *not* known to be useful for cryptography. To date, *all* lattice-based cryptographic constructions rely on polynomial approximation factors $\gamma(n) \geq n$. As shown in Fig. 1.1, when $\gamma(n) \geq \sqrt{n}$, $\mathsf{GapSVP}_\gamma \in \mathsf{NP} \cap \mathsf{coNP}$. Thus it is unlikely that the lattice problems relevant to cryptography are NP-hard.

Algorithms for SVP. The main class of algorithms for solving lattice problems are based on the lattice reduction algorithms of Lenstra, Lenstra, and Lovász [LLL82]. These are polynomial-time algorithms and solve SVP to subexponential approximation factors $\gamma = 2^{\Theta(n \log\log n/\log n)}$. To achieve a polynomial approximation factor, the current best algorithms all run in exponential time (e.g., $2^{\Theta(n)}$) [WLW15, ALNS20, ALS21]. We can also achieve trade-offs between the approximation factor and the running time: namely, we can solve GapSVP_γ in time $2^{\tilde{\Theta}(n/\log\gamma)}$ [Sch87], where $\tilde{\Theta}(\cdot)$ suppresses constant and polylogarithmic terms. These tradeoffs are primarily interesting when considering super-polynomial approximation factors. These algorithms also represent the state-of-the-art when considering *quantum* algorithms.

CS 395T: Topics in Cryptography January 24, 2022

Lecture 2: Short Integer Solutions

Lecturer: David Wu *Scribe: Niels Kornerup*

2.1 Recap of lecture one

Definition 2.1. A lattice $\mathcal{L}$ is the set of integer linear combinations of some basis $\mathcal{B}$.

$$\mathcal{L} = \mathcal{L}(\mathcal{B}) = \mathcal{B} \cdot \mathbb{Z}^k = \{x_i b_i : x_i \in \mathbb{Z}, b_i \in \mathcal{B}\}$$

where the $b_i \in \mathbb{R}^n$ and they are linearly independent.

Some examples of lattices include $\mathbb{Z}^n$ and $q\mathbb{Z}^n$ where $q \in \mathbb{Q}$.

Definition 2.2. $\lambda_1(\mathcal{L})$ is the norm of the shortest vector in $\mathcal{L}$.

$$\lambda_1(\mathcal{L}) \coloneqq \min_{v \in \mathcal{L} \setminus \{0\}} \|v\|_2$$

λ_i is the smallest value of $r \in \mathbb{R}$ such that $\mathcal{L}$ contains i linearly independent vectors $v_1, \ldots, v_i$ where $\|v_j\| \leq r$.

As an example, for the lattice $\mathcal{L} = 2\mathbb{Z}^n$, $\lambda_1(\mathcal{L}) = \lambda_n(\mathcal{L}) = 2$. We will often use either the L2 or the L∞ norm in this class, but we will see later that this choice doesn't matter too much. For this lecture, we are using the L2 norm.

2.2 Computational problems over lattices

Definition 2.3 (Shortest vector problem (SVP))**.** Given a basis $\mathcal{B}$ of an n-dimensional lattice, find $v \in \mathcal{L}$ such that $\|v\| = \lambda_1(v)$

The shortest vector problem is not known to be in NP, but it is NP-hard. We also consider an approximation to SVP parameterized by the approximation factor $\gamma = f(n) \geq 1$.

Definition 2.4 (Approximate shortest vector problem (SVP$_\gamma$))**.** Given $\mathcal{B}$ for lattice $\mathcal{L} = \mathcal{L}(\mathcal{B})$, find a vector v such that $\|v\| \leq \gamma\lambda_1(\mathcal{L})$.

When $\gamma = 1$, the above is the shortest vector problem. For larger γ, we are looking for an approximate solution. It is not obvious how to solve the above two problems in NP, since we are not given any information about the value of $\lambda_1(\mathcal{L})$. Instead, we can construct a promise problem that can be solved in NP.

Definition 2.5 (Decisional approximate SVP (GapSVP$_\gamma$))**.** Given a basis $\mathcal{B}$ for n-dimensional lattice $\mathcal{L} = \mathcal{L}(\mathcal{B})$, decide whether:

1. $\lambda_1(\mathcal{L}) \leq 1$
2. $\lambda_1(\mathcal{L}) \geq \gamma$

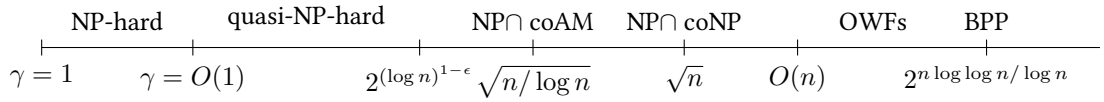
under the promise that one of these cases hold for $\mathcal{L}$.

One witness to this problem is a vector $v \in \mathcal{L}$ such that $\|v\| < \gamma$. We consider one final computational problem.

Definition 2.6 (Approximate shortest independent vector problem (SIVP$_\gamma$))**.** Given a full-rank n-dimensional lattice $\mathcal{L} = \mathcal{L}(\mathcal{B})$, output a set of linearly independent vectors $b'_1, \ldots b'_n \in \mathcal{L}$ such that $\|b'_i\| \leq \gamma\lambda_n(\mathcal{L})$.

2.3 Hardness of GapSVP$_\gamma$

The hardness of GapSVP$_\gamma$ depends significantly on the value of γ. If γ is large, then sampling of elements in $\mathcal{L}$ is likely to reveal one with a norm less than γ. This plot shows how the hardness of GapSVP$_\gamma$ changes with γ.



When $\gamma = O(1)$ we have NP-hardness under a randomized reduction that maps no instances to no instances with probability 1 and maps yes instances to yes instances with probability $\frac{2}{3}$. Thus a polynomial time algorithm for GapSVP$_{O(1)}$ implies that NP=RP (one sided error polynomial time).

It is unlikely that when $\gamma = n^\epsilon$ this problem is NP hard, since that will collapse the polynomial hierarchy. We have constructions for one way functions under the assumption that $\gamma = O(n)$ is hard for polynomial time algorithms and we have an efficient randomized algorithm for GapSVP$_{2^{n \log\log n/\log n}}$.

It is unlikely that we can use lattices to develop NP-hardness based cryptography, since our problems are already in $NP \cap coNP$ in the settings where we get cryptography.

2.3.1 Open problems

1. Finding a non-randomized reduction when γ is sub-polynomial. We have proven NP-hardness for GapSVP$_\gamma$ for $\gamma = 2^{(\log n)^{1-\epsilon}}$ under the L∞ norm. This is open for the L2 norm.
2. Finding a polynomial time reduction for super-constant approximate factors.

2.3.2 Algorithms for SVP

Lenstra-Lenstra-Lovasz (LLL) is a class of lattice reduction algorithms. They are closely related to Gram–Schmidt normalization and they run in polynomial time for a $\gamma = 2^{n \log\log n/\log n}$ approximation. All known poly(n) approximations to GapSVP run in time $2^{\Theta(n)}$. We also know how to obtain tradeoffs between the approximation factor γ and our running time. Specifically we can solve GapSVP$_\gamma$ in time $2^{\Theta(n/\log\gamma)}$. Most of these algorithms require $2^{\Theta(n)}$ space, but recent works have tried to reduce the space complexity. Right now we do not know any way that quantum algorithms can create asymptotic improvements for these problems other than a Grover's quadratic speedup.

2.4 Short Integer Solutions (SIS) problem

In this course we will base hardness on average-case assumptions that are implied by GapSVP / SIVP.

Definition 2.7 (Short integer solutions (SIS))**.** $SIS_{n,m,q,\beta}$ is defined with respect to lattice dimension n, number of instances m, modulus q, and noise bound β. Given a random $A \in \mathbb{Z}_q^{n\times m}$, no efficient adversary can find a non-zero $x \in \mathbb{Z}^m$ such that $Ax \equiv 0(\mathrm{mod} q)$ and $\|x\|_2 \leq \beta$.

In general we want that $\beta < q$m as otherwise $x = (q, 0, \ldots, 0)$ would always be a solution. When $m = \Omega(n \log q)$ and $\beta > \sqrt{m}$, we know that a solution exists. This follows from the fact that there are 2^m possible $\{0, 1\}$ valued inputs for x while there are only q^n outputs produced by these inputs. Thus two distinct inputs must map to the same output and their difference maps to zero.

2.4.1 SIS as a lattice problem

We can view SIS as average-case SVP on a lattice defined by $A \in \mathbb{Z}_q^{n\times m}$. We define:

$$\mathcal{L}^\perp(A) = \{x \in \mathbb{Z}^m : Ax \equiv 0(\mathrm{mod} q)\}$$

Then SIS is approximate SVP in $\mathcal{L}^{\perp}(A)$. In coding theory terms, A is the parity check matrix.

Theorem 2.8. *For any $m = \mathsf{poly}(n)$, any $\beta > 0$, and sufficiently large $q > \beta \cdot \mathsf{poly}(n)$, there is a probabilistic poly-time (PPT) reduction from solving GapSVP$_\gamma$ or SIVP$_\gamma$ in the worst-case to solving SIS$_{n,m,q,\beta}$ with non-negligible probability.*

There are some tightness results about the above theorem:

1. Micciancio and Regev [MR04]: $\gamma = \beta \cdot \tilde{O}(\sqrt{n})$ with $q = \beta\tilde{O}(n\sqrt{m}) = n^{O(1)}$
2. Gentry-Peikert-Vaikuntanathan [GPV08]: same γ, but allows smaller $q = \beta \cdot \tilde{O}(\sqrt{n})$

We should view this as saying that the hardness of GapSVP$_\gamma$ / SIVP$_\gamma$ in the worst case implies that SIS is hard on average.

CS 395T: Topics in Cryptography January 26, 2022

Lecture 3: Cryptographic Constructions from SIS

Lecturer: David Wu *Scribe: Steven Cheng*

3.1 Collision-Resistant Hash Functions from SIS

Using the SIS assumption, we can begin to construct various cryptographic primitives. We first examine the construction of a collision-resistant hash function (CRHF).

Definition 1.1 (Collision-Resistant). A keyed hash family $H : K \times X \to Y$ is collision-resistant if it satisfies the following properties:

- **Compressing:** The hash function compresses the number of inputs: $|Y| < |X|$.
- **Collision-Resistant:** All efficient adversaries $\mathcal{A}$ can only find collisions with negligible probability:

$$Pr\left[k \xleftarrow{R} K; (x, x') \leftarrow \mathcal{A}(k) : H(k, x) = H(k, x') \wedge x \neq x'\right] = \mathsf{negl}(\lambda)$$

CRHF from SIS. We can derive a CRHF directly from SIS. Let $H : \mathbb{Z}_q^{n \times m} \times \{0,1\}^m \to \mathbb{Z}_q^n$, where we set $m > \lceil n \log q \rceil$, be defined as $H(A, x) = Ax \pmod q$. We can prove that H is collision resistant:

- **Compressing:** $|Y| = |\mathbb{Z}_q^n| = q^n$ and $|X| = |\{0,1\}^m| = 2^m$. Taking the log of both, we get $n \log q < m$, which is true by definition.
- **Collision Resistant:** Suppose an adversary finds $x_1, x_2 \in \{0,1\}^m$ where $x_1 \neq x_2$ and $Ax_1 = Ax_2$. Rearranging, we get $A(x_1 - x_2) = 0$, so $x_1 - x_2$ is a solution. Furthermore, $x_1 - x_2 \in \{-1, 0, 1\}^m$, so $||x_1 - x_2||_2 \leq m$, so we have found a solution for $\mathsf{SIS}_{n,m,q,\beta}$ (where $\beta = \lceil n \log q \rceil$). Thus, collision resistance follows from the SIS assumption.

Local updates. Suppose we have a public hash $H(A, x)$, where x is a bitstring: $x \in \{0,1\}^m$. We want to update $x \mapsto x'$ where x and x' only differ on a single index, i^*. Observe that

$$H(A, x) = A \cdot x = \begin{bmatrix} | & | & \cdots & | \\ a_1 & a_2 & \cdots & a_m \\ | & | & \cdots & | \end{bmatrix} \cdot \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_m \end{bmatrix} = \sum_{i=1}^{m} a_i x_i$$

$$H(A, x') = \sum_{i=1}^{m} a_i x'_i = \sum_{i=1}^{m} a_i x_i + \sum_{i=1}^{m} a_i (x'_i - x_i) = H(A, x) + a_{i^*}(x'_{i^*} - x_{i^*})$$

Thus, we can easily find $H(A, x')$ by simply adding $a_{i^*}(x'_{i^*} - x_{i^*})$ to $H(A, x)$, removing the need to fully compute $A \cdot x'$. Local updates can provide crucial speedups in situations where only small parts of large databases get updated at a time, such as updating an entry in an address book.

3.2 Leftover Hash Lemma

Definition 2.1 (Universality). A keyed hash function $H : K \times X \to Y$ is ϵ-universal if, for all $x_0, x_1 \in X$ where $x_0 \neq x_1$,

$$Pr[k \xleftarrow{R} K : H(k, x_0) = H(k, x_1)] \leq \epsilon$$

When $\epsilon = 1/|Y|$, then we say it is universal.

Universality of SIS. We can prove that the SIS hash function is universal. Let $x_0, x_1 \in \{0,1\}^m$ with $x_0 \neq x_1$ be two arbitrary inputs such that $H(A, x_0) = H(A, x_1)$ for some $A \in \mathbb{Z}_q^{n \times m}$. Let $a_1, \ldots, a_n$ be the columns of A. Thus,

$$H(A, x_0) - H(A, x_1) = Ax_0 - Ax_1 = A(x_0 - x_1) = \sum_{i=1}^{m} a_i(x_{0,i} - x_{1,i}) = 0$$

Since $x_0 \neq x_1$, they must differ on some index j. Observe that $x_{0,j} - x_{1,j} \in \{-1, 1\}$, so it is always invertible mod q. Thus, the above relation holds only if

$$a_j = (x_{0,j} - x_{1,j})^{-1} \sum_{i \neq j} a_i(x_{0,i} - x_{1,i})$$

Since all columns of A are sampled independently, then the right side of the relation is entirely independent of a_j. Viewing each element of a_j, there is a $\frac{1}{|q|}$ chance that it matches the right side. Thus, the probability of the relation holding is:

$$\begin{aligned}
&Pr[A \xleftarrow{R} \mathbb{Z}_q^{n \times m} : A(x_0 - x_1) = 0] \\
=&Pr[a_1, \ldots, a_n \xleftarrow{R} \mathbb{Z}_q^n : a_j = (x_{0,j} - x_{1,j})^{-1} \sum_{i \neq j} a_i(x_{0,i} - x_{1,i})] \\
=&\frac{1}{q^n} = \frac{1}{|Y|}
\end{aligned}$$

Thus, the SIS hash function is universal.

Definition 2.2 (Guessing probability and min-entropy.) Let X be a random variable taking on values in a finite set S. The guessing probability of X is

$$\max_{s \in S} Pr[X = s]$$

Further, we define the min-entropy of X to be

$$H_\infty(X) = -\log \max_{s \in S} Pr[X = s]$$

Intuitively, if X has k bits of entropy, then X has at least 2^k possible values, with the most likely outcome appearing with probability at most 2^{-k}. In other words, we can think of X as roughly resembling a random k-bit string.

Definition 2.3 (Statistical distance.) Let D_0, D_1 be distributions with a common finite support S. The statistical distance between D_0 and D_1 is defined to be

$$\Delta(D_0, D_1) = \frac{1}{2} \sum_{s \in S} |Pr[t \xleftarrow{R} D_0 : t = s] - Pr[t \xleftarrow{R} D_1 : t = s]|$$

If D_0 and D_1 are ϵ-close (that is, $\Delta(D_0, D_1) \leq \epsilon$), then no adversary can distinguish between D_0 and D_1 with advantage better than ϵ. When ϵ is negligible, then D_0 and D_1 are statistically indistinguishable.

Note that this differs from computational indistinguishability, which states that no *efficient* adversary can distinguish between the distributions.

Leftover Hash Lemma. Let $H : K \times X \to Y$ be an ϵ-universal hash function. Suppose $x \in X$ is a random variable with b bits of entropy. Define the following two distributions:

$$D_0 : \quad k \xleftarrow{R} K, y \leftarrow H(k, x); \quad \text{output } (k, y)$$
$$D_1 : \quad k \xleftarrow{R} K, y \xleftarrow{R} Y; \quad \text{output } (k, y)$$

The Leftover Hash Lemma (LHL) states that the statistical distance between these two distributions is at most

$$\Delta(D_0, D_1) \leq \frac{1}{2} \sqrt{\frac{|Y|}{2^b} + (|Y|\epsilon - 1)}$$

When H is universal, then this reduces to

$$\Delta(D_0, D_1) \leq \frac{1}{2} \sqrt{\frac{|Y|}{2^b} + \left(|Y| \left(\frac{1}{|Y|}\right) - 1\right)} = \frac{1}{2} \sqrt{\frac{|Y|}{2^b}}$$

Randomness extractor. LHL is often applied in settings where H is universal and $|Y| = 2^{b-2\lambda}$:

$$\Delta(D_0, D_1) \leq \frac{1}{2} \sqrt{\frac{|Y|}{2^k}} = \frac{1}{2} \cdot \frac{1}{2^\lambda}$$

In this setting, the distance between distributions is negligible, so $(k, H(k, x))$ is statistically indistinguishable from (k, y) where $y \xleftarrow{R} Y$. This is an example of a randomness extractor, where we are able to take the randomness from a non-uniform but random variable x, and produce from it a *uniform* random value, $H(k, x)$. However, this extraction "costs" 2λ bits of entropy to perform.

We can apply randomness extractors to generate uniformly random cryptographic keys from non-uniform sources. For example, consider $H : \mathbb{Z}_2^{n \times m} \times \{0,1\}^m \to \mathbb{Z}_2^n$:

$$H(A, x) = Ax$$

Recall the DDH assumption, which tells us that g^{ab}, the result of a Diffie Hellman key exchange, looks random. Thus, we can use the binary representation of g^{ab} as a random but non-uniform variable with some min-entropy guarantee. Applying LHL, we extract the randomness from g^{ab} and produce $H(A, g^{ab})$, which is suitable as a uniform random key for symmetric encryption.

We can further apply LHL to the SIS hash function. For some random variable $v \in \{0,1\}^m$ which has a min-entropy guarantee (but does not need to be uniform), LHL tells us that $H(A, v) = Av$ is uniform random when $m > n \log q + 2\lambda$. Since n is usually used as the security parameter when dealing with lattices, $m = \Theta(n \log q)$ suffices.

Extending this via a hybrid argument, we can sample a random $R \xleftarrow{R} \{0,1\}^{m \times m}$, and obtain AR which is statistically close to a uniform random variable over $\mathbb{Z}_q^{n \times m}$. This result will be useful in later constructions.

3.3 Commitments from SIS

Definition 3.1 (Commitments). Recall that a commitment scheme consists of two functions:

- $\text{Setup}(1^\lambda) \to \text{crs}$: Samples a common reference string from the security parameter, λ.
- $\text{Commit}(\text{crs}, \mu; r) \to \sigma$: Commits to a message μ using the randomness r.

To open a commitment, we simply provide (μ, r), and a verifier can check that $\sigma = \text{Commit}(\text{crs}, \mu; r)$. For this to be secure, commitment schemes must also satisfy two properties:

- **Hiding.** Adversaries cannot distinguish between commitments of two different messages. That is,

$$\{(\text{crs}, \sigma) : \sigma \leftarrow \text{Commit}(\text{crs}, m_1)\} \approx \{(\text{crs}, \sigma) : \sigma \leftarrow \text{Commit}(\text{crs}, m_2)\}$$

- **Binding.** For a random common reference string, adversaries cannot find (m_1, r_1) and (m_2, r_2) such that $\text{Commit}(\text{crs}, m_1, r_1) = \text{Commit}(\text{crs}, m_2, r_2)$.

Commitments can be either statistically or computationally hiding / binding, depending on whether the adversaries are required to be efficient or not.

Commitments from SIS. We can construct the following commitment scheme from SIS where n, q be lattice parameters and $m = \Theta(n \log q)$.

- $\text{Setup}(1^\lambda)$: Sample $A_1, A_2 \xleftarrow{R} \mathbb{Z}_q^{n \times m}$. The common reference string is simply (A_1, A_2).
- $\text{Commit}((A_1, A_2), \mu; r)$, where $\mu, r \in \{0,1\}^m$: Output $\sigma = A_1 m + A_2 r = \begin{bmatrix} A_1 | A_2 \end{bmatrix} \begin{bmatrix} m \\ r \end{bmatrix}$.

We can prove the two necessary properties for the SIS commitment scheme:

- **Statistically Hiding.** If $m > 3n \log q$, then the scheme is *statistically* hiding. The proof follows from LHL. For $r \xleftarrow{R} \{0,1\}^m$, $A_2 r$ is statisically indistinguishable from a uniform random value of $\mathbb{Z}_q^n$ (by LHL), so $A_2 r$ acts as a one-time pad and entirely obscures $A_1 m$.
- **Computational Binding.** We can show that the scheme is computationally binding through a reduction to $\text{SIS}_{n, 2m, q, \sqrt{2m}}$. Thus, suppose there exists an efficient adversary $\mathcal{A}$ that can break the binding property. We use $\mathcal{A}$ to construct $\mathcal{B}$, an adversary for SIS:

1. The SIS challenger samples $A \xleftarrow{R} \mathbb{Z}_q^{n \times 2m}$ and sends it to $\mathcal{B}$.
2. $\mathcal{B}$ separates $A = [A_1 | A_2]$ where $A_1, A_2 \in \mathbb{Z}_q^{n \times m}$, and provides the common reference string (A_1, A_2) to $\mathcal{A}$.
3. $\mathcal{A}$ breaks the binding property and provides $\sigma \in \mathbb{Z}_q^n$ and two pairs $(m_1, r_1), (m_2, r_2)$.
4. $\mathcal{B}$ sends the vector $\begin{bmatrix} m_1 - m_2 \\ r_1 - r_2 \end{bmatrix}$ to the adversary, and the game ends.

Let $x = \begin{bmatrix} m_1 - m_2 \\ r_1 - r_2 \end{bmatrix}$, the vector that $\mathcal{B}$ answers with. If $\mathcal{A}$ succeeds, then $\begin{bmatrix} A_1 | A_2 \end{bmatrix} \begin{bmatrix} m_1 \\ r_1 \end{bmatrix} = \begin{bmatrix} A_1 | A_2 \end{bmatrix} \begin{bmatrix} m_2 \\ r_2 \end{bmatrix} = \sigma$. After rearranging, we find $\begin{bmatrix} A_1 | A_2 \end{bmatrix} \begin{bmatrix} m_1 - m_2 \\ r_1 - r_2 \end{bmatrix} = Ax = 0$. Furthermore, since $m_1, m_2, r_1, r_2 \in \{0,1\}^m$, then

$x \in \{-1, 0, 1\}^{2m}$ so it has norm $||x||_2 \leq \sqrt{2m}$. Finally, since the two messages that $\mathcal{A}$ provides must be different, then $m_1 - m_2 \neq 0$ and so x is non-zero. Thus, x is a valid SIS solution.

Parallels to Discrete Log. Recall Pedersen commitments from discrete log:

- Setup(1^λ): Generate a prime order group $G \leftarrow \text{GroupGen}(1^\lambda)$, and sample $g, h \xleftarrow{R} G$ as the common reference string.
- Commit$((g, h), \mu; r)$, where $\mu, r \in \mathbb{Z}^p$: Output $\sigma = g^\mu h^r$.

Observe that, in Pedersen commitments, we obscure the message and randomness using exponents, while with SIS commitments, we obscure the message and randomness through matrix multiplication:

Discrete Log		SIS
$g, h \xleftarrow{R} G$	$\longrightarrow$	$A_1, A_2 \xleftarrow{R} \mathbb{Z}_q^{n \times m}$
g^μ	$\longrightarrow$	$A_1 \mu$
h^r	$\longrightarrow$	$A_2 r$

Later on in this course, we will see many parallels between discrete log based systems and lattice-based systems.

Lecture 4: Lattice Trapdoors and Digital Signatures

Lecturer: David Wu *Scribe: Jeffrey Champion*

4.1 Commitments from SIS and the ISIS Problem

Construction 4.1. The following is a commitment scheme from SIS:
Setup(1^λ): $\mathbf{A}_1, \mathbf{A}_2 \xleftarrow{R} \mathbb{Z}_q^{n \times m}$, $\mathsf{crs} = (\mathbf{A}_1, \mathbf{A}_2)$
Commit($\mathsf{crs}, m; r$): $\sigma = \mathbf{A}_1 m + \mathbf{A}_2 r$

Claim 4.2. *Construction 1.1 is statistically hiding.*

Proof. By LHL, we know that $(\mathbf{A}_2, \mathbf{A}_2 r) \approx (\mathbf{A}_2, \mathbf{u})$. Since $\mathbf{A}_1, \mathbf{A}_2$ are independent

$$\{(\mathbf{A}_1, \mathbf{A}_2), (\mathbf{A}_1 m + \mathbf{A}_2 r)\} \approx \{(\mathbf{A}_1, \mathbf{A}_2), (\mathbf{A}_1 m + \mathbf{u})\} \equiv \{(\mathbf{A}_1, \mathbf{A}_2), \mathbf{u}\}.$$

□

Claim 4.3. *Construction 1.1 is computationally binding by* $\mathsf{SIS}_{n,2m,q,\sqrt{2m}}$.

Proof. Given the SIS challenge $\mathbf{A} = [\mathbf{A}_1 | \mathbf{A}_2]$ and an adversary $\mathcal{A}$ that breaks binding, adversary $\mathcal{B}$ sets $\mathsf{crs} = (\mathbf{A}_1, \mathbf{A}_2)$ and runs $\mathcal{A}$. On output $\sigma, m_1, r_1, m_2, r_2$, $\mathcal{B}$ outputs $\mathbf{x} = [(m_1 - m_2) | (r_1 - r_2)]^T$ to break SIS. Since each entry of $\mathbf{x}$ is in $\{-1, 0, 1\}$ and there are $2m$ entries we have $\|\mathbf{x}\| \leq \sqrt{2m}$ as desired. □

Note that the scheme parallels Pederson commitments from discrete log where the matrix multiplication and addition correspond to group exponentiation and multiplication, respectively.

Inhomogeneous SIS (ISIS): Given $\mathbf{A} \xleftarrow{R} \mathbb{Z}_q^{n \times m}$, $\mathbf{y} \xleftarrow{R} \mathbb{Z}_q^n$, find $\mathbf{x}$ such that $\mathbf{A}\mathbf{x} = \mathbf{y}$ and $\|\mathbf{x}\|_2 \leq \beta$.

This problem corresponds to finding a short vector in the coset of a lattice:

$$\mathcal{L}_{\mathbf{u}}^{\perp}(\mathbf{A}) := \mathbf{c} + \mathcal{L}^{\perp}(\mathbf{A}),$$

where $\mathbf{A}\mathbf{c} = \mathbf{u}$ and $\mathbf{c}$ is an arbitrary solution. We need this variant of SIS in order to make lattice trapdoors. For the rest of these notes, the default norm will be the ℓ_∞ norm, which is defined as $\|\mathbf{x}\|_\infty = \max_i |x_i|$.

4.2 Lattice Trapdoors

A lattice trapdoor function needs the following procedures:

- $\mathsf{TrapGen}(n, m, q, \beta) \to (\mathbf{A}, \mathsf{td}_{\mathbf{A}})$ where $\mathbf{A} \in \mathbb{Z}_q^{n \times m}$
- $f_{\mathbf{A}}(\mathbf{x})$: on input $\mathbf{x} \in \mathbb{Z}_q^m$, output $\mathbf{A}\mathbf{x} \in \mathbb{Z}_q^n$
- $f_{\mathbf{A}}^{-1}(\mathsf{td}_{\mathbf{A}}, y)$: on input $\mathbf{y} \in \mathbb{Z}_q^n$, output $\mathbf{x} \in \mathbb{Z}_q^m$ where $\mathbf{A}\mathbf{x} = \mathbf{y}$ and $\|\mathbf{x}\| \leq \beta$

We now define a "gadget matrix" as a useful tool in achieving trapdoor functions from lattices. There are several variations, but a useful choice is to set $\mathbf{G} = \mathbf{I}_n \otimes \begin{bmatrix} 1 & 2 & 4 & 8 & \cdots & 2^{\lfloor \log q \rfloor} \end{bmatrix}$, where $\otimes$ denotes the tensor product. Note that $\mathbf{G} \in \mathbb{Z}_q^{n \times m}$, where $m = n \log q$.

A simple but very useful observation is that both SIS and ISIS are both easy with respect to $\mathbf{G}$. For SIS, an example solution would be $\mathbf{v} = 2e_1 - e_2$, where e_i's refer to standard basis vectors. For ISIS, a solution would be

$\mathbf{x} = [\mathsf{BitDecomp}(y_1) \cdots \mathsf{BitDecomp}(y_n)]^T$. We refer to this particular solution as a function $G^{-1} : \mathbb{Z}_q^n \to \mathbb{Z}_q^m$ where $\mathbf{G} \cdot G^{-1}(\mathbf{y}) = \mathbf{y} \ \ (\forall \mathbf{y} \in \mathbb{Z}_q^n)$.

We now wish to find a gadget trapdoor for a given matrix $\mathbf{A} \in \mathbb{Z}_q^{n \times m}$, which is $\mathbf{R} \in \mathbb{Z}_q^{m \times m}$ such that $\mathbf{AR} = \mathbf{G}$ and $\|\mathbf{R}\|_\infty = \max_{i,j} r_{ij}$ is small. Having such an $\mathbf{R}$ allows us to solve ISIS with respect to $\mathbf{A}$ efficiently, since we can simply output $\mathbf{x} = \mathbf{R} \cdot G^{-1}(\mathbf{y})$, which will make $\mathbf{Ax} = \mathbf{y}$ as desired. Let $\|\mathbf{R}\|_\infty = \beta$, and note that $\left\|G^{-1}(\mathbf{y})\right\|_\infty = 1$. This implies $\left\|\mathbf{R} \cdot G^{-1}(\mathbf{y})\right\|_\infty \leq \beta m$.

We can now define $\mathsf{TrapGen}(n, m, q, \beta)$. First, sample $\overline{\mathbf{A}} \xleftarrow{R} \mathbb{Z}_q^{n \times m}$, $\overline{\mathbf{R}} \xleftarrow{R} \{0,1\}^{m \times m}$. Then, set $\mathbf{A} = \left[\overline{\mathbf{A}} \;\middle|\; \overline{\mathbf{A}}\overline{\mathbf{R}} + \mathbf{G}\right]$ and $\mathbf{R} = \left[-\overline{\mathbf{R}} \;\middle|\; \mathbf{I}_m\right]^T$. Note that $\mathbf{G}$ is a constant and $(\mathbf{A}, \mathbf{AR}) \approx (\mathbf{A}, \mathbf{T})$ by the Leftover Hash Lemma, so $\mathbf{A}$ looks uniform. We can see by inspection that $\mathbf{AR} = \mathbf{G}$ and $\|\mathbf{R}\|_\infty = 1$, as desired.

4.3 Digital Signatures from Lattice Trapdoors in the Random Oracle Model

Construction 4.4. The following is a candidate signature scheme from lattice trapdoors:
$\mathsf{KeyGen}(1^\lambda)$: $(\mathbf{A}, \mathsf{td}_\mathbf{A} = \mathbf{R}) \leftarrow \mathsf{TrapGen}(n, m, q, \beta)$; $\mathsf{vk} = \mathbf{A}, \mathsf{sk} = \mathbf{R}$ (where $\mathbf{AR} = \mathbf{G}$)
$\mathsf{Sign}(\mathsf{sk}, m)$: $\sigma = \mathbf{R} \cdot G^{-1}(H(m))$
$\mathsf{Verify}(\mathsf{vk}, m, \sigma)$: check $\|\sigma\| < \beta$ and $\mathbf{A}\sigma = H(m)$

Where we assume $H : \{0,1\}^* \to \mathbb{Z}_q^n$ is a random oracle. However, it turns out that doing this is not enough, because this scheme is totally broken. In particular, an adversary can make a bunch of sign queries and compute $G^{-1}(H(m))$ for each to construct a system of equations that will allow it to solve for $\mathbf{R}$. In order to properly hide $\mathbf{R}$, randomness is needed when signing. This can be accomplished by constructing a preimage sampleable trapdoor function.

Definition 4.5. We say $f : \mathcal{X} \to \mathcal{Y}$ is a preimage sampleable trapdoor function if there exists efficiently sampleable $\mathcal{D}$ over $\mathcal{X}$ and a trapdoor inversion algorithm $\mathsf{SamplePre}$ where:

$$\{x \leftarrow \mathcal{D}, y \leftarrow f(x) \ : \ (x, y)\} \approx \{y \xleftarrow{R} \mathcal{Y}, x \leftarrow \mathsf{SamplePre}(\mathsf{td}, y) \ : \ (x, y)\}$$

If our signing algorithm uses $\mathsf{SamplePre}$ in place of G^{-1}, Construction 3.1 is secure by an analogous argument to RSA-FDH.

CS 395T: Topics in Cryptography Febuary 2, 2022

Lecture 5: Preimage-Sampleable Trapdoor Functions

Lecturer: David Wu *Scribe: Yeonsoo Jeon*

5.1 Constructing Preimage Sampleable Trapdoor functions from SIS

recall the SIS hash function

$$f_A(x) := Ax \ (mod\ q)$$

where $A \in \mathbb{Z}_q^{n \times m}$ and $x \in \mathbb{Z}_q^m$. Then our goal is given target $y \in \mathbb{Z}_q^n$ sample some $x \in \mathbb{Z}^m$ such that $Ax = y$.

We want to decide some distribution over the preimages that we want to sample and it is independent to the trapdoor information

recall that the SIS lattice is defined as

$$\mathcal{L}^\perp(A) = \{x \in \mathbb{Z}^m : Ax = 0 \ (mod\ q)\}.$$

Its coset is also defined as

$$\mathcal{L}_u^\perp(A) = c + \mathcal{L}^\perp(A) = \{x \in \mathbb{Z}^m : Ax = u\}$$

where $c \in \mathbb{Z}_q^m$ and $Ac = u$.

Our challenge is to define a suitable distribution over $\mathcal{L}_u^\perp(A)$ which is conducive for pre-image sampling. That is, given a trapdoor, sampling preimage must be efficient and samples must not leak trapdoor. This distribution is typically a discrete Gaussian distribution. First we will define discrete Gaussian distribution, then use the distribution to construct such sampling processes.

5.2 Discrete Gaussian Distribution

Definition 5.1 (Discrete Gaussian Distribution)**.** For a parameter $s > 0$, we define the Gaussian function over $\mathbb{R}^n$ with width s as follows:

$$\rho_s(x) := \exp(-\pi \|x\|^2 / s^2)$$

$$\rho_{s,c}(x) := \exp(-\pi \|x - c\|^2 / s^2)$$

Discrete Gaussian over $\mathcal{L}$ centered at c:

$$D_{\mathcal{L},s,c}(x) \propto \begin{cases} \rho_{s,c}(x) & \text{if} \quad x \in \mathcal{L} \\ 0 & \text{otherwise} \end{cases}$$

$$Pr[x] = \frac{\rho_{s,c}(x)}{\sum_{x \in \mathcal{L}} \rho_{s,c}(x)}$$

Definition 5.2 (Truncated Discrete Gaussian Distribution)**.**

$$Pr[x] \propto \begin{cases} \rho_s(x) & \text{if} \quad x \in \mathcal{L} \quad \text{and} \quad \|x\| < \beta \\ 0 & \text{otherwise} \end{cases}$$

If $\beta > \omega(\sqrt{\log \lambda})$, truncated Gaussian distribution is statistically indistinguishable to discrete Gaussian distribution.

Figure 5.1 shows an example of discrete Gaussian distribution over $\mathbb{Z}^2$. Discrete Gaussian has good properties that are useful.

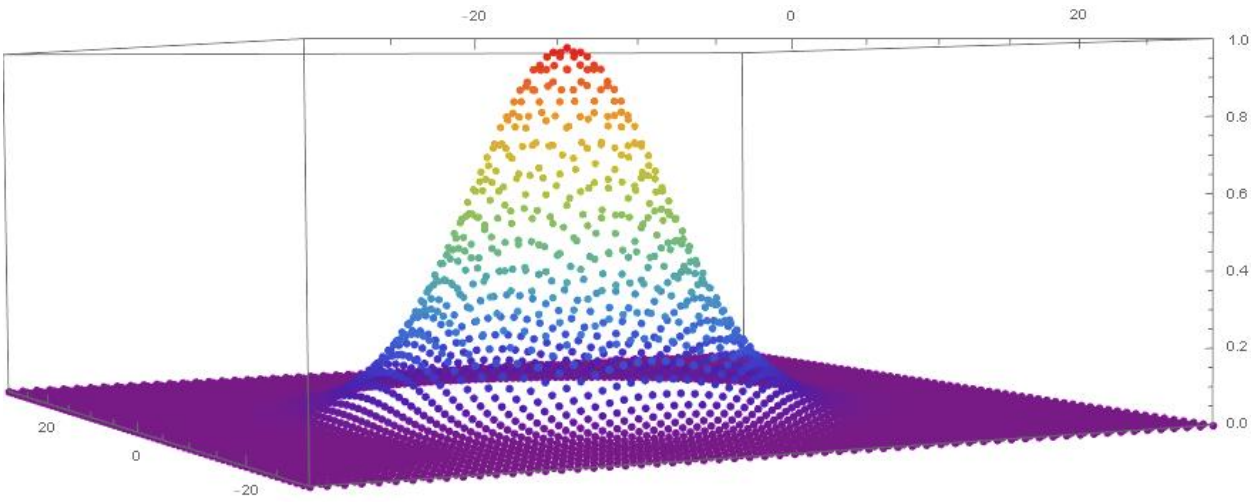


Figure 5.1: Discrete Gaussian distribution over $\mathbb{Z}^2$

- Rotational invariance over $\mathbb{R}^n$
  $\rho_s(\vec{\mathbf{x}}) = \prod_{i \in [n]} \rho_s(x_i)$
- (Gaussian Convolution Lemma) Sum of discrete Gaussian is another discrete Gaussian

Gaussian Convolution Lemma is the property that we will use to implement our Gaussian sampling procedure.

5.3 Gram-Schmidt Orthogonalization

Recall Gram-Schmidt Orthogonalization from linear algebra. Let $b_1, \ldots, b_n \in \mathbb{R}^d$ be a collection of vectors with $\text{span}(b_1, \ldots, b_n) = V$. Then the Gram-Schmidt orthogonalization process outputs $\tilde{b_1}, \ldots, \tilde{b_n} \in \mathbb{R}^n$ where $\text{span}(\tilde{b_1}, \ldots, \tilde{b_n}) = V$ and $\tilde{b_i}^\intercal \tilde{b_j} = 0$ for all $i \neq j$.

Algorithm 1 Gram-Schmidt Orthogonalization

Input $b_1, \ldots, b_n \in \mathbb{R}^d$ with $\text{span}(b_1, \ldots, b_n) = V$
Output $\tilde{b_1}, \ldots, \tilde{b_n} \in \mathbb{R}^n$ where $\text{span}(\tilde{b_1}, \ldots, \tilde{b_n}) = V$ and $\tilde{b_i}^\intercal \tilde{b_j} = 0$ for all $i \neq j$

1: $\tilde{b_1} \leftarrow b_1$
2: **for each** $i = 2, \ldots, n$ **do**
3: $\quad \tilde{b_i} \leftarrow b_i - \sum_{j<i} \frac{b_i^\intercal \tilde{b_j}}{\tilde{b_j}^\intercal \tilde{b_j}} \cdot \tilde{b_j}$
4: **end for**

Note that $B = [b_1 | \cdots | b_n]$ and $\tilde{B} = [\tilde{b_1} | \cdots | \tilde{b_n}]$ span the same vector space over $\mathbb{R}^n$, but do not necessarily generate the same lattice because $\tilde{b_i}$ is not necessarily an integer linear combination of $b_1, \ldots, b_n$.

The norm of the Gram-Schmidt vectors provides a bound on the minimum distance of a lattice. Let $\tilde{B} = [\tilde{b_1} | \cdots | \tilde{b_n}]$ be the Gram-Schmidt basis. Then the following holds:

$$\lambda_1(\mathcal{L}(B)) \geq \min_{i \in [n]} \left\| \tilde{b_i} \right\|.$$

To prove the statement, take any lattice point $Bx \neq 0$ where $x \in \mathbb{Z}^n$. Let k be the largest index where $x_k \neq 0$.

Consider the product

$$(Bx)^\intercal \tilde{b_k} = \sum_{i \le k} x_i b_i^\intercal \tilde{b_k} = x_k b_k^\intercal \tilde{b_k} = x_k \left\| \tilde{b_k} \right\|^2$$

since $\tilde{b_k}$ is orthogonal to $b_1, \ldots, b_{k-1}$ and $b_k^\intercal \tilde{b_k} = \left\| \tilde{b_k} \right\|^2$. By Cauchy-Schwarz inequality ($|u^\intercal v| \le \|u\| \cdot \|v\|$), we have

$$\|Bx\| \left\| \tilde{b_k} \right\| \ge |(Bx)^\intercal \tilde{b_k}| = |x_k| \cdot \left\| \tilde{b_k} \right\|^2$$

since $x_k \in \mathbb{Z}$ and $x_k \neq 0$.
Therefore,

$$\|Bx\| \ge \min_{i \in [n]} \left\| \tilde{b_i} \right\|.$$

5.4 Approach for Preimage Sampling

Theorem 5.3 (Gentry-Peikert-Vaikuntanathan). *There is an efficient algorithm that takes a basis B of a lattice $\mathcal{L} = \mathcal{L}(B)$, a coset $c + \mathcal{L}$ and a Gaussian width parameter $s \ge \left\| \tilde{B} \right\| \cdot \omega(\sqrt{\log n})$ where $\left\| \tilde{B} \right\| = \max_i \left\| \tilde{b_i} \right\|_2$ and outputs a sample whose distribution is statistically close to $D_{c+\mathcal{L},s}$.*

With the help of GPV theorem (we will cover the algorithm in the future lecture) we can construct the sampling processes.

Forward Sampling	:	Sample $x \leftarrow D_{\mathbb{Z}^m,s}$ and output (x, Ax).
Reverse Sampling	:	$y \xleftarrow{R} \mathbb{Z}_q^n$, compute any solution $z \in \mathbb{Z}_q^m$ where $Az = y$, Sample $v \leftarrow D_{\mathcal{L}^\perp(A),s,-z}$ and output $(z + v, y)$.

We know that if $x \xleftrightarrow{R} \{0,1\}^m$ then $(x, Ax) \overset{s}{\approx} (x, u)$ where $u \xleftarrow{R} \mathbb{Z}_q^n$. However, now x is sampled from discrete Gaussian distribution. Therefore, we need to show that for $x \leftarrow D_{\mathbb{Z}^m,s}$, $(x, Ax) \overset{s}{\approx} (x, u)$ where $u \xleftarrow{R} \mathbb{Z}_q^n$. In general, this statement does not hold. However, when s is sufficiently large, $Ax \overset{s}{\approx} Unif$. How large s should be in order to meet the condition is called "smoothing parameter".

CS 395T: Topics in Cryptography February 7, 2022

Lecture 6: Discrete Gaussian Sampling

Lecturer: David Wu *Scribe: Nitesh Kartha*

6.1 Preimage Sampleable Trapdoor Functions

Recall, that our goal is to construct preimage-sampleable trapdoor functions:

- **Forward sampling:** $x \leftarrow D_{\mathbb{Z}^m,s}$
  output: (x, Ax) [SIS Function]
- **Backward sampling:** $y \xleftarrow{R} \mathbb{Z}_q^n$ sample $v \leftarrow D_{\mathcal{L}^\perp(A),s,-z}$ where $z \in \mathbb{Z}_q^m$ such that $Az = y$;
  output: (v + z, y) [note that $A(v+z) = Av + Az = y$ since $Av = 0$]

We first need to show that the distribution of forward samples and backward samples are statistically indistinguishable for a significantly large s. To help us, we will first introduce the concept of the *smoothing parameter of a lattice.*

Informally, the **smoothing parameter of a lattice** is the minimum amount of Gaussian noise that needs to be added to "smooth out" the discrete structure of a lattice (i.e. the minimum width $s > 0$ such that every coset $c + \mathcal{L}$ has the same Gaussian mass). The more formal definition we will use in class is defined below:

Definition 6.1 (Smoothing parameter of a lattice η)**.** The smoothing parameter is the minimum width parameter $s > 0$ such that: For all $c \in \mathbb{R}^2$: $\rho_{s,c}(\mathcal{L}) \in [1 - \mathsf{negl}(n), 1] \cdot \rho_s(\mathcal{L})$.
It is denoted as $\eta(\mathcal{L})$ and $\eta(\mathcal{L}) \leq \lambda_n(\mathcal{L}) \cdot \omega(\sqrt{\log n})$

Note that you can simply consider the Gaussian mass of the fundamental parallel-piped created by the basis vectors of the lattice since everything in the lattice is an invariant of this region.Using this definition, we will prove the following claim:

Claim 6.2. *Given* $A \leftarrow \mathbb{Z}_q^{n\times m}$ *and* $x \leftarrow D_{\mathbb{Z}^m,s}$ $\quad Ax \overset{s}{\approx} \mathsf{Uniform}(\mathbb{Z}_q^n)$ *when* $s > \eta(\mathcal{L}^\perp(A))$ *and* $m \geq 3n \log q$

In order to help us in this proof, we will first prove this claim:

Claim 6.3. $x \bmod \mathcal{L}^\perp(A) \overset{s}{\approx} \mathsf{Uniform}(\mathbb{Z}^m/\mathcal{L}^\perp(A))$

Proof. Take any coset $c + \mathcal{L}^\perp(A)$ Since $x \leftarrow D_{\mathbb{Z}^m,s}$, $\mathsf{Pr}[x \in c + \mathcal{L}^\perp(A)] \propto \rho_s(c + \mathcal{L}^\perp(A))$ by the definition of a Discrete Gaussian. When $s > \eta(\mathcal{L}^\perp(A))$, it doesn't matter whether you center the distribution at the origin or an arbitrary point which means that $\rho_s(c + \mathcal{L}^\perp(A)) \overset{s}{\approx} \rho_s(\mathcal{L}^\perp(A))$. Importantly, this does not depend on what c is; it holds for all $c \in \mathbb{R}_n$. Thus, $x \bmod \mathcal{L}^\perp(A)$ is uniform over $\mathbb{Z}^m/\mathcal{L}^\perp(A)$. □

We can also note that $\mathbb{Z}^m/\mathcal{L}^\perp(A) \cong \mathbb{Z}_q^n$ using the multiplication with A as the homomorphism $(x+\mathcal{L}^\perp(A) \mapsto Ax)$. Alternatively, note that $x \in \mathbb{Z}_q^m \mapsto Ax \in \mathbb{Z}_q^n$ and the kernel of that group homomorphism is $\mathcal{L}^\perp(A)$ and the relationship follows from group theory.

Also note that when $m > 3n\log(q)$, the leftover hash lemma states that $\{(A, Ax) : A \xleftarrow{R} \mathbb{Z}_q^{n\times m}, x \xleftarrow{R} \{0,1\}^m\} \overset{s}{\approx} \{(A, u) : A \xleftarrow{R} \mathbb{Z}_q^{n\times m}, u \xleftarrow{R} \mathbb{Z}_q^n\}$. More concretely, the statistical distance is $q^- n$ which means that Ax is likely to map to the entirety of $\mathbb{Z}_q^n$.

Thus we have shown that if we forward sample (x, Ax), Ax is indistinguishable from $\mathsf{Uniform}(\mathbb{Z}_q^n)$. We now need to show that backward sampling is a Discrete Gaussian conditioned on Ax being a particular value.

Proof. Consider the distribution of x conditioned on a value y, $D_x(\hat{x}) : \frac{\rho_s(\hat{x})}{\rho_s(z+\mathcal{L}^\perp(A))}$ where $z : Az = y$. This is equivalent to: $\frac{\rho_{s,-z}(\hat{x}-z)}{\rho_{s,-z}(\mathcal{L}^\perp(A))}$ based on the definitions of the Gaussian mass function. This is equivalent to the Discrete Gaussian $D_{\mathcal{L}^\perp(A),s,-z}(\hat{x}-z)$ which is the distribution of v for the backward sampling technique.

If we write $x = z + v$, then $v = x - z$ and note that the distribution of v $D_v(\hat{v}) = D_x(\hat{v}-z) = D_{\mathcal{L}^\perp(A),s,-z}(\hat{v}+z-z) = D_{\mathcal{L}^\perp(A),s,-z}(\hat{v})$. ■

Thus we have proven that the backward and forward sampling distributions are statistically indistinguishable. Since this is sufficient to define a preimage samplable trapdoor function, all that remains is actually sampling from a Discrete Gaussian distribution.

6.2 Discrete Gaussian Sampling

A naive approach is to sample from a continuous Gaussian distribution over $\mathbb{R}^n$ and round to the nearest lattice point. However, this is **incorrect** and produces a distribution that is statistically far from the Discrete Gaussian. To demonstrate this, we will consider the one-dimensional case but this problem amplifies as the number of dimensions and complexity increases.

Considering the one-dimensional case, $\mathcal{L} = \mathbb{Z}$, we simply have to look at the probability mass assigned to 0 in the "rounded" Gaussian distribution to see how it differs from the Discrete Gaussian distribution:

In the "rounded Gaussian" technique, consider $y \leftarrow \mathsf{Gaussian}(s)$, y will only round to 0 when it is in the range $[-1/2, 1/2)$.

$$\mathsf{Pr}[y \in [-1/2, 1/2)] = \frac{1}{s}\int_{-1/2}^{1/2} \rho_s(y)\, dy$$

where s is the normalization form and is equal to $\int_{-\infty}^{\infty} \rho_s(y)\, dy$.

Note that this integral can be simplified to:

$$\frac{2}{s}\int_{0}^{1/2} \rho_s(y)\, dy$$

And using the change of variable with $t = \frac{\sqrt{\pi}y}{s}$, we get:

$$\frac{2}{\sqrt{\pi}}\int_{0}^{\sqrt{\pi}/2s} e^{-t^2}\, dt$$

since $\rho_s(y) = e^{\frac{-\pi y^2}{s}}$. This integral is equivalent to $\mathsf{erf}(\frac{\sqrt{\pi}}{2s})$. Note that $\mathsf{erf}(x)$ is equivalent to $\frac{2}{\sqrt{\pi}}(x - \Omega(x^3))$ using its Taylor expansion. Plugging in our result then yields $\frac{1}{s} - \Omega(\frac{1}{s^3})$.

For the discrete Gaussian, it is a bit more complicated to get a simple answer as it requires the use of Fourier transforms. For a more detailed explanation, refer to the lecture notes. Below is a simplified discussion.

We can note that:

$$D(0) = \frac{\rho_s(0)}{\sum_{x\in\mathbb{Z}} \rho_s(x)} = \frac{1}{\sum_{x\in\mathbb{Z}} \rho_s(x)}$$

The denominator $\sum_{x\in\mathbb{Z}} \rho_s(x)$ is equivalent to $\sum_{y\in\mathbb{Z}} \rho_{1/s}(y)$ from the Fourier transform. This summation is equivalent to $\sum_{y\in\mathbb{Z}} s \cdot e^{-\pi s^2 y^2}$ which is simplified to $s + s\sum_{y\neq 0} e^{-\pi s^2 y^2} = s \cdot (1 + \mathsf{negl}(\lambda)$. Thus, the overall result is negligible.

However, the "rounded Gaussian" distribution had probability mass of 0 to be non-negligible, thus showing that this naive approach is incorrect.

Finally, we will briefly go over how to sample from discrete Gaussians, with a more in-depth explanation in the next lecture. Recall the gadget trapdoor R is a "short" matrix (i.e. the elements of R are small) where $AR = G$ and we will use R to sample from the discrete Gaussian $D_{\mathcal{L}^{\perp}(A),s}$ where $s > s_1(R) \cdot \omega(\sqrt{\log n})$ and s_1 is the largest single value of R and $s_1 \leq m$:

$$[A|AR + G] \begin{bmatrix} -R \\ I_n \end{bmatrix} \tag{6.1}$$

CS 395T: Topics in Cryptography February 9, 2022

Lecture 7: Discrete Gaussian Sampling

Lecturer: David Wu *Scribe: Kristin Sheridan*

7.1 Review of Gaussian sampling

Goal: preimage sampling

The following should be statistically indistinguishable when s is sufficiently large compared to the smoothing parameter

- forward sampling: $x \leftarrow D_{\mathbb{Z}^n,s}, y \leftarrow Ax$: output (x, y)
- reverse sampling: $y \leftarrow \mathbb{Z}_q^n, v \leftarrow D_{\mathcal{L}^\perp(A),s,-z}$, where $Az = y$ for arbitrary x: output $(v + z, y)$

Today's focus: how to actually sample from a discrete Gaussian to achieve this

7.2 Rejection sampling

Goal: Sample from $D_{\mathbb{Z},s,c}$

- Recall from last time that rounding a sample from the continuous distribution can get us something far from the correct distribution even in one dimension
- Instead, use rejection sampling (better for lower dimensions)
  1. Sample $x \leftarrow \mathbb{Z} \cap [-t \cdot s + c, t \cdot s + c]$
     - idea behind this: "truncate the tails" of the Gaussian distribution, after t standard deviations from the center
     - When we do this, we will set $t(n) := \omega(\sqrt{\log n})$
     - By Gaussian tail bounds, $Pr_{x \leftarrow D_{\mathcal{Z},x,c}}[|x - c| > t \cdot s] \leq 2e^{-\pi t^2}$, which is negligible in n when $t = \omega(\sqrt{\log n})$ and thus "cutting the tail off" only marginally changes the distribution
     - Now we have a random integer from this range, but we need to convert to match the Gaussian distribution
  2. Output x with probability $\rho_{s,c}(x)$, when normalized (using the regular Gaussian density function); otherwise reject and restart
- Now we want to check that this actually works
  1. Does this terminate?
     - Modify the algorithm to try at most $t(n) \cdot \omega(\sqrt{\log n})$ times and output 0 if you have not succeeded at that point
     - How do we show we're unlikely to hit the "just return 0" point?
     - If on a given round we have $x \in [c - s, c + s]$, the probability that we output at this round is at least $e^{-\pi s^2/s^2} = e^{-\pi} = O(1)$, as it is at least the probability that we accept at $c - s$ or at $c + s$, the least likely acceptance points in this range
     - We also have $Pr_{x \leftarrow [-ts+c,ts+c]}[x \in [c-s, c+s]] = \frac{2s}{2ts} = \frac{1}{t}$, so we are fairly likely to pick something in this small range
     - Thus, we terminate on a given round with probability at least $O(1/t)$, and Chernoff bounds give us that after $t(n) \cdot O(\log \lambda)$ iterations, we will output something with probability at least $1 - negl(\lambda)$

7.3 GPV algorithm

Theorem 7.1. *Given a basis B for a $\mathcal{L} = \mathcal{L}(B)$, there exists an efficient algorithm that samples from a distribution statistically close to $D_{\mathcal{L},s,c}$ for any $s \geq |\tilde{B}|| \cdot \omega(\sqrt{\log n})$ (namely the GPV algorithm given below)*

Below we discuss the GPV algorithm, which samples from $D_{\mathcal{L},s,c}$ for an arbitrary $\mathcal{L} = \mathcal{L}(B)$ (as long as $s \geq ||\tilde{B}||\omega(\sqrt{\log n})$, where $\tilde{B}$ is the Graham-Schmidt normalization of B. The proof of this algorithm is given in the extra lecture notes but wasn't discussed in lecture.

1. Let $\mathbf{v}_n \leftarrow 0^n, \mathbf{c}_n \leftarrow \mathbf{c}$; for $i = n, n-1, \ldots, 1$
   (a) Compute $c_i' \leftarrow \frac{\mathbf{c}_i^T \tilde{\mathbf{b}}_i}{\tilde{\mathbf{b}}_i^T \tilde{\mathbf{b}}_i} \in \mathbb{R}$ ($\mathbf{c}_i$ projected onto i^{th} basis vector)
   Also compute $s_i' \leftarrow \frac{s}{||\tilde{\mathbf{b}}_i||}$
   (b) Sample $z_i \leftarrow D_{\mathbb{Z},s_i',c_i'}$
   (c) Update $\mathbf{c}_{i-1} \leftarrow \mathbf{c}_i - z_i \mathbf{b}_i$ and $\mathbf{v}_{i-1} \leftarrow \mathbf{v}_i + z_i \mathbf{b}_i$
2. Output $\mathbf{v}_0$

Intuitive sense of what this algorithm is doing:

- We have the Graham Schmidt orthogonal basis of the lattice; we start off with our estimate at 0^n, then we will repeatedly adjust it based on the outcome of repeated Gaussian sampling
- To do this, we are essentially picking coefficients z_i to place on the original basis vectors so that $\mathbf{v}_i = \sum_i z_i \mathbf{b}_i$
- To select the coefficient z_i, we first project the "current center" onto the i^{th} *normalized* basis vector and essentially normalize s to the length of the *original* i^{th} basis vector
- Then sample the coefficient from the discrete Gaussian on $\mathbb{Z}$, with the projected center and normalized variance
- Finally, we shift the center over by the negative of the amount we are adjusting our output value by

7.4 Using the GPV algorithm for signatures

Now we review the signature scheme we discussed before in the context of our new algorithms. Recall

$$G = \begin{bmatrix} 1 & 2 & \cdots & \log q & 0 & 0 & \cdots & 0 & 0 & \cdots & 0 & 0 & 0 & \cdots & 0 \\ 0 & 0 & \cdots & 0 & 1 & 2 & \cdots & \log q & 0 & \cdots & 0 & 0 & 0 & \cdots & 0 \\ \cdot & \cdot & \cdots & \cdot & \cdot & \cdot & \cdots & \cdot & \cdot & \cdots & \cdot & \cdot & \cdot & \cdots & \cdot \\ 0 & 0 & \cdots & 0 & 0 & \cdots & 0 & 0 & 0 & \cdots & 0 & 1 & 2 & \cdots & \log q \end{bmatrix}$$

and $g^T = \begin{bmatrix} 1 & 2 & \cdots & \log q \end{bmatrix}$ (ie $G = g^T \otimes I_n$). We also have $AR = G$ where $R \in \{0,1\}^{m \times m}$.

- First, we will sample $z \leftarrow D_{\mathcal{L}_y^\perp(G),s}$
  - To do this, notice that if $Gx = y$, then we can divide x and y into several vectors of length $\log q$ that are all concatenated together to form x and y; denote the i^{th} of these vectors by x^i and y^i
  - This means $g^T x^i = y^i$
  - Then we can use GPV to sample from $D_{\mathcal{L}_y^\perp(g^T),s}$ (ie sample some x^i such that $g^T x^i = y^i$) and concatenate the results together
- To use GPV this we need to find a basis for $\mathcal{L}_y^\perp(g^T)$

- We have that $D_{\mathcal{L}^{\perp}_{y}(g^T),s}$ is a re-centered version of $D_{\mathcal{L}^{\perp}(g^T),s}$, so we can use

$$B = \begin{bmatrix} 2 & 0 & 0 & \cdots & 0 & 0 \\ -1 & 2 & 0 & \cdots & 0 & 0 \\ 0 & -1 & 2 & \cdots & 0 & 0 \\ \cdot & \cdot & \cdot & \cdots & \cdot & \cdot \\ 0 & 0 & 0 & \cdots & -1 & 2 \end{bmatrix}$$

  as a short basis for $\mathcal{L}^{\perp}(g^T)$ which is sufficient (recalling that this lattice is the set of x such that $g^T x = 0$)
- The above basis has as its Graham-Schmidt norm $\tilde{B} = 2I_n$, which means $||\tilde{B}|| = 2$
- Thus, we can use GPV if $s > \omega(\sqrt{\log n})$

- Now we have that $ARx = Gx = y$, so we could try using Rx as our output value
  - However, while the distribution of Rx is a Gaussian, its covariance is not the same in every direction (ie it is not spherical)
  - In fact, it has covariance matrix $s^2 RR^T$, so it doesn't look like the forward sampling
  - One idea to make this look more like forward sampling would be to change the forward sampling method to have this covariance, however this is still an issue as we can recover RR^T if we see enough signatures and this has the same problem as our original scheme proposal

To fix this, we'll use the following Gaussian convolution lemma [Pei 11]

Lemma 7.2. *(Gaussian convolution lemma - informal) Under mild conditions, sum of two discrete Gaussians is a discrete Gaussian, and moreover the covariances add.*

- So far, we have sampled x from the appropriate discrete Gaussian with covariance $s^2 RR^T$ such that $Ax = y$, and we are looking to just correct the covariance of the random variable
- We will sample a perturbation $p \in \mathbb{Z}^m$ with covariance $\hat{s}^2 I - s^2 RR^T$, where $Ap = 0$
  - Here $\hat{s} \geq s \cdot s_1(R)$, where $s_1(R)$ is the largest singular value of R (this ensures we get that $\hat{s}^2 I - s^2 RR^T$ to be positive definite, a necessary condition of a covariance matrix)
- Then we can output $x + p$, and the overall covariance is $s^2 RR^T + \hat{s}^2 I - s^2 RR^T = \hat{s}^2 I$, using the Gaussian convolution lemma
- Notably, this does change the output value of $A(p+x)$ - we can fix this by picking p first and then sampling x such that $Ax = y - Ap$ instead of $Ax = y$, leaving us with $A(p+Rx) = Ap + ARx = Ap + Gx = Ap + (y - Ap) = y$

Summary of the overall algorithm we now have:

1. Sample $p \leftarrow D_{\mathbb{Z}^m, \hat{s}^2 I - s^2 RR^T}$
2. Sample $x \leftarrow D_{\mathcal{L}^{\perp}(G), s, y - Ap}$
3. Output $p + Rx$

The second homework problem will look at the GPV scheme using the sampling method and find an error with the version we've talked about so far. As a reminder, the scheme is as follows:
Setup: $vk = A$, $sk = B$ (where B is a short basis for A)
Sign: Find $H(m)$ and sample $x \leftarrow D_{\mathcal{L}^{\perp}_{H(m)}(A), s}$, output x
Verify: Check that $Ax = H(m)$ and $||x|| \leq \beta$

CS 395T: Topics in Cryptography February 14, 2022

Lecture 8: Learning With Errors

Lecturer: David Wu *Scribe: Steven Xu*

This lecture, we introduced the Learning With Errors (LWE) problem and used it to build public key encryption.

8.1 Basics of LWE

Definition 8.1 (Learning with Errors)**.** Let λ be a security parameter and $n = n(\lambda)$, $m = m(\lambda)$, and $q = q(\lambda)$ be lattice parameters. Also, let χ be an error distribution over $\mathbb{Z}_q$. χ is typically a discrete Gaussian so that samples are generally short.

The $\mathsf{LWE}_{n,m,q,\chi}$ assumption states that for $\mathbf{A} \xleftarrow{R} \mathbb{Z}_q^{n \times m}$, $\mathbf{s} \xleftarrow{R} \mathbb{Z}_q^n$, and $\mathbf{e} \xleftarrow{R} \chi$, the distributions

$$(\mathbf{A}, \mathbf{s}^T\mathbf{A} + \mathbf{e}^T) \text{ and } (\mathbf{A}, \mathbf{u}^T) \text{ for } \mathbf{u} \xleftarrow{R} \mathbb{Z}_q^m$$

are computationally indistinguishable. The vector $\mathbf{s}$ is commonly referred to as the secret, and the vector $\mathbf{e}$ is known as the error.

Intuitively, the LWE assumption states that a noisy random linear combinations of n vectors in $\mathbb{Z}^m$ are indistinguishable from random. For $m < n$, LWE holds since the image of $\mathbf{A}$ (under left multiplication by row vectors) is almost always the entire space of $\mathbb{Z}^m$, so multiplying by a random vector yields a random vector.

But for $m > n$, the image of $\mathbf{A}$ is a strict subspace of $\mathbb{Z}^m$. If there were no error, we'd be able to use Gaussian elimination to check of $\mathbf{s}^T\mathbf{A}$ is in the image of $\mathbf{A}$, and if we were working in the reals, we'd be able to use linear regression to check the likelyhood that $\mathbf{s}^T\mathbf{A} + \mathbf{e}$ was sampled from random. Typically, we take $m \gg n$ to be sufficiently large so that the secret vector is uniquely determined.

8.2 Properties of LWE

Claim 8.2 (SIS from LWE)**.** $\mathsf{LWE}_{n,m,q,\chi}$ *implies* $\mathsf{SIS}_{n,m,q,\beta}$.

Proof. We will prove the contrapositive. Suppose SIS is easy for n, m, q, β. Then we can use our SIS solver to find a $x \in \mathbb{Z}_q^m$ such that $\mathbf{A}x = 0$ and $||x|| < \beta$. We can use the short solution x to "cancel" our secret term and compute $(\mathbf{s}^T\mathbf{A} + \mathbf{e}^T)x = \mathbf{s}^T(\mathbf{A}x) + \mathbf{e}^Tx = \mathbf{e}^Tx$. However, $||\mathbf{e}^Tx|| \leq \beta||\mathbf{e}|| \ll q$, but for a uniformly sampled $\mathbf{u} \xleftarrow{R} \mathbb{Z}_q^m$, $\mathbf{u}^Tx$ is unlikely to be short, allowing us to solve LWE. □

Another notable property is that LWE is hard even if the secret $\mathbf{s}$ is sampled from a discrete Gaussian distribution. More precisely, if LWE is hard for $\mathbf{s} \xleftarrow{R} \mathbb{Z}_q^n$, then LWE is also hard for $\mathbf{s} \xleftarrow{R} \chi'$, where χ' is some discrete Gaussian over $\mathbb{Z}_q^n$ of sufficiently large width. Importantly, this means that LWE is hard even when the secret $\mathbf{s}$ is short.

There are two problems associated with LWE—the decision problem, which the LWE assumption states directly is hard, and the search problem, where the goal is to recover $\mathbf{s}$ given $(\mathbf{A}, \mathbf{s}^T\mathbf{A} + \mathbf{e}^T)$. More precisely, the goal of the search problem is to find the vector

$$v \in \mathcal{L}(\mathbf{A}^T) = \left\{\mathbf{A}^T\mathbf{s} \;\middle|\; \mathbf{s} \in \mathbb{Z}_q^n\right\} + q\mathbb{Z}^m$$

which is closest to some given point $\mathbf{z}$, where $\mathbf{z}$ would be our $\mathbf{s}^T\mathbf{A} + \mathbf{e}^T$ transposed. Note that this is an instance of the bounded distance decoding problem over the lattice $\mathcal{L}(\mathbf{A}^T)$. There is a reduction from search LWE to decision LWE.

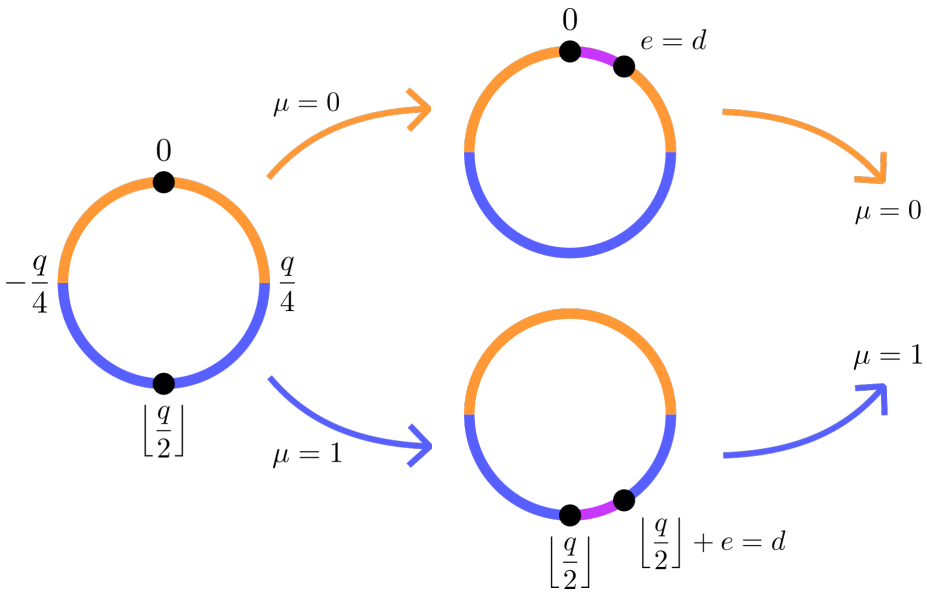
Finally, there exists a worst case *quantum* reduction from GapSVP to LWE. This means that given a quantum algorithm for solving average case LWE, there exists a quantum algorithm for solving worst case GapSVP. However, there are certain partial results giving classical worst case reductions for restricted parameters regimes.

Proposition 8.3. *For any $m = \mathsf{poly}(n)$, $q < 2^{\mathsf{poly}(n)}$ and some discrete gaussian χ with values bounded by β, solving $\mathsf{LWE}_{n,m,q,\chi}$ on a quantum computer is at least as hard as solving GapSVP_γ on a quantum computer for an arbitrary n dimensional lattice with approximation factor $\gamma = \tilde{O}(nq/\beta)$.*

8.3 Symmetric Encryption with LWE

For our symmetric encryption scheme, we will encrypt a single bit ($\mathcal{M} = \{0, 1\}$). The idea behind the symmetric encryption scheme is that we'll take $m = 1$, our LWE secret $\mathbf{s}$ to be our secret key, and we'll use $\mathbf{s}^T\mathbf{A} + \mathbf{e}$ to hide our single bit.

Let μ be our message. To encrypt, we'll add $\lfloor q/2 \rfloor$ to $\mathbf{s}^T\mathbf{A} + e$ if $\mu = 1$ and 0 if $\mu = 0$. Our encrypted message is then $(\mathbf{A}, \mathbf{s}^T\mathbf{A} + e + \mu \lfloor q/2 \rfloor)$. To decrypt, we'll subtract $\mathbf{s}^T\mathbf{A}$ from $\mathbf{s}^T\mathbf{A} + e + \mu \lfloor q/2 \rfloor$ to get $d = e + \mu \lfloor q/2 \rfloor$. Since e is small, d is approximately just $\mu \lfloor q/2 \rfloor$. Thus we'll decrypt to 0 if d is closer to 0 and 1 if d is closer to $\lfloor q/2 \rfloor$ in the ring $\mathbb{Z}_q$. The algorithm is demonstrated in the diagram below.



More precisely, the LWE symmetric encryption scheme works as follows.

Construction 8.4 (Symmetric encryption from LWE)**.** Let λ be a security parameter and $n = n(\lambda)$, $m = m(\lambda)$, and $q = q(\lambda)$ be lattice parameters. We construct our symmetric encryption scheme with message space $\mathcal{M} = \{0, 1\}$ as follows:

- $\mathsf{Setup}(1^n)$: Sample $\mathbf{s} \xleftarrow{R} \mathbb{Z}_q^n$. This is our secret key.
- $\mathsf{Encrypt}(\mathbf{s}, \mu)$: Sample $\mathbf{A} \xleftarrow{R} \mathbb{Z}_q^n$ and $e \xleftarrow{R} \chi$. Output $\mathbf{c} = (\mathbf{A}, \mathbf{s}^T\mathbf{A} + e + \mu \lfloor q/2 \rfloor)$ as our ciphertext.
- $\mathsf{Decrypt}(\mathbf{s}, (\mathbf{c}_1, c_2))$: Output $\mu = 0$ if $-q/4 \leq c_2 - \mathbf{s}^T\mathbf{c}_1 < q/4$ and 1 otherwise.

If $(\mathbf{c}_1, c_2) = (\mathbf{A}, \mathbf{s}^T\mathbf{A} + e + \mu \lfloor q/2 \rfloor)$, then $c_2 - \mathbf{s}^T\mathbf{c}_1 = e + \mu \lfloor q/2 \rfloor = d$, so our decryption is correct by the logic from earlier. For security, we have $(\mathbf{A}, \mathbf{s}^T\mathbf{A} + e) \approx (\mathbf{A}, r) \equiv (\mathbf{A}, r + \mu \lfloor q/2 \rfloor) \approx (\mathbf{A}, \mathbf{s}^T\mathbf{A} + e + \mu \lfloor q/2 \rfloor)$, where $r \xleftarrow{R} \mathbb{Z}_q$. But $(\mathbf{A}, \mathbf{s}^T\mathbf{A} + e)$ is the encryption of 0 and $(\mathbf{A}, \mathbf{s}^T\mathbf{A} + e + \mu \lfloor q/2 \rfloor)$ is the encryption of 1, demonstrating that they are indistinguishable. Note that $(\mathbf{A}, r) \equiv (\mathbf{A}, r + \mu \lfloor q/2 \rfloor)$ since r is uniformly random over $\mathbb{Z}_q$, and the shift of a uniformly random value is still uniformly random.

This encryption scheme is extremely inefficient. It takes n bits to encrypt a single bit, where n is the security parameter (the size of the key). One possible improvement is to split the ring of $\mathbb{Z}_q$ into p disjoint parts rather than

just 2 parts in order to encrypt p values. However, that improvement still isn't very efficient, and there exist other optimizations for this encryption scheme which are asymptotically optimal.

Note that this encryption scheme is additively homomorphic. That is, for two ciphertexts $(\mathbf{A}_1, \mathbf{s}^T\mathbf{A}_1 + e_1 + \mu_1 \lfloor q/2 \rfloor)$ and $(\mathbf{A}_2, \mathbf{s}^T\mathbf{A}_2 + e_2 + \mu_2 \lfloor q/2 \rfloor)$, their sum (mod q) is

$$(\mathbf{A}_1 + \mathbf{A}_2, \mathbf{s}^T(\mathbf{A}_1 + \mathbf{A}_2) + (e_1 + e_2) + (\mu_1 + \mu_2) \lfloor q/2 \rfloor)$$

which is a ciphertext of $\mu_1 + \mu_2$ (mod 2). However, care must be taken to make sure magnitude of the error term $e_1 + e_2$ does not get too big. More precisely, taking into account rounding error, we must have $|e_1 + e_2 + 1| < q/4$ for the addition to work correctly.

8.4 Public Key Encryption with LWE

In order to build a public key encryption scheme, we will take advantage of the fact that symmetric LWE encryption is additively homomorphic. The following scheme is called primal Regev encryption. The public key is a set m of encryptions of 0s in symmetric LWE, and the private key is just the secret key from symmetric LWE.

To encrypt, a user will choose a random subset of the encryptions of 0s and compute their sum, yielding another encryption of 0. Then if their message is 1, they will add in $\lfloor q/2 \rfloor$ to the second term of the sum, turning the encryption of 0 into an encryption of 1. The main idea is that the subset sum acts as a seemingly fresh encryption of 0 which then hides the message.

More precisely, the primal Regev encryption scheme works as follows.

Construction 8.5 (Public key encryption from LWE)**.** Let λ be a security parameter and $n = n(\lambda)$, $m = m(\lambda)$, and $q = q(\lambda)$ be lattice parameters. We construct our public key encryption scheme with message space $\mathcal{M} = \{0, 1\}$ as follows:

- $\mathsf{Setup}(1^n)$: Sample $\mathbf{s} \xleftarrow{R} \mathbb{Z}_q^n$, $\mathbf{A} \xleftarrow{R} \mathbb{Z}_q^{n \times m}$, and $\mathbf{e} \xleftarrow{R} \chi^m$. Our secret key is $\mathbf{s}$, and our public key is $(\mathbf{A}, \mathbf{s}^T\mathbf{A} + \mathbf{e}^T)$.
- $\mathsf{Encrypt}((\mathbf{A}, \mathbf{s}^T\mathbf{A} + \mathbf{e}^T), \mu)$: Sample $\mathbf{r} \xleftarrow{R} \{0, 1\}^m$, and output $(\mathbf{A}\mathbf{r}, (\mathbf{s}^T\mathbf{A} + \mathbf{e}^T)\mathbf{r} + \mu \lfloor q/2 \rfloor)$ for the ciphertext.
- $\mathsf{Decrypt}(\mathbf{s}, (\mathbf{c}_1, \mathbf{c}_2))$: Output $\mu = 0$ if $-q/4 \leq \mathbf{c}_2 - \mathbf{s}^T\mathbf{c}_1 < q/4$ and 1 otherwise.

For correctness, if we have $(\mathbf{c}_1, \mathbf{c}_2) = (\mathbf{A}\mathbf{r}, (\mathbf{s}^T\mathbf{A} + \mathbf{e}^T)\mathbf{r} + \mu \lfloor q/2 \rfloor)$, then

$$\mathbf{c}_2 - \mathbf{s}^T\mathbf{c}_1 = (\mathbf{s}^T\mathbf{A} + \mathbf{e}^T)\mathbf{r} + \mu \lfloor q/2 \rfloor - \mathbf{s}^T\mathbf{A}\mathbf{r} = \mathbf{e}^T\mathbf{r} + \mu \lfloor q/2 \rfloor$$

which from our analysis of symmetric LWE, we know will successfully decrypt if $|\mathbf{e}^T\mathbf{r}| < q/4$.

Claim 8.6. *The above construction is semantically secure.*

Proof. We begin by defining a sequence of hybrid arguments.

- In Hyb_0, the adversary sees $(\mathbf{A}, \mathbf{s}^T\mathbf{A} + \mathbf{e}^T, \mathbf{A}\mathbf{r}, (\mathbf{s}^T\mathbf{A} + \mathbf{e}^T)\mathbf{r} + \mu \lfloor q/2 \rfloor)$. The first two terms are the public key, and the last two terms are an encryption of μ generated using the public key.
- In Hyb_1, the adversary sees $(\mathbf{A}, \mathbf{u}^T, \mathbf{A}r, \mathbf{u}^T\mathbf{r} + \mu \lfloor q/2 \rfloor)$ for some $\mathbf{u} \xleftarrow{R} \mathbb{Z}_q^m$.
- In Hyb_2, the adversary sees $(\mathbf{A}, \mathbf{u}^T, \mathbf{z}_1, \mathbf{z}_2 + \mu \lfloor q/2 \rfloor)$, where $\mathbf{u}$ is the same as in Hyb_1, $\mathbf{z}_1 \xleftarrow{R} \mathbb{Z}_q^m$, and $\mathbf{z}_2 \xleftarrow{R} \mathbb{Z}_q$.

Hyb_0 and Hyb_1 are indistinguishable because by LWE, $\mathbf{s}^T\mathbf{A} + \mathbf{e}^T$ is indistinguishable from random. Hyb_1 and Hyb_2 are indistinguishable because by the leftover hash lemma, we have that $[\mathbf{A}^T \mid \mathbf{u}]^T\mathbf{r}$ is indistinguishable from random since $\mathbf{A}$, $\mathbf{u}^T$, and $\mathbf{r}$ are uniform random. Note that this requires m be $\Omega(n \log q)$. Then since $(\mathbf{A}, \mathbf{u}^T, \mathbf{z}_1, \mathbf{z}_2 + \mu \lfloor q/2 \rfloor)$ clearly hides μ, our public key encryption scheme is secure. □

CS 395T: Topics in Cryptography February 16, 2022

Lecture 9: Fully Homomorphic Encryption

Lecturer: David Wu *Scribe: Garrett Gu*

9.1 Fully Homomorphic Encryption

We begin by defining Fully Homomorphic Encryption (FHE). For over 30 years, FHE was considered the "holy grail of cryptography".

Definition 9.1. Suppose we have two participants, Alice and Bob. Alice has a message and calculates a ciphertext $\text{ct} \in \text{Enc}(\text{pk}, \mu)$, then sends this ciphertext to Bob. A **Fully Homomorphic Encryption Scheme** allows Bob to calculate $\text{Enc}(\text{pk}, f(\mu))$ for arbitrary function f.

As it turns out, we have already seen an additively homomorphic encryption scheme, ElGamal.

Recall that the public key in ElGamal is $g, h = g^\alpha$ where α is secret and g generates a cyclic group of order q. Then encryption of μ_1 is calculated by sampling random $r_1 \in \mathbb{Z}_q$ and calculating

$$g^{r_1}, h^{r_1} g^{\mu_1} \tag{9.1}$$

If we obtained another ciphertext $g^{r_2}, h^{r_2} g^{\mu_2}$, we can calculate a ciphertext for $\mu_1 + \mu_2$ by simply multiplying the corresponding elements to obtain

$$g^{r_1 + r_2}, h^{r_1 + r_2} g^{\mu_1 + \mu_2} \tag{9.2}$$

and we're done!

However, ElGamal is not multiplicatively homomorphic (as far as we know). We would like to get an encryption scheme that is both additively and multiplicatively homomorphic over $\mathbb{Z}_2$, since this suffices to evaluate **arbitrary Boolean gates**. Addition corresponds to an XOR gate in $\mathbb{Z}_2$ and multiplicatioon coresonds to an AND gate.

We can express any function over $\mathbb{Z}_2^n$ as a circuit consisting of XOR/AND gates, then if we have an encryption scheme that is both additively and multiplicatively homomorphic over $\mathbb{Z}_2$, we can evaluate this Boolean circuit to obtain the encryption of $f(\mu)$.

The first Fully Homomorphic Encryption Scheme was created by a PhD student named Craig Gentry. His scheme consisted of two steps:

1. Build a **somewhat homomorphic encryption scheme** (SWHE). This scheme supports a bounded number of homomorphic operations.
2. Bootstrap this SWHE scheme into an FHE scheme by "refreshing" the ciphertext. This can be done if the SWHE can perform enough operations to evaluate its own decryption circuit, with an additional circular security assumption.

9.2 Somewhere Homomorphic Encryption from LWE

Today we will focus on building a SWHE. We begin with Regev's encryption. Recall that Regev's encryption scheme has public key

$$A = \begin{bmatrix} \bar{A} \\ \bar{s}^T \bar{A} + e^T \end{bmatrix} \tag{9.3}$$

where

$$\bar{A} \xleftarrow{R} \mathbb{Z}_q^{n \times \bar{m}}$$
$$\bar{s} \xleftarrow{R} Z_q^n$$
$$e \leftarrow x^{\bar{m}}$$

and the secret key is

$$s = \left[\begin{array}{c} -\bar{s} \\ \hline 1 \end{array}\right] \tag{9.4}$$

Then

$$\begin{aligned} s^T A &= -\bar{s}^T \bar{A} + \bar{s}^T \bar{A} + e^T \\ &= e^t \\ &\approx 0 \end{aligned}$$

Now, to obtain a ciphertext, we first sample

$$r \xleftarrow{R} \{0,1\}^m \tag{9.5}$$

and calculate

$$c \leftarrow Ar + \left[\begin{array}{c} 0^{n-1} \\ q/2 \cdot \mu \end{array}\right] \tag{9.6}$$

Decryption is then performed by multiplying the secret key with the ciphertext:

$$\begin{aligned} s^T c &= s^T Ar + s^T \left[\begin{array}{c} 0^{n-1} \\ q/2 \cdot \mu \end{array}\right] \\ &= e^T r + \frac{q}{2} \cdot \mu \end{aligned}$$

We can then obtain the value of μ using rounding.

We will begin by extending the ciphertext into a matrix. First we describe a bogus encryption scheme that has a lot of nice intuitions if we ignore the error, and we will later discuss how to handle the error.

First we extend the matrix A into a square matrix.

$$\hat{A} = \left[\begin{array}{c} A \\ \hline 0^{(n-m)\times m} \end{array}\right] \in Z_q^{m \times m} \tag{9.7}$$

We will also pad the key accordingly.

$$\hat{s} = \left[\begin{array}{c} s \\ \hline 0^{m-n} \end{array}\right] \in \mathbb{Z}_q^m \tag{9.8}$$

Now note

$$\hat{s}^T \hat{A} = s^T A = e^T \tag{9.9}$$

To encrypt, first sample

$$R \xleftarrow{R} \{0,1\}^{m \times m} \tag{9.10}$$

then compute

$$C \leftarrow \hat{A} + \mu \cdot \left\lfloor \frac{q}{2} \right\rceil \cdot \left[\begin{array}{c|c} I_n & 0^{n\times(m-n)} \\ \hline 0^{(m-n)\times n} & 0^{(m-n)\times(m-n)} \end{array}\right] \tag{9.11}$$

Then decryption is still performed by multiplying the secret key with the ciphertext.

$$\begin{aligned} \hat{s}^T C &= \hat{s}^T \hat{A} R + \mu \cdot \left\lfloor \frac{q}{2} \right\rceil \cdot \hat{s} \left[\begin{array}{c|c} I_n & 0 \\ \hline 0 & 0 \end{array}\right] \\ &= e^T R + \mu \cdot \left\lfloor \frac{q}{2} \right\rceil \cdot \hat{s}^T \end{aligned}$$

Now, if we ignored the error term, we would end up with the equation

$$\hat{s}^T C = \mu \cdot \left\lfloor \frac{q}{2} \right\rceil \cdot \hat{s}^T \tag{9.12}$$

Now, if we temporarily ignore the $\left\lfloor \frac{q}{2} \right\rceil$ term, then $\hat{s}$ is a left eigenvector of C with associated message equal to the message!

Now suppose we have two ciphertexts,

$$\hat{s}^T C_1 = \mu_1 \cdot \hat{s}^T$$
$$\hat{s}^T C_2 = \mu_2 \cdot \hat{s}^T$$

Then we have additive homomorphism:

$$\begin{aligned} \hat{s}^T (C_1 + C_2) &= \hat{s}^T C_1 + \hat{s}^T C_2 \\ &= (\mu_1 + \mu_2) \hat{s}^T \end{aligned}$$

And multiplicative homomorphism:

$$\begin{aligned} \hat{s}^T C_1 C_2 &= \mu_1 \hat{s}^T C_2 \\ &= \mu_1 \mu_2 \hat{s}^T \end{aligned}$$

We get both additive and multiplicative homormorphism from the additive and multiplicative properties of an eigenvector/eigenvalue system.

But the catch is that we ignored the error and $\left\lfloor \frac{q}{2} \right\rceil$ to get these properties, so we really haven't done anything since the error is important for security in LWE.

If we did include the error, we would run into an issue.

Adding the two ciphertexts together, we get

$$\hat{s}^T (C_1 + C_2) = e^T (R_1 + R_2) + (\mu_1 + \mu_2) \cdot \left\lfloor \frac{q}{2} \right\rceil \cdot \hat{s}^T \tag{9.13}$$

This is fine since we are simply adding the error terms together, so our total error does not increase by much.

However, if we tried to multiply two ciphertexts together, we would get

$$\begin{aligned} \hat{s}^T C_1 C_2 &= (e^t R_1 + \mu_1 \cdot \left\lfloor \frac{q}{2} \right\rceil \cdot (\hat{s})^T) C_2 \\ &= e^T R_1 C_2 + \mu_1 \mu_2 \cdot \left\lfloor \frac{q}{2} \right\rceil \cdot \hat{s}^T + \mu_1 \cdot \left\lfloor \frac{q}{2} \right\rceil \cdot e^T R_2 \end{aligned}$$

This is bad because both $e^T R_1 C_2$ and $\mu_1 \cdot \left\lfloor \frac{q}{2} \right\rceil \cdot e^T R_2$ are both large, meaning our noise blows up.

But we have a mechanism of compressing a matrix while preserving matrix products, namely the G matrix.

Note that

$$\begin{aligned} &G^{-1} : \mathbb{Z}_q^n \to \mathbb{Z}_q^m \\ &G \cdot G^{-1}(v) = v \qquad \forall v \in \mathbb{Z}_q^n \\ &||G^{-1}(v)||_\infty = 1 \end{aligned}$$

9.3 Gentry-Sahai-Waters FHE

Now we have everything we need to construct Gentry-Sahai-Waters FHE (GSW). The setup is identical to Regev's Encryption. The encryption is performed as below:

$$\begin{aligned} &R \xleftarrow{\text{R}} \{0,1\}^{m \times m} \\ &C \leftarrow AR + \mu \cdot G \end{aligned}$$

And decryption is performed as below:

$$\begin{aligned} s^T C &= s^T AR + \mu \cdot s^T G \\ &= (e^T R + \mu \cdot s^T G) G^{-1}(\frac{q}{2} \cdot I_n) \\ &= e^T R G^{-1}(\frac{q}{2} \cdot I_n) + \mu \cdot s^T \cdot \frac{q}{2} I_n \end{aligned}$$

Now suppose e comes from a bounded Gaussian distribution with bound B, then the resulting decryption noise will be $B \cdot n^2$. Since we require the noise to be less than $q/4$, it suffices to use a value of $q > 4Bm^2$.

The security properties are identical to Regev's Encryption.

Now we will show that GSW is homomorphic. Suppose

$$\begin{aligned} C_1 &= AR_1 + \mu_1 G \\ C_2 &= AR_2 + \mu_2 G \end{aligned}$$

Then

$$C_1 + C_2 = A(R_1 + R_2) + (\mu_1 + \mu_2) \cdot G$$

So the scheme is additively homomorphic with negligible change in noise.

Additionally,

$$\begin{aligned} C_1 G^{-1}(C_2) &= (AR_1 + \mu_1 G) G^{-1}(C_2) \\ &= AR_1 G^{-1}(C_2) + \mu_1 \cdot C_2 \\ &= AR_1 G^{-1}(C_2) + \mu_1 AR_2 + \mu_1 \mu_2 G \\ &= A[R_1 G^{-1}(C_2) + \mu_1 R_2] + \mu_1 \mu_2 G \end{aligned}$$

Then the randomness accumulated through multiplication is

$$||R_x||_\infty \leq m \cdot ||R_1|| + ||R_2|| \tag{9.14}$$

which is small.

So after d multiplications, our noise increases by a factor of $m^{O(d)}$. We would then have to set $q \approx m^{O(d)}$ or

$$\log(q) \approx d \log(m) \tag{9.15}$$

Therefore the number of bits we need to represent the modulus is proportional to the multiplicative depth we would like to support.

CS 395T: Topics in Cryptography February 21, 2022

Lecture 10: FHE Bootstrapping

Lecturer: David Wu *Scribe: Rachit Garg*

In the last lecture, we saw how to construct a Somewhat Homomorphic Encryption Scheme (SWHE). This construction supported bounded depth computation where the LWE modulus $q > m^{O(d)}$, where d is the multiplicative depth of the computation. Additionally, if the depth of the circuit is $\omega(1)$, we would require a super-polynomial modulus and a stronger assumption. In this lecture, we will see how to obtain Fully Homomorphic Encryption (FHE). We will study about Gentry's idea on bootstrapping.

High Level Idea Gentry proposed the idea of refreshing ciphertexts (re-encryption) to reduce noise in the evaluated ciphertexts. Let's say we compute a function f homomorphically on a ciphertext ct (with noise B). By our homomorphic evaluation, we end up with $\mathsf{ct}_{f(x)}$ where the noise is $\mathsf{B} \cdot m^{O(d)}$. Our refresh procedure would allow us to perform a public operation on $\mathsf{ct}_{f(x)}$ and output $\mathsf{ct}'_{f(x)}$ with noise $\mathsf{B}' = B\mathsf{poly}(n)$. If we have such a refresh procedure, then we can refresh when our ciphertexts become to big and thus eventually perform unbounded depth computation.

To see how to achieve this, suppose we have a ciphertext ct that encrypts x, i.e. $\mathsf{Decrypt}(\mathsf{sk}, \mathsf{ct}) = x$. We define a boolean circuit $C_{\mathsf{ct}} : \{0,1\}^{n \log q} \to \{0,1\}$, where $C_{\mathsf{ct}}(\mathsf{sk}) = \mathsf{Decrypt}(\mathsf{sk}, \mathsf{ct})$. Suppose we publish the public parameters, $\mathsf{Encrypt}(\mathsf{pk}, \mathsf{sk})$ and homomorphically compute the boolean circuit C_{ct} on $\mathsf{Encrypt}(\mathsf{pk}, \mathsf{sk})$, i.e. we output the computation $\mathsf{Encrypt}(\mathsf{pk}, C_{\mathsf{ct}}(\mathsf{sk}))$ which is equal to $\mathsf{Encrypt}(\mathsf{pk}, x)$. We note the properties that we want from such a procedure. Let's assume that our SWHE scheme can support d levels of computation.

- The public parameters should post a fresh encryption of the secret key (with noise B). If the homomorphic evaluation of circuit C_{ct} consumes $d' < d$ levels, then the refreshed ciphertext can still support $d - d'$ levels of computation. Arguing that C_{ct} can be computed in $d' < d$ levels will be the main challenge we face in this lecture.
- The security requires that the scheme should be secure even if the public key includes a copy of the decryption key. This requires us to make a circular security assumption. (Open Question: Get FHE without circular security from LWE - without going through the route of indistinguishability obfuscation).

10.1 GSW Encryption

Let's take a closer look on how to achieve bootstrapping for GSW encryption.

Setup: Set $\mathsf{pk} = A \in \mathbb{Z}_q^{n \times m}$ and $\mathsf{sk} = s \in \mathbb{Z}_q^n$ where $s^\top A = e^\top$ such that $e \leftarrow \chi^m$ (χ^m denotes the guassian error distribution and consists of small values with high probability).

$\mathsf{Encrypt}(\mathsf{pk}, \mu \in \{0,1\})$: We compute $C \leftarrow AR + \mu \cdot G$.

$\mathsf{Decrypt}(\mathsf{sk}, C)$: Compute $s^\top C$ and round.

Correctness Specifically, $s^\top C = s^\top AR + \mu \cdot s^\top G = e^\top R + \mu \cdot s^\top G$. Note that $s^\top G = [-\tilde{s}^\top | 1]G$ and we can analyze the last column to observe if it's closer to 0 or $q/2$ to decide if the message is 0 or 1.

Analyzing the depth of the decryption circuit Consider that our scheme can support multiplications and additions and circuits with depth d, i.e. $q > m^{O(d)}$. We now examine the depth of implementing GSW decryption. Since we only need to examine depth, we perform the computation $s^\top C = s^\top [C_1 | \ldots | C_m]$ in parallel and focus on evaluating one column. Note that this simplification does not increase the depth. Let the decryption circuit take in input $s \in \mathbb{Z}_q^n$ has hardwired $c_m \in \mathbb{Z}_q^n$ and computes $\mathsf{round}(s^\top c_m \mod q)$.

- We can write $s^\top c_m = \sum_{i=1}^{n} \sum_{j=0}^{\lceil \log q \rceil} s_{i,j} \cdot (2^j \cdot c_{m,i} \mod q)$ where $s_{i,j}$ denotes the j^{th} bit of i^{th} component of s, $c_{m,i}$ denotes the binary representation of the i^{th} component of c_m.
- Since c_m is available during setup of the circuit and 2^j simply denotes the left shift operation. $(2^j \cdot c_{m,i} \mod q)$ can be hardwired in the circuit.
- $s_{i,j} \cdot (2^j \cdot c_{m,i} \mod q)$ denotes one AND gate operation on q bits.
- Computing $c \top c_m$ requires computing $n\lceil \log q \rceil$ additions on q bits. Using an addition tree, computing n additions on values over k bits can be performed in $O(\log n + \log k)$ depth. Thus the full computation can be performed in $O(\log n + \log \log q)$ depth.
- Finally, we need to compute the modulo operation i.e. $s^\top c_m \mod q$. If q is a power of two, this is easy and just involves ignoring the higher order bits. We can also do this by brute force, i.e. subtract our expression by atmost $n\lceil \log q \rceil$ multiples of q. This is because $s^\top c_m$ is the addition of $n\lceil \log q \rceil$ values. We can compute all possible multiples of q and set it up so that we choose the one that is within $\mathbb{Z}_q$.

  Note that selection is a tree of AND gates and would also take computation depth $O(\log n + \log \log q)$.
- Rounding just involves picking the most significant bit and outputting it. Since we set $q < 2^n$ (for security), the final depth of the circuit is $O(\log n + \log \log q)$ and thus $O(\log n)$.

For correctness, we need to satisfy that $q \sim m^{O(\log n)}$. This we can easily set and achieve full FHE functionality from LWE and a circular security assumption. The caveat in our construction is that we need to set $q > m^{O(\log n)}$ i.e. super polynomial in the security parameter. This means that we rely on worst case hardness of GapSVP_γ where γ is super polynomial (the reduction sets the γ dependent on the modulus-to-noise ratio). This means we need to rely on a stronger assumption than regular public key encryption. Next, we see how to do better than this.

10.2 FHE with Polynomial Modulus

We show that we can do better by exploiting the asymmetric noise growth of GSW multiplication. Recall that if $C_1 = AR_1 + \mu_1 \cdot G$ and $C_2 = AR_2 + \mu_2 \cdot G$.
Then $C_x = C_1 G^{-1}(C_2) = A(R_1 G^{-1}(C_2) + \mu_1 R_2) + \mu_1 \mu_2 G$. Let $R_x = R_1 G^{-1}(C_2) + \mu_1 R_2$ and $\|R_x\|_\infty \leq \|R_1\|_\infty \cdot \mathsf{poly}(m) + \|R_2\|_\infty$.

Suppose we have $C_1, \ldots, C_t$ with noise $R_1, \ldots, R_t$ where for every $i \in [t]$, $\|R_i\|_\infty \leq \mathsf{B}$. Consider any sequence of homomorphic multiplication which involves multiplication with one of $C_1, \ldots, C_t$. The noise accumulation after T multiplications is bounded by $T \cdot \mathsf{B} \cdot \mathsf{poly}(m)$. Each multiplication increases noise by an additive factor.

Key Takeaway If input to every multiplication is a fresh ciphertext, then noise growth is additive, not multiplicative in the depth! Asymmetric noise growth is thus extremely useful both theoretically (relying on weaker assumptions) and practically (more efficient PIR (Private Information Retrieval) protocls). But how to exploit this for GSW bootstrapping? The analsysis we performed above requires computing inner products. Idea - use Branching Programs!

Layered Branching Programs Any graph that can be decomposed into layers, where there are edges between adjacent layers. On each layer, the program reads some bit of the input (Only one bit is read, and this is same across all nodes in the layer). On the final layer, we end up at either an accepting node that outputs 1 or a rejecting node that outputs 0.

These are useful in complexity theory for capturing space-bounded computations where the width of the branching program denotes the spcae. For this lecture, we focus specifically on Permutation Branching Programs (PBPs), where the transition matrix between layers can be described by a permutation matrix.

Theorem 10.1 (Barrington)**.** *Let $C : \{0,1\}^k \to \{0,1\}$ be a boolean circuit with depth d and fan-in 2 (i.e. each gate has two inputs). Then we can compute C using a Permutation Branching Program (PBP) of length $\ell \leq 4^d$ and width 5.*

In particular, observe that if $d = O(\log n)$, then the length of the PBP is $\leq 4^{O(\log n)} = \mathsf{poly}(n)$. We describe our new scheme, let $\mathsf{PBP} = (\mathsf{inp}, \{M_{i,0}, M_{i,1}\}_{i\in[\ell]})$ be a permutation branching program on input $x \in \{0,1\}^n$ with length ℓ and width w.

- $\mathsf{inp} : [\ell] \to [n]$, specifies which bit of the input to read in a given layer.
- For every $i \in [\ell]$, $M_{i,0}, M_{i,1} \in \{0,1\}^{w\times w}$ denote the transition for reading 0 or 1 respectively in layer i. Note that these our permutation matrices.
- Let $v_0 \in \{0,1\}^w = \begin{bmatrix} 1 \\ \vdots \\ 0 \end{bmatrix}$ be initial state.
- Let $t \in \{0,1\}^w$ be indicator for accepting states in output layer.
- We can compute $\mathsf{PBP}(x) = t^\top A_{\ell,x_{\mathsf{inp}(\ell)}} \cdots A_{1,x_{\mathsf{inp}(1)}} \cdot v_0$.

Our decryption circuit of depth $O(\log n)$ can be written as a PBP of length $\mathsf{poly}(n)$. To compute homomorphically, given fresh encryption of bits of $x_{\mathsf{inp}(i)}$ we can homomorphically compute $A_{1,x_{\mathsf{inp}(i)}} = x_{\mathsf{inp}(i)}A_{i,1} + (1 - x_{\mathsf{inp}(i)})A_{i,0}$. If encryptions of x have noise at most B, then encryptions of $A_{i,\mathsf{inp}(i)}$ have noise at most $2\mathsf{B}$.

We can thus homomorphically compute $t^\top \left[A_{\ell,x_{\mathsf{inp}(\ell)}} \cdots A_{1,x_{\mathsf{inp}(1)}}\right] \cdot v_0$. Note that each product invoves at least one "fresh" ciphertext $A_{i,x_{\mathsf{inp}(i)}}$, so by assumption and asymmetric noise growth of GSW multiplication, overall noise is $\ell \cdot \mathsf{B} \cdot \mathsf{poly}(m)$. Overall noise from bootstrapping is thus $\mathsf{poly}(n)$. For correctness, it suffices to use $q = \mathsf{poly}(n)$, and thus we can achieve FHE with polynomial modulus q.

CS 395T: Topics in Cryptography February 23, 2021

Lecture 11: Lattice-Based Key Exchange

Lecturer: David Wu *Scribe: Charlotte LeMay*

11.1 Homomorphically Evaluating Decryption

A key step of the bootstrapping technique that makes unbounded-depth homomorphic encryption work is the homomorphic evaluation of the decryption function. Specifically, to refresh a ciphertext ct such that $\mathsf{Decrypt}(\mathsf{sk}, \mathsf{ct}) = x$, we first find the circuit $C_{\mathsf{ct}}(\mathsf{sk}) := \mathsf{Decrypt}(\mathsf{sk}, \mathsf{ct})$, then homomorphically evaluate C_{ct} on a fresh encryption of sk, $\mathsf{Encrypt}(\mathsf{pk}, \mathsf{sk})$. If this fresh ciphertext permits d levels of multiplication, and evaluating C_{ct} uses up only d', then the generated ciphertext will allow for $d - d'$ further levels of multiplication. This is the key to allowing homomorphic evaluation of unbounded-depth arithmetic circuits.

To finish the construction begun in last lecture, it is necessary to show how decryption can be expressed in as an arithmetic circuit, so that it can be homomorphically evaluated.

Recall that GSW decryption can be computed by a Boolean circuit of depth $d = O(logn)$.

We also have Barrington's theorem: Any Boolean circuit of depth d can be computed by a permutation branching program of length 4^d and width 5.

Therefore the GSW decryption circuit can be computed by a Permutation Branching Program of length $\ell = \mathsf{poly}(n)$ and width 5.

Let us represent the program by $PBP = (\mathsf{inp}_i, M_{i,0}, M_{i,1})$, where $\mathsf{inp}_i \in [n]$ is the bit of the input read at layer i of the program, and $M_{i,b} \in \{0,1\}^{5\times 5}$, $b \in \{0,1\}$, represents the matrix form of the permutation performed at layer i when the read value of x_{inp_i} is b.

Our goal is now to arithmetize PBP, that is, to convert it into an arithmetic circuit, so that it can be evaluated homomorphically.

11.1.1 Arithmetizing the PBP

Suppose that the start state of the program is the first node of the first layer, and that some designated set of nodes in the final layer are accept states. Then if $v_0 = \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}$ represents the state of the initial layer, and $t \in \{0,1\}^5$ is an indicator for the accept states, the action of PBP on $x = x_1 x_2 \cdots x_n$ can be represented by

$$PBP(x) = t^T \left(M_{\ell, x_{\mathsf{inp}_\ell}} \cdots M_{2, x_{\mathsf{inp}_2}} M_{1, x_{\mathsf{inp}_1}} v_0 \right).$$

This is a certainly a sequence of additions and multiplications, but it is not yet an arithmetization since it involves conditional behavior – the choice of which matrices to multiply depends on x. To eliminate this dependence, we can set $M_i = x_{\mathsf{inp}_i} \cdot M_{i,1} + (1 - x_{\mathsf{inp}_i}) \cdot M_{i,0}$, and write

$$PBP(x) = t^T M_\ell \cdots M_2 M_1 v_0.$$

This PBP can now be computed with GSW homomorphism.

11.1.2 Noise Discussion

The PBP above involves only a series of ℓ matrix multiplications, so if the initial noise of the input x is B, the noise after homomorphically evaluating the decryption circuit is $\mathsf{poly}(\ell, m) \cdot B$. Since $\ell = \mathsf{poly}(n)$ and m is also

polynomial in n, this means the overall noise is still $\mathsf{poly}(n)$, which will be asymptotically smaller than q. This ensures that correctness holds (the result is the encryption of x under the public key) except with negligible probability.

Bootstrapping with polynomial noise accumulation is secure assuming GapSVP_γ with $\gamma = \mathsf{poly}(n)$, and the circular security assumption.

11.2 Regev Encryption of Vectors

11.2.1 Protocol

Previously Regev encryption gave LWE-backed encryption of single bits. To get a more efficient rate of communication, we can adapt the scheme to encrypt multiple bits at once. This will look like:

$$\begin{aligned}
\mathsf{Setup}(1^\lambda) :& A \xleftarrow{R} \mathbb{Z}_q^{n\times m} \\
& S \xleftarrow{R} \mathbb{Z}_q^{n\times \ell} \\
& E \leftarrow \chi^{m\times \ell} \\
& B \leftarrow S^T A + E^T \\
& \text{Output} : \mathsf{pk} \leftarrow (A, B^T), \mathsf{sk} \leftarrow S \\
\mathsf{Encrypt}(\mathsf{pk}, \mu \in \mathbb{Z}_2^\ell) :& r \xleftarrow{R} \{0,1\}^m \\
& \text{Output} : \mathsf{ct} \leftarrow (Ar, B^T r + \mu \cdot \left\lfloor \frac{q}{2} \right\rceil) \\
\mathsf{Decrypt}(\mathsf{sk}, \mathsf{ct} = (\mathsf{ct}_0, \mathsf{ct}_1)) :& \text{Output} : \left\lfloor \mathsf{ct}_1 - s^T \mathsf{ct}_0 \right\rceil
\end{aligned}$$

11.2.2 Correctness

The change from the one-bit version we introduced previously is that B is now a matrix with ℓ entries rather than simply a vector. What we have done is essentially reuse the matrix A on ℓ different bits, in that each row of B is a Regev encryption of the corresponding bit of μ, using the corresponding columns of S and E. Correctness therefore follows from correctness in the one-bit case.

11.2.3 Security

As for security, B is indistinguishable from uniform by LWE. More technically, we can construct a hybrid argument where we argue that each successive row of B is secure by LWE. Ar is indistinguishable from uniform by the leftover hash lemma.

11.2.4 Savings

One issue that the scheme presents is the key size – we require public keys (A, B^T) of size $\Theta(mn) = \Theta(n^2 \log q)$. For q of a standard size of approximately 2^{12}, shaving off the $\log q$ factor would result in a twelve-fold increase in communication efficiency, which is not inconsiderable. We can remove the $\log q$ factor in the public key by letting A be an $n \times n$ matrix, and adding a right-multiplicative matrix to the secret key. Details of this are left to the student as an exercise.

11.3 Key Exchange from LWE

11.3.1 Protocol

In analogy with the Diffie-Hellman key exchange protocol, which I (your scribe) will not reproduce here, we want to define a protocol for key exchange relying on LWE. It will operate as follows: Alice will sample $A \xleftarrow{R} \mathbb{Z}_q^{n\times n}$,

$S_1, E_1 \leftarrow \chi^{n \times k_1}$, $B_1^T \leftarrow S_1^T A + E_1$, and send (A, B_1) to Bob. Bob will sample $S_2, E_2 \leftarrow \chi^{n \times k_2}$, $B_2 \leftarrow As_2 + E_2$, and send B_2 to Alice. Alice then computes $C \leftarrow \lfloor S_1^T B_2 \rceil$ and $C' \leftarrow \lfloor B_1^T S_2 \rceil$. As we will show below, we expect with high probability that $C = C'$ will then be a shared value between Alice and Bob, usable as a key.

11.3.2 Correctness

To prove correctness, notice that

$$S_1^T B_2 = S_1^T A S_2 + S_1^T E_2$$

and

$$B_1^T S_2 = S_1^T A S_2 + E_1^T S_2.$$

Since E_1, E_2, S_1, S_2 are sampled from the error distribution, they have small value, and we expect $|S_1^T E_2| < B$ and $|E_1^T S_2| < B$ for some bound B (where $|\cdot|$ denotes the largest entry). Suppose that when we round we extract a T-bit long value from each of the $k_1 \cdot k_2$ entries of C, meaning we round to one of 2^T values. There are then 2^T "crossover points" in $\mathbb{Z}_q$ at which an error of magnitude B could change the value we extract. There are at most $2B$ values in the neighborhood of each of these crossover points that could cause an error, so the probability that any entry rounds incorrectly is at most $\frac{2^{T+1}B}{q}$. Using a union bound, the probability that one or more of the $k_1 \cdot k_2$ entries causes an error is then at most $\frac{2^{T+1}Bk_1k_2}{q}$. Appropriate choices of the parameters can render this quantity negligible, so $C = C'$ with all but negligible probability.

11.3.3 Security

To show security, we must show that the observed transcript

$$(A, B_1^T = S_1^T A + E_1^T, B_2 = AS_2 + E_2, \lfloor S_1^T B_2 \rceil)$$

leaks no information to probabilistic polynomial-time adversaries. The first step is to say that, based on LWE, B_2 is indistinguishable from a uniform random matrix U_2, making the transcript equivalent to

$$(A, B_1^T = S_1^T A + E_1^T, U_2, \lfloor S_1^T U_2 \rceil).$$

For the next step, we must be careful. We cannot simply replace B_1 with uniform, since both it and $\lfloor S_1^T U_2 \rceil$ have a dependence on S_1^T. What we will do instead is say that this distribution is statistically close to a distribution with an additional error $\tilde{E} \leftarrow \chi^{k_1 \times k_2}$ added before rounding $S_1^T U_2$. Therefore the above transcript is statistically close to

$$(A, B_1^T = S_1^T A + E_1^T, U_2, \left\lfloor S_1^T U_2 + \tilde{E} \right\rceil).$$

NOW we can apply LWE to say that since

$$S_1^T \left[A \| U_2\right] + \left[E_1^T \| \tilde{E}\right] = \left[B_1^T \| S_1^T U_2 + \tilde{E}\right]$$

is indistinguishable from a uniform $[U_1 \| U_3]$, this means that the above transcript is indistinguishable from

$$(A, U_1, U_2, \lfloor U_3 \rceil),$$

which clearly leaks no information. Therefore by LWE, the key exchange protocol is secure.

CS 395T: Topics in Cryptography February 28, 2021

Lecture 12: Homomorphic Signatures

Lecturer: David Wu *Scribe: Jiahui Liu*

12.1 Definition

A homomorphic signature scheme HS consists of the following algorithms:

- $\mathsf{KeyGen}(1^\lambda) \to (\mathsf{sk}, \mathsf{vk})$: takes in a security parameter and outputs a signing key, verification key pair $(\mathsf{sk}, \mathsf{vk})$.
- $\mathsf{Sign}(\mathsf{sk}, x) \to \sigma$: takes in a signing key sk and a message x; outputs signature σ.
- $\mathsf{Verify}(\mathsf{vk}, x, \sigma)$: takes a verification key vk, message x and claimed signature σ; outputs 0 for Reject or 1 for Accept.
- $\mathsf{Eval}(f, x, \sigma)$: takes a function description f, message x and signature σ for x; outputs a signature $\sigma_{f,f(x)}$.
- $\mathsf{Preprocess}(\mathsf{vk}, f) \to \mathsf{vk}_f$: takes in a verifivation key vk and function description f, outputs vk_f.

A homomorphic signature scheme should satisfy the following properties:

Correctness Honestly generated signatures should pass verification with almost 1 probability:

$$\Pr\left[\mathsf{Verify}(\mathsf{vk}_f, f(x), \sigma_{f(x)}) = 1 \;\middle|\; \begin{array}{c} (\mathsf{sk}, \mathsf{vk}) \leftarrow \mathsf{KeyGen}(1^\lambda, k), \\ \sigma_x \leftarrow \mathsf{Sign}(\mathsf{sk}, x), \\ \sigma_{f(x)} \leftarrow \mathsf{Eval}(f, x, \sigma_x) \\ \mathsf{vk}_f \leftarrow \mathsf{Preprocess}(\mathsf{vk}, f) \end{array}\right] \geq 1 - \mathsf{negl}(\lambda)$$

Unforgeability We describe the following security game:

1. The challenger runs KeyGen to obtain $(\mathsf{sk}, \mathsf{vk})$ and sends vk to adversary $\mathcal{A}$.
2. $\mathcal{A}$ sends in a message x; the challenger signs the message $\sigma_x \leftarrow \mathsf{Sign}(\mathsf{sk}, x)$ and gives σ_x to $\mathcal{A}$.
3. $\mathcal{A}$ then provides $f, y, \sigma_{f,y}$, where f is a function description, y is an alleged evaluation and $\sigma_{f,y}$ is a signature.
4. The challenger checks if $f(x) = y$ and if $\mathsf{Verify}(\mathsf{vk}_f, y, \sigma_{f,y}) = 1$ where $\mathsf{vk}_f \leftarrow \mathsf{Preprocess}(\mathsf{vk}, f)$. $\mathcal{A}$ wins if if $f(x) \neq y$ and $\mathsf{Verify}(\mathsf{vk}_f, y, \sigma_{f,y}) = 1$.

An unforgeable homomorphic signature scheme should satisfy: for any PPT adversary $\mathcal{A}$, there exists a negligible function $\mathsf{negl}(\cdot)$ such that for all λ, the followig holds:

$$\Pr[\mathcal{A} \text{ wins the above game }] \leq \mathsf{negl}(\lambda)$$

.

Succinctness A non-trivial HS should also satisfy: size of $\sigma_{f,y}$(signature after evaluation $\mathsf{Eval}(f, x, \sigma_x)$ where $y = f(x)$) should be $|f(x)| \cdot \mathsf{poly}(\lambda)$. In particular, the size of $\sigma_{f,y}$ should NOT depend on $|f|$ or $|x|$.

Otherwise, if we don't consider succinctness, we can construct a trivial scheme:

- $\mathsf{Eval}(f, x, \sigma_x) \to \sigma_y = (\sigma_x, x, f)$. The server(evaluator) gives y, σ_y to the client.
- The client can then verify with the algorithm: parse σ_y as (σ_x, x, f). Check if $y = f(x)$ and run $\mathsf{Verify}(\mathsf{vk}, x, \sigma_x)$.

12.2 Construction

We next give a construction for HS, bearing similarities to the GSW encryption scheme.

- $\mathsf{KeyGen}(1^\lambda)$: set lattice parameter $n = n(\lambda), q = q(\lambda)$. Let $s = s(\lambda)$ be Gaussian width parameter for pre-iamge sampling.
  1. Sample $(\mathbf{A}, \mathbf{T}) \leftarrow \mathsf{TrapGen}(n, q)$ where $\mathbf{A} \in \mathbb{Z}_q^{n \times m}, \mathbf{T} \in \{0,1\}^{m \times t}, t = n \lceil \log q \rceil, \mathbf{AT} = \mathbf{G}$ and $\mathbf{G}$ is the gadget matrix.
  2. Sample random $\mathbf{B}_1, \cdots, \mathbf{B}_\ell \leftarrow \mathbb{Z}_q^{n \times t}$.
  3. Output $\mathsf{vk} = (\mathbf{A}, \mathbf{B}_1, \cdots, \mathbf{B}_\ell)$ and $\mathsf{sk} = \mathbf{T}$.
- $\mathsf{Sign}(\mathsf{sk}, x \in \{0,1\}^\ell)$:
  1. Compute preimage sampling $\mathbf{R}_i \leftarrow \mathsf{SamplePre}(\mathbf{A}, \mathbf{T}, \mathbf{B}_i - x_i\mathbf{G}) \in \mathbb{Z}_q^{m \times t}$, where $\mathbf{G}$ is the gagdet matrix.
  2. Note that we have:
  $$\begin{aligned}\mathbf{A}[\mathbf{R}_1|\cdots|\mathbf{R}_\ell] &= [\mathbf{B}_1 - x_1\mathbf{G}|\cdots|\mathbf{B}_\ell - x_\ell\mathbf{G}] \\ &= [\mathbf{B}_1|\cdots|\mathbf{B}_\ell] - x \otimes \mathbf{G}\end{aligned}$$
  3. Output $\sigma = (\mathbf{R}_1, \cdots, \mathbf{R}_\ell)$.
- $\mathsf{Verify}(\mathsf{vk}, x, \sigma)$:
  1. Parse vk as $(\mathbf{A}, \mathbf{B}_1, \cdots, \mathbf{B}_\ell)$ and σ as $(\mathbf{R}_1, \cdots, \mathbf{R}_\ell)$.
  2. Check $\|\mathbf{R}_i\| \leq \beta$ where $\beta = s \cdot \omega(\sqrt{\log n})$.
  3. Check if $\mathbf{A}[\mathbf{R}_1|\cdots|\mathbf{R}_\ell] = [\mathbf{B}_1|\cdots|\mathbf{B}_\ell] - x \otimes \mathbf{G}$
  4. If both of the above verifications pass, then output Accept; otherwise output Reject.
- $\mathsf{Eval}(f, x, \sigma)$: Now we show how we compute homomorphic evaluation on signatures, along with how we generate the verification key associated with function f:

  Addition To obtain sum of two bits $x_i + x_j$:
  $$\begin{aligned}\mathbf{R}_+ &= \mathbf{R}_i + \mathbf{R}_j \\ \mathbf{B}_+ &= \mathbf{B}_i + \mathbf{B}_j\end{aligned}$$
  The verification algorithm will then check if
  $$\mathbf{AR}_+ = \mathbf{B}_+ - (x_i + x_j)\mathbf{G}$$
  where $\mathbf{R}_+$ is the new signature generated for $x_i + x_j$ and $\mathbf{B}_+$ is the new verification component associated with the addition operation.

Multiplication To obtain product $x_i \cdot x_j$: We have $\mathbf{AR}_i = \mathbf{B}_i - x_i\mathbf{G}$ and $\mathbf{AR}_j = \mathbf{B}_j - x_j\mathbf{G}$. We would want $\mathbf{R}_\times$(a function of $\mathbf{R}_i, \mathbf{R}_j, x_i, x_j$ and is also short) and $\mathbf{B}_\times$(a function of $\mathbf{B}_i, \mathbf{B}_j$ but independent of x_i, x_j since the verification algorithm does not know x) so that $\mathbf{AR}_\times = \mathbf{B}_\times - x_i x_j \cdot \mathbf{G}$.

By the following computation:

$$\begin{aligned}
\mathbf{AR}_i = \mathbf{B}_i - x_i\mathbf{G} \rightarrow \mathbf{B}_i &= \mathbf{AR}_i + x_i\mathbf{G} \\
\text{Let } \mathbf{B}_i\mathbf{G}^{-1}\mathbf{B}_j &= (\mathbf{AR}_i + x_i\mathbf{G})\mathbf{G}^{-1}\mathbf{B}_j \\
&= \mathbf{AR}_i\mathbf{G}^{-1}\mathbf{B}_j + x_i\mathbf{B}_j \\
&= \mathbf{AR}_i\mathbf{G}^{-1}\mathbf{B}_j + x_i(\mathbf{AR}_j + x_j\mathbf{G}) \\
&= \mathbf{A}(\mathbf{R}_i\mathbf{G}^{-1}\mathbf{B}_j + x_i\mathbf{R}_j) + x_i x_j\mathbf{G}
\end{aligned}$$

Therefore, we would have $\mathbf{R}_\times = \mathbf{R}_i\mathbf{G}^{-1}\mathbf{B}_j + x_i\mathbf{R}_j$ and $\mathbf{B}_\times = \mathbf{B}_i\mathbf{G}^{-1}\mathbf{B}_j$.

Observation 1 $\mathbf{R}_\times = \mathbf{R}_i\mathbf{G}^{-1}\mathbf{B}_j + x_i\mathbf{R}_j$ is a function of the inputs and $\|\mathbf{R}_\times\|_\infty \leq \|\mathbf{R}_j\|_\infty - t + \|\mathbf{R}_i\|_\infty$ $\mathbf{B}_\times = \mathbf{B}_i\mathbf{G}^{-1}\mathbf{B}_j$is a function of only the verification keys, as desired (as in GSW homomorphic multiplication operations).

Observation 2 Note that $\mathbf{R}_+ = \mathbf{R}_i + \mathbf{R}_j = [\mathbf{R}_i|\mathbf{R}_j]\begin{bmatrix}\mathbf{I}_t \\ \mathbf{I}_t\end{bmatrix}$ and $\mathbf{R}_\times = \mathbf{R}_i\mathbf{G}^{-1}\mathbf{B}_j + x_i\mathbf{R}_j = [\mathbf{R}_i|\mathbf{R}_j]\begin{bmatrix}\mathbf{G}^{-1}\mathbf{B}_j \\ x_i\mathbf{I}_t\end{bmatrix}$. The matrices on the right side of are small linear functions of $\mathbf{R}_i, \mathbf{R}_j$.

Since we already have addition(XOR) and multiplication(AND), we can compose the above operations to obtain signature $\mathbf{R}_{f(x)}$ for evaluation $f(x)$.

Moreover, multiplication scales noise by a factor of t. If f can be computed by a circuit of depth d, then $\|\mathbf{R}_t\|_\infty \leq t^{O(d)} = (n \log q)^{O(d)}$.

- $\mathsf{Preprocess}(\mathsf{vk}, f)$ and $\mathsf{Verify}(\mathsf{vk}_f, f(x), \sigma_{f(x)})$: To verify a signature $\sigma_{f(x)} = \mathbf{R}_{f(x)}$ on f and $y = f(x)$(supposedly), one can compute $\mathbf{B}_f$ from $\mathbf{B}_1, \cdots, \mathbf{B}_\ell$; then check if $\mathbf{AR}_{f(x)} = \mathbf{B}_f - y \cdot \mathbf{G}$ and then check if $\|\mathbf{R}_t\|_\infty \leq t^{O(d)}$.

  Now if we have $\mathbf{AR}_i = \mathbf{B}_i - x_i\mathbf{G}$ for each x_i, then we can verify $\mathbf{AR}_f = \mathbf{B}_f - f(x) \cdot \mathbf{G}$ by computing $\mathbf{B}_f$ from evaluating f on $\mathbf{B}_1 \cdots, \mathbf{B}_\ell$.

More generally, we can write:

$$\begin{aligned}
\mathbf{R}_{f(x)} &= [\mathbf{R}_1|\cdots|\mathbf{R}_\ell] \cdot \mathbf{H}_{f,x} \\
\mathbf{AR}_{f(x)} &= \mathbf{A}[\mathbf{R}_1|\cdots|\mathbf{R}_\ell] \cdot \mathbf{H}_{f,x} \\
&= [\mathbf{B}_1 - x_1\mathbf{G}|\cdots|\mathbf{B}_1 - x_\ell\mathbf{G}] \cdot \mathbf{H}_{f,x} \\
&= \mathbf{B}_f - f(x) \otimes \mathbf{G}
\end{aligned}$$

where $\mathbf{H}_{f,x} \in \mathbb{Z}_q^{\ell t \times t}$ and is small ($\|\mathbf{H}_{f,x}\|_\infty \leq (n \log q)^{O(d)}$).

Note that in the above equations, computing $\mathbf{B}_f$ from $\mathbf{B}_1 \cdots, \mathbf{B}_\ell$ and f is input independent; computing $[\mathbf{B}_1 - x_1\mathbf{G}|\cdots|\mathbf{B}_1 - x_\ell\mathbf{G}] \cdot \mathbf{H}_{f,x}$ is input dependent. These are important homomorphic operations in lattice based schemes.

CS 395T: Topics in Cryptography March 2, 2022

Lecture 13: Homomorphic Signatures and Commitments

Lecturer: David Wu *Scribe: Alexander Burton*

13.1 Homomorphic Signature Schemes

Recall the GVW homomorphic signature scheme discussed in the previous lecture, with the key points summarized below:

- The verification key is $\mathsf{vk} = (\mathbf{A}, \mathbf{B}_1, \ldots, \mathbf{B}_\ell)$, where $\mathbf{A} \in \mathbb{Z}_q^{n \times m}$, and $\mathbf{B}_1, \ldots, \mathbf{B}_\ell \in \mathbb{Z}_q^{m \times t}$ are random.
- The signing key is $\mathsf{sk} = \mathbf{T}$, a trapdoor for $\mathbf{A}$.
- A signature of a message $x \in \{0,1\}^\ell$ is a collection $\mathbf{R}_1, \ldots, \mathbf{R}_\ell \in \mathbb{Z}_q^{m \times t}$ of short matrices satisfying $\mathbf{A}\mathbf{R}_i = \mathbf{B}_i - x_i \mathbf{G}$.
- Homomorphic evaluation is done with the addition matrix $\mathbf{B}_+ = \mathbf{B}_i + \mathbf{B}_j$ and the multiplication matrix $\mathbf{B}_\times = \mathbf{B}_i \cdot \mathbf{G}^{-1}(\mathbf{B}_j)$. The evaluation of a function f of multiplicative depth d is then given by

  $$\mathbf{B}_{f,x} = \mathbf{A}\mathbf{R}_{f,x} - f(x) \cdot \mathbf{G}$$

  where $\mathbf{R}_{f,x} = \left[\mathbf{R}_1 \mid \cdots \mid \mathbf{R}_\ell\right] \cdot \mathbf{H}_{f,x}$ and $\|\mathbf{H}_{f,x}\| \leq \beta := (n \log q)^{O(d)}$.

13.2 Unforgeability

We turn to the question of unforgeability, for which there are two similar notions.

Definition 13.1. A homomorphic signature scheme is said to be **adaptively unforgeable** if there does not exist an adversary with a non-negligible advantage in the following security game:

Adversary		Challenger
		$(\mathsf{vk}, \mathsf{sk}) \leftarrow \mathsf{KeyGen}\left(1^\lambda\right)$
	$\xleftarrow{\mathsf{vk}}$	
	$\xrightarrow{x}$	$\sigma_x \leftarrow \mathsf{Sign}(\mathsf{sk}, x)$
	$\xleftarrow{\sigma_x}$	
	$\xrightarrow{f, y, \sigma_{f,y}}$	Check $y \neq f(x)$ and $\mathsf{Verify}(\mathsf{vk}_f, y, \sigma_{f,y}) = 1$.

Definition 13.2. A homomorphic signature scheme is said to be **selectively unforgeable** if there does not exist an adversary with a non-negligible advantage in the following security game:

Adversary		Challenger
	$\xrightarrow{x}$	
	$\xleftarrow{\mathsf{vk}}$	$(\mathsf{vk}, \mathsf{sk}) \leftarrow \mathsf{KeyGen}\left(1^\lambda\right)$
		$\sigma_x \leftarrow \mathsf{Sign}(\mathsf{sk}, x)$
	$\xleftarrow{\sigma_x}$	
	$\xrightarrow{f, y, \sigma_{f,y}}$	Check $y \neq f(x)$ and $\mathsf{Verify}(\mathsf{vk}_f, y, \sigma_{f,y}) = 1$.

The only difference between the two definitions is when the adversary sends the message x. Selective unforgeability is a weaker notion than adaptive unforgeability, and much easier to work with; we will prove selective unforgeability of our signature scheme. It is not too difficult to extend our scheme to an adaptively secure one (compose with a vanilla signature scheme).

Theorem 13.3. *Under* LWE*, the given scheme is selectively unforgeable.*

Proof. We use a hybrid argument. Define the following hybrid games:

- Hyb_0 is the normal selective unforgeability game.
- Hyb_1 is Hyb_0, except for key generation, instead of choosing $\mathbf{B}_i$ at random, the challenger chooses $\mathbf{R}_i \in \{0,1\}^{m \times t}$ at random and sets $\mathbf{B}_i \leftarrow \mathbf{A}\mathbf{R}_i + x_i\mathbf{G}$. We can output $\mathsf{vk} = (\mathbf{A}, \mathbf{B}_1, \ldots, \mathbf{B}_\ell)$ and $\sigma_x = (\mathbf{R}_1, \ldots, \mathbf{R}_\ell)$.
- Hyb_2 is Hyb_1, except for key generation, instead of choosing $\mathbf{A}$ at random, the challenger chooses a LWE matrix for $\mathbf{A}$:

$$\begin{aligned}
\bar{\mathbf{A}} &\xleftarrow{R} \mathbb{Z}_q^{(n-1)\times m} \\
\bar{\mathbf{s}} &\xleftarrow{R} \mathbb{Z}_q^{n-1} \\
\mathbf{e} &\xleftarrow{R} \chi^m \\
\mathbf{A} &\leftarrow \begin{bmatrix} \bar{\mathbf{A}} \\ \bar{\mathbf{s}}^T\bar{\mathbf{A}} + \mathbf{e}^T \end{bmatrix}
\end{aligned}$$

  Note that we do not need trapdoor information for $\mathbf{A}$ since we only need to sign x, which is programmed into the verification key.

Observe that Hyb_0 and Hyb_1 are statistically equivalent, since $\mathbf{A}\mathbf{R}_i$ and thus $\mathbf{A}\mathbf{R}_i + x_i\mathbf{G}$ are statistically uniform by Leftover Hash Lemma. Furthermore Hyb_1 and Hyb_2 are computationally equivalent, since by LWE, random $\mathbf{A}$ and LWE $\mathbf{A}$ are computationally indistinguishable.

To finish the reduction, we can show that Hyb_2 produces a contradiction unconditionally. Let $\mathcal{A}$ be an adversary that wins Hyb_2; suppose it produces $\mathbf{R}^*, f, y$ where $\|\mathbf{R}^*\| \leq \beta$, $y \neq f(x)$, and

$$\mathbf{A}\mathbf{R}^* = \mathbf{B}_f - y \cdot \mathbf{G}.$$

Now consider an honest evaluation of $f(x)$, i.e. $\mathbf{R}_{f,x} = \left[\mathbf{R}_1 \mid \cdots \mid \mathbf{R}_\ell\right] \cdot \mathbf{H}_{f,x}$, where:

$$\mathbf{A}\mathbf{R}_{f,x} = \mathbf{B}_f - f(x) \cdot \mathbf{G}.$$

Subtracting and collecting yields

$$\mathbf{A}(\mathbf{R}^* - \mathbf{R}_{f,x}) = \pm\mathbf{G}$$

Multiplying through by the solution vector $\left[-\bar{\mathbf{s}}^T \mid 1\right]$ on the left gives

$$\mathbf{e}^T(\mathbf{R}^* - \mathbf{R}_{f,x}) = \pm\left[-\bar{\mathbf{s}}^T \mid 1\right] \cdot \mathbf{G}$$

Observe now that the left hand side is small, and in particular is much less than q. However, the right hand side is large, since the entries of $\mathbf{G}$ are large ($\approx q/2$). This yields our contradiction. □

13.3 Context-Hiding

Another important security notion is context-hiding. Informally, a context-hiding scheme is one in which computed signatures $\sigma_{f,f(x)}$ reveal nothing about x (except for $f(x)$).

Definition 13.4. A homomorphic signature scheme is said to be (statistically) **context-hiding** if there exists an efficient simulator $\mathcal{S}$ where for any $(\mathsf{vk}, \mathsf{sk}) \leftarrow \mathsf{KeyGen}(1^\lambda)$, $x \in \{0,1\}^\ell$, and $f : \{0,1\}^\ell \to \{0,1\}$,

$$\{\mathsf{vk}, \mathsf{Eval}(\mathsf{vk}, f, \sigma)\} \overset{S}{\approx} \{\mathsf{vk}, \mathcal{S}(\mathsf{sk}, \mathsf{vk}, f, f(x))\}.$$

The current scheme is not context-hiding, since $\mathbf{R}_{f,x}$ depends on $\mathbf{H}_{f,x}$, which is a function of x. However, we can modify the construction to make it context-hiding by re-randomizing the $\mathbf{R}_{f,x}$ at the end.

Construction 13.5. Suppose $\mathbf{A}\mathbf{R}_{f,x} = \mathbf{B}_f - y \cdot \mathbf{G}$ where $\mathbf{R}_{f,x}$ is small and $y \in \{0,1\}$. To re-randomize $\mathbf{R}_{f,x}$, we can use preimage sampling. First, the evaluator can compute from y and vk:

$$\begin{aligned}\mathbf{V} &= \left[\mathbf{A} \mid \mathbf{B}_f + (y-1)\cdot\mathbf{G}\right] \\ &= \left[\mathbf{A} \mid \mathbf{A}\mathbf{R}_{f,x} + (2y-1)\cdot\mathbf{G}\right]\end{aligned}$$

Note that $2y - 1 \in \{-1, 1\}$. Observe that $\mathbf{R}_{f,x}$ is now a trapdoor for $\mathbf{V}$. Thus, we can add a random target $\mathbf{z} \xleftarrow{R} \mathbb{Z}_q^n$ to vk, and make the signature be a short $\mathbf{t}$ such that $\mathbf{V}\mathbf{t} = \mathbf{z}$. The signing algorithm will then compute

$$t \leftarrow \mathsf{SamplePre}(\mathbf{V}, \pm\mathbf{R}_{f,x}, \mathbf{z}, s); \quad s = (n \log q)^{O(d)}.$$

Verification of $\sigma_{f,y}$ can be done by computing $\mathbf{V}$ as above and checking that $\mathbf{V}\sigma_{f,y} = \mathbf{z}$ and $\|\sigma_{f,y}\| \le \beta$.

To prove that this is context hiding, we observe that $t \sim D_{\mathcal{L}^\perp_{\mathbf{z}}(\mathbf{V}),s}$, where $\mathbf{V}$ depends only on $\mathbf{A}, \mathbf{B}_i, f, y$. A homework problem shows that since $\mathbf{V}$ is an extension of $\mathbf{A}$, we can sample from $D_{\mathcal{L}^\perp_{\mathbf{z}}(\mathbf{V}),s}$ using a trapdoor for $\mathbf{A}$. Proof of unforgeability follows the same strategy as the previous proof.

13.4 Dual-Mode Homomorphic Commitment Schemes

It turns out that there is a nice framework that unifies homomorphic encryption and homomorphic signatures. Observe the parallels between these two schemes:

Homomorphic Encryption (GSW)	Homomorphic Signatures (GVW)
pk: a LWE matrix $\mathbf{A}$	vk: a random matrix $\mathbf{A}$
ciphertexts: $\mathbf{C} = \mathbf{A}\mathbf{R} + \mu\mathbf{G}$	signatures: $\mathbf{A}\mathbf{R} = \mathbf{B} + \mu\mathbf{G}$.

The key insight is that GSW homomorphisms are homomorphic in both the message and the randomness.

$$\begin{aligned}&\textit{HE eval / HS verify:} \quad f, \mathbf{C}_1, \dots, \mathbf{C}_\ell \mapsto \mathbf{C}_f \\ &\textit{HS eval:} \quad \left[\mathbf{C}_1 - x_1\mathbf{G} \mid \cdots \mid \mathbf{C}_\ell - x_\ell\mathbf{G}\right]\cdot\mathbf{H}_{f,x} = \mathbf{C}_f - f(x)\cdot\mathbf{G} \\ &\mathbf{A}\cdot\underbrace{\left[\mathbf{R}_1 \mid \cdots \mid \mathbf{R}_\ell\right]\cdot\mathbf{H}_{f,x}}_{\mathbf{R}_{f,x}} \mapsto \mathbf{C}_f = \mathbf{A}\underbrace{\mathbf{R}_{f,x}}_{\text{randomness}} + \underbrace{f(x)}_{\text{message}}\cdot\mathbf{G}.\end{aligned}$$

We can combine this into the following idea.

Construction 13.6. A **dual-mode homomorphic commitment scheme** is (informally) given by the following construction:

- pp: choose $\mathbf{A} \in \mathbb{Z}_q^{n\times m}$ according to some distribution.
- $\mathsf{Commit}(\mu)$: sample $\mathbf{R} \xleftarrow{R} \{0,1\}^{m\times m}$, output $\mathbf{C} = \mathbf{A}\mathbf{R} + \mu\mathbf{G}$.
- $\mathsf{Open}(\mathbf{C})$: reveal $\mathbf{R}$; check $\mathbf{C} = \mathbf{A}\mathbf{R} + \mu\mathbf{G}$ and $\|\mathbf{R}\| \le \beta$.
- $\mathsf{Eval}(\mathbf{C}_i, f)$: compute $\mathbf{R}_{f,x}$ using $\mathbf{H}_{f,x}$. This is no different than GSW homomorphic evaluation.

The remarkable property is that we can get statistical binding/hiding depending on where $\mathbf{A}$ is taken from.

1. If $\mathbf{A}$ is an LWE matrix,
$$\mathbf{A} = \begin{bmatrix} \bar{\mathbf{A}} \\ \bar{\mathbf{s}}^T + \bar{\mathbf{A}} + \mathbf{e}^T \end{bmatrix},$$
then the scheme is **extractable** – given a trapdoor, we can "decrypt" the commitment to the *unique* message (if it exists) for which an opening exists. Furthermore, the scheme is statistically binding and computationally hiding.

2. If $\mathbf{A}$ is a uniform matrix,
$$\mathbf{A} \xleftarrow{R} \mathbb{Z}_q^{n \times m},$$
then the scheme is **equivocable** – given a trapdoor, we can open a commitment to *both* 0 and 1 by preimage sampling. Furthermore, the scheme is statistically hiding and computationally binding.

We can see that the first mode is GSW homomorphic encryption, and the second mode is GVW homomorphic signatures.

CS 395T: Topics in Cryptography March 7, 2022

Lecture 14: Homomorphic Commitments

Lecturer: David Wu *Scribe: Yingchen Wang*

In this lecture we are going to discuss homormorphic commitment and an application of homomorphic commitment to none interactive zero knowlwegde.

14.1 Recap: GSW Homomorphic Commitments

Construction 14.1. GSW homomorphic commitment scheme

- pp: $\mathbf{A} \in \mathbb{Z}_q^{n \times m}$ from some distribution.
- Commit(μ): $\mu \in \{0,1\}$. Sample $\mathbf{R} \leftarrow \mathcal{D}_{\mathbb{Z}^m,s}^{m \times t}$, $s \sim \omega(\sqrt{logn})$. $\mathbf{C} = \mathbf{AR} + \mu\mathbf{G}$, which is just a GSW ciphertext (still sematically secure but cannot use LHL because $\mathbf{R}$ is not binary, instead using Gaussian smoothing lemma).
- Open($\mathbf{C}$): Publish $\mathbf{R} \in \mathbb{Z}_q^{m \times t}$. The verifier checks that $\mathbf{C} = \mathbf{AR} + \mu\mathbf{G}$ and $\|R\|_\infty \leqslant \beta$.
- Eval($\mathbf{C}_i, f$): Given $\mathbf{C}_1 = \mathbf{AR}_1 + \mu_1\mathbf{G}$, ...$\mathbf{C}_l = \mathbf{AR}_l + \mu_l\mathbf{G}$; we can apply GSW homomorphic encryption operation $\mathbf{C}_f = \mathbf{AR}_{f,x} + f(x)\mathbf{G}$, where $\mathbf{R}_{f,x} = [\mathbf{R}_1|...|\mathbf{R}_l] * \mathbf{H}_{f,x}$ and $\mathbf{H}_{f,x}$ is short ($\|\mathbf{H}_{f,x}\|_\infty \leqslant (nlogq)^{O(d)}$ and d is the depth of the computation).

14.2 Dual Mode Commitments

1. **A extractable commitment scheme is statistically binding**. Suppose $\mathbf{A} \in \mathbb{Z}_q^{n \times m}$ is a LWE matrix, where $\mathbf{A} = \begin{bmatrix} \bar{\mathbf{A}} \\ \bar{\mathbf{s}}^T\bar{\mathbf{A}} + \mathbf{e}^T \end{bmatrix}$. Then the commitment scheme is extractable, which means that given a trapdoor td and Commitment $\mathbf{C}$, we can recover the committed message μ. The reason is that: consider $\mathbf{C} = \mathbf{AR} + \mu\mathbf{G}$, the trapdoor is $\mathbf{s} = [-\bar{\mathbf{s}}|1]$. We can compute $\mathbf{s}^T\mathbf{C} = \mathbf{s}^T\mathbf{AR} + \mu\mathbf{s}^T\mathbf{G} = \mathbf{e}^T\mathbf{R} + \mu\mathbf{s}^T\mathbf{G}$, where $\mathbf{e}^T\mathbf{R}$ is short, so we can recover μ.
   We can only open to the value that is extracted, because we cannot open the commitment to 2 values simultaneously. Suppose there exists $\mathbf{R}_1, \mathbf{R}_2$ such that $\mathbf{C} = \mathbf{AR}_1 + \mu_1\mathbf{G} = \mathbf{AR}_2 + \mu_2\mathbf{G}$, we have $\mathbf{A}(\mathbf{R}_1 - \mathbf{R}_2) = \pm\mathbf{G}$ and $\mathbf{s}^T\mathbf{A}(\mathbf{R}_1 - \mathbf{R}_2) = \mathbf{e}^T(\mathbf{R}_1 - \mathbf{R}_2) = \pm\mathbf{s}^T\mathbf{G}$. $\mathbf{e}^T(\mathbf{R}_1 - \mathbf{R}_2)$ is small whereas $\mathbf{s}^T\mathbf{G}$ contains large entries. So we prove by contradiction that only one value can be opened by this scheme (a extractable commitment scheme must statistically binding).

2. **A equivocable commitment scheme is statistically hiding**. Suppose $\mathbf{A} \xleftarrow{R} \mathbb{Z}_q^{n \times m}$ is uniform random matrix. Then the equivalent scheme is equivocable.
   Given a trapdoor td and any commitment $\mathbf{C}$, we can open $\mathbf{C}$ to both 0 and 1. The td can be constructed as the matrix $\mathbf{T}$ such that $\mathbf{AT} = \mathbf{G}$. Given $\mathbf{C}$, $\mathbf{T}$, and $\mu \in \{0,1\}$, our goal is to find $\mathbf{R}$ such that $\mathbf{C} = \mathbf{AR} + \mu\mathbf{G}$ and $\|R\| < \beta$. To solve this relation, we can use the preimage sampler. We can sample $\mathbf{R} \xleftarrow{R} \mathsf{SamplePre}(\mathbf{A}, \mathbf{T}, \mathbf{C} - \mu\mathbf{G}, \mathbf{s})$ for both $\mu = 0$ or $\mu = 1$. In such a way, the commitment can be opened to both messages, so it is equivocable. Equivocation implies that the commitment scheme is statistically hiding because it can open to any message, so the message is completely hidden.

Can we have both? statistically binding and statistically hiding? Although we cannot get statistically for both, we can have statistically binding and computationally hiding or statistically hiding and computationally binding.

For example, the second scheme is above is both statistically hiding and computationally binding because:

- Hybrid_0: Adversary breaks binding. Given pp as $\mathbf{A}$, adversary can output $\mathbf{C}$, $\mathbf{R}_1$, $\mathbf{R}_2$, such that $\mathbf{R}_1$ opens to 0, and $\mathbf{R}_2$ opens to 1.
- Hybrid_1: Adversary given pp as a LWE matrix $\mathbf{A} = \begin{bmatrix} \bar{\mathbf{A}} \\ \bar{\mathbf{s}}^T \bar{\mathbf{A}} + \mathbf{e}^T \end{bmatrix}$ and output the same thing as in Hybrid_0.

Under the LWE assumption, Hybrid_0 and Hybrid_1 are indistinguishable. We know Hybrid_1 is statistically binding, which means it is extractable. The chance of adversary to win in Hybrid_1 is negl no matter how powerful the adversary is. So the advantage of adversary in Hybrid_0 is still negl. Therefore, even though scheme 2 above has a equivocation property, it is possible to open any $\mathbf{C}$ to 2 different messages, a computationally bounded adversary cannot find these 2 values.

Similar arguments can show that the first scheme above is statistically binding and computationally hiding.

There are 2 distributions to sample the public parameters. Each gives either statistically binding or statistically hiding, and the 2 distributions are computationally indistinguishable. If we sample from distribution $\mathcal{D}_1$ we will get statistical property that $\mathcal{D}_1$ provides and computational property that the other distribution $\mathcal{D}_2$ provides.

A dual mode commitment scheme is useful because we can argue statistical property easily and then argue computational property by switching between the 2 strings in the hybrids.

14.3 An Application: Designated-Prover NIZKs for NP

Construction 14.2. Designated-prover NIZK

- Setup: Output a pair of (vk, sk). The sk will be given to the prover, who also has a statement (s) and witness (w). The vk will be given to the verifier, who knows the statement s. The prover wants to convince the verifier that it has a witness w follows the relation associated with the NP language $\mathcal{L}$by sending a proof $\diamond$. The scheme is not interactive because the only communication is prover sending a message to the verifier.
- Completeness: if $\mathsf{x} \in \mathcal{L}$, then verifier(vk, x, Π) = 1.
- Soundness: if $\mathsf{x} \notin \mathcal{L}$, then Pr[verifier(vk, x, Π^*) = 1] = negl, for all Π^*.
- Zero-Knowledge: The same proving key can be reused to generate multiple proofs and still preserve Zero-Knowledge. For every efficient adversary $\mathcal{A}$, there exists an efficient simulator $\mathcal{S} = (S_0, S_1)$ such that $(\mathsf{vk}, \mathsf{sk}) \leftarrow \mathsf{Setup}(1^\lambda)$, $(\tilde{\mathsf{vk}}, \tilde{\mathsf{st}}) \leftarrow S_0(1^\lambda)$, and the probability that $-\Pr[\mathcal{A}^{O_0(\mathsf{sk},\mathsf{x},\mathsf{w})}(\mathsf{vk}) = 1]$ - $\Pr[\mathcal{A}^{O_1(\tilde{\mathsf{st}},\mathsf{x},\mathsf{w})}(\tilde{\mathsf{vk}}) = 1]-$ = negl. $O_0(\mathsf{sk}, \mathsf{x}, \mathsf{w})$ outputs Prove(sk, x, w) if $\mathcal{R}$(x,w) = 1 and $\perp$ otherwise; and $O_1(\tilde{\mathsf{st}}, \mathsf{x}, \mathsf{w})$ outputs $S_1(\tilde{\mathsf{st}}, \mathsf{x})$ if $\mathcal{R}$(x,w) = 1 and $\perp$ otherwise.

  Zero-Knowledge says that an adversary who can see the real proof in O_0 can not tell the difference between real proof and simulated proof (O_1).

14.3.1 Construction Attempt 1

- pp: $\mathsf{Setup}(1^\lambda)$
- Prover knows statement (x) and witness (w). Verifier knows the statement (x). The public parameter pp will given to both prover and verifier.
- The prover will first commit to the witness as: σ_w : Commit(pp,w), and then compute on NP relation function $\mathcal{R}(x, w)$: $\sigma_{\mathcal{R}_x(w)}$: Eval(σ_w, $\mathcal{R}_x$). The prover creates the proof τ by open the commitment: Open(pp, $\sigma_{\mathcal{R}_x(w)}$, 1) (If the statement is true, should be open to 1). Prover sends the commited witness σ_w and proof τ to the verifier.
- The verifier will compute $\sigma_{\mathcal{R}_x(w)}$ upon receiving σ_w. The function $\mathcal{R}_x()$ depends on statement x only. Then check that $\sigma_{\mathcal{R}_x(w)}$ opens to 1.

The Attempt is zero-knowledge because σ_w is from a hiding commitment scheme so σ_w does not leak anything about the witness w. The opening τ is context hiding.

However this protocol is not sound because σ_w might not be generated honestly.

14.3.2 Construction Attempt 2: Move commitment to the public parameter

- pp: Setup(1^λ), σ_w: Commit(pp, w).
- Prove can compute commitment $\sigma_{\mathcal{R}_x(w)}$: Eval(σ_w, $\mathcal{R}_x$) and proof τ as Open(pp, $\sigma_{\mathcal{R}_x(w)}$, 1). Prover will only send τ to the verifier.
- The verifier will compute $\sigma_{\mathcal{R}_x(w)}$ upon receiving σ_w. The function $\mathcal{R}_x()$ depends on statement x only. Then check that $\sigma_{\mathcal{R}_x(w)}$ opens to 1.

We solve the above soundness problem because σ_w is part of public parameter so prover cannot cheat it. There does not exist a witness for a false statement. Zero-Knowledge is not affected because we still have context hiding and binding.

However the public parameter now depends on the witness. The prover cannot choose the statement and witness it wants to prove.

14.3.3 Solution: add a layer of indirection

- Setup: Sample a symmetric key k. Sample pp for commitment scheme. The secret proving key will be sk: (k, r), and the public verification key is pk: σ_k, and pp, where σ_k is a commitment to the secret key k and r is an opening to σ_k.
- Prover starts by encrypting the witness: ct: Encrypt(k, w). Create a circuit $\mathcal{C}$with ct, and x hard coded inside it: $\mathcal{C}_{\text{ct, x}}(k) : \mathcal{R}$, Decrypt(k, ct). Then use the commitment to the secret key σ_k to compute $\sigma_{\mathcal{C}_{ct,x}(k)}$. The proof τ will be the ct and opening to $\sigma_{\mathcal{C}_{ct,x}(k)}$. Decrypt(k, ct) will decrypt to the witness so (ct and opening to $\sigma_{\mathcal{C}_{ct,x}(k)}$) is a commitment to 1.
- Verifier Homomorphically evaluates $\mathcal{C}_{\text{ct, x}}()$ on σ_k and checks that if the opening to $\sigma_{\mathcal{C}_{ct,x}(k)}$ really opens to 1.

The witness is hiding because of the semantic security of the encryption scheme. (ct and opening to $\sigma_{\mathcal{C}_{ct,x}(k)}$) does not depend on the witness so can be simulated by a simulator so the scheme is Zero-Knowledge.

The Soundness holds because of binding and unforgeability property of commitment. If x $\notin \mathcal{L}$, then $\mathcal{C}_{\text{ct, x}}(k) = 0$ for all ciphertext.

CS 395T: Topics in Cryptography March 9, 2022

Lecture 15: Attribute-Based Encryption

Lecturer: David Wu *Scribe: George Lu*

15.1 Preliminaries

Definition 15.1. We say set of algorithms $(\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Encrypt}, \mathsf{Decrypt})$ is a correct attribute-based encryption scheme if for all messages μ, functions f and attributes x where $f(x) = 1$

$$\Pr\left[\mathsf{Decrypt}(\mathsf{sk}_f, \mathsf{ct}_{x,\mu}) = \mu \;\middle|\; \begin{array}{c}(\mathsf{pk}, \mathsf{msk}) \leftarrow \mathsf{Setup}(1^\lambda) \\ \mathsf{sk}_f \leftarrow \mathsf{KeyGen}(\mathsf{msk}, f) \\ \mathsf{ct}_{x,\mu} \leftarrow \mathsf{Encrypt}(\mathsf{pk}, x, \mu)\end{array}\right] = 1$$

Definition 15.2. We say set of algorithms $(\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Encrypt}, \mathsf{Decrypt})$ are a secure attribute-based encryption scheme for all admissible[1] PPT adversaries, the advantage in the following experiments (parameterized by bit b) is negligible.

- **Setup Phase:**
  - Challenger generates $(\mathsf{pk}, \mathsf{msk}) \leftarrow \mathsf{Setup}(1^\lambda)$
  - Challenger sends pk to adversary
- **Pre-Challenge Key Queries:** Adversary repeats as many times as it desires
  - Adversary sends a function f to Challenger
  - Challenger runs $\mathsf{sk}_f \leftarrow \mathsf{KeyGen}(\mathsf{msk}, f)$ and sends sk_f to adversary
- **Challenge Ciphertext:**
  - Adversary sends an attribute string x and challenge messages μ_0, μ_1
  - Challenger runs $\mathsf{ct}_{x,b} \leftarrow \mathsf{Encrypt}(\mathsf{pk}, x, \mu_b)$ and sends $\mathsf{ct}_{x,b}$ to adversary.
- **Post-Challenge Key Queries:** Adversary repeats as many times as it
  - Adversary sends a function f to Challenger
  - Challenger runs $\mathsf{sk}_f \leftarrow \mathsf{KeyGen}(\mathsf{msk}, f)$ and sends sk_f to adversary
- Adversary returns a bit b'

15.2 Construction

15.2.1 Dual-Regev Encryption

Our starting point is a plain public-key encryption scheme which can be viewed as a 'dual' of the normal Regev encryption PKE scheme covered previously, in the sense that the ciphertext structure of dual Regev mirrors the structure of the public key of primal Regev and vice versa.

- $\mathsf{KeyGen}(1^\lambda)$

[1] An adversary is considered admissible if $f(x) = 0$ for all key queries made by the adversary

- $A \leftarrow \mathbb{Z}_q^{n \times m}, r \leftarrow \{0,1\}^m, t = Ar$
- Output $(\mathsf{pk} = (A, t), \mathsf{sk} = r)$

- $\mathsf{Encrypt}(\mathsf{pk}, \mu)$
  - $s \leftarrow \mathbb{Z}_q^n, e \leftarrow \chi^m, e' \leftarrow \chi$
  - Output $\mathsf{ct} = \left(s^\top A + e^\top, s^\top t + e' + \mu \cdot \lfloor \frac{q}{2} \rfloor\right)$
- $\mathsf{Decrypt}(\mathsf{sk} = r, \mathsf{ct} = (\mathsf{ct}_0, \mathsf{ct}_1))$
  - Output $\lceil \mathsf{ct}_1 - \mathsf{ct}_0 r \rfloor_2$

Correctness

Lemma 15.3. *If* $\Pr\left[|\chi| \leq \frac{q}{4(m+1)}\right] = 1$*, then the above is a correct PKE scheme*

Proof. Observe we can write

$$\begin{aligned} &\lceil \mathsf{ct}_1 - \mathsf{ct}_0 r \rfloor_2 \\ &= \left\lceil s^\top t + e' + \mu \cdot \left\lfloor \frac{q}{2} \right\rfloor - \left(s^\top A + e^\top\right) \cdot r \right\rfloor_2 \\ &= \left\lceil s^\top t + e' + \mu \cdot \left\lfloor \frac{q}{2} \right\rfloor - s^\top A r - e^\top r \right\rfloor_2 \\ &= \left\lceil \mu \cdot \left\lfloor \frac{q}{2} \right\rfloor + e' - e^\top r \right\rfloor_2 \end{aligned}$$

Since we can bound e' and the entries of e with $\frac{q}{4(m+1)}$, we can bound $|e' - e^\top r|$ with $\frac{q}{4}$, so our rounding will output μ. □

Security

Lemma 15.4. *Assuming* $\mathsf{LWE}_{n,m+1,q,\chi}$ *and* $m \in \Omega(n \log q)$*, the above is a secure PKE scheme*

Proof. Consider the following sequence of hybrids:

- Hybrid 0
  - $A \leftarrow \mathbb{Z}_q^{n \times m}, r \leftarrow \{0,1\}^m, t = Ar$
  - $s \leftarrow \mathbb{Z}_q^n, e \leftarrow \chi^m, e' \leftarrow \chi$
  - Adversary receives $\left(s^\top A + e^\top, s^\top t + e' + \mu \cdot \lfloor \frac{q}{2} \rfloor\right)$
- Hybrid 1
  - $A \leftarrow \mathbb{Z}_q^{n \times m}, \mathbf{t} \leftarrow \mathbb{Z}_{\mathbf{q}}^{\mathbf{n}}$
  - $s \leftarrow \mathbb{Z}_q^n, e \leftarrow \chi^m, e' \leftarrow \chi$
  - Adversary receives $\left(s^\top A + e^\top, s^\top t + e' + \mu \cdot \lfloor \frac{q}{2} \rfloor\right)$
- Hybrid 2
  - Sample $\mathsf{ct}_0 \leftarrow \mathbb{Z}_q^m, \mathsf{ct}_1 \leftarrow \mathbb{Z}_q$
  - Adversary receives $(\mathsf{ct}_0, \mathsf{ct}_1)$

Hybrids 0 and 1 are indistinguishable via the leftover hash lemma, as $m \in \Omega(n \log q)$. Hybrids 1 and 2 are indistinguishable via LWE, with secret s and matrix $\left[\frac{A}{t}\right]$.

□

15.2.2 ABE Construction (Informal)

For notational convenience, we will say decryption succeeds iff $f(x) = 0$ (instead of $f(x) = 1$ as defined above.

- $\mathsf{Setup}(1^\lambda) \to (\mathsf{pk}, \mathsf{msk})$: Sample random matrices $B_1, \ldots B_\ell \in \mathbb{Z}_q^{n \times m}$, a random vector $t \leftarrow \mathbb{Z}_q^n$, and a trapdoor $(A, td_A) \leftarrow \mathsf{TrapGen}(n, m, q)$, outputting $(A, B_1, \ldots, B_\ell, t)$ as the public parameters and the lattice trapdoor td_A as the master secret key.
- $\mathsf{KeyGen}(\mathsf{msk}, f) \to \mathsf{sk}_f$: Sample a short vector z_f using td_A such that $[A|B_f]z_f = t$ to output as the function secret key.
- $\mathsf{Encrypt}(\mathsf{pk}, x, \mu)$: Sample vectors $e^\top, \tilde{e}^\top, e'$ from error distribution, and output $s^\top A + e^\top$, $s^\top[B_1 - x_1 G| \ldots |B_\ell - x_\ell \cdot G] + \tilde{e}^\top$, $s^\top t + e' + \mu \cdot \lfloor \frac{q}{2} \rfloor$
- $\mathsf{Decrypt}(\mathsf{sk}_f, \mathsf{ct})$: Recall $B_f - f(x) \cdot G = [B_1 - x_1 G| \ldots |B_\ell - x_\ell \cdot G] \cdot H_{f,x}$. Using the first 2 ciphertext components, when $f(x) = 0$, this allows us to compute $s^\top[A|B_f] + error$. Multiplying by a short z_f allows us to recover $s^\top t + error$ while keeping the error 'small', which allows us to recover μ via computing the difference with the third ciphertext component

CS 395T: Topics in Cryptography March 21, 2022

Lecture 16: Attribute-Based Encryption

Lecturer: David Wu *Scribe: Steven Cheng*

In this lecture, we will focus on constructing attribute-based encryption. Recall that in an attribute-based encryption scheme, a ciphertext ct is associated with a *public* attribute x, while decryption keys sk_f are bound to a function f. Decryption of ct using sk_f should only succeed if $f(x) = 0$, outputting $\perp$ otherwise. Note that this flips the standard convention, where successful decryption occurs when $f(x) = 1$.

16.1 ABE From Dual Regev Encryption

Construction 16.1 (Attribute Based Encryption)**.** . We can construct the following attribute based encryption scheme from dual Regev encryption:

- $\mathsf{Setup}(1^\lambda)$: Set $n = n(\lambda), q = q(\lambda), m = \Theta(n \log q)$ as the lattice parameters, $\chi = \chi(\lambda)$ as the error distribution, and $\sigma = \sigma(\lambda)$ as the parameter for preimage sampling. Next, we sample the following:

$$\begin{aligned} &(A, T) \leftarrow \mathsf{TrapGen}(n, q) && A \in \mathbb{Z}_q^{n \times m} \\ &B_1, \ldots, B_\ell \xleftarrow{R} \mathbb{Z}_q^{n \times t} && t = \lceil n \log q \rceil \\ &p \xleftarrow{R} \mathbb{Z}_q^n \end{aligned}$$

  Finally, we output $\mathsf{mpk} = (A, B_1, \ldots, B_\ell, p)$ and $\mathsf{msk} = T$.

- $\mathsf{KeyGen}(\mathsf{mpk}, \mathsf{msk}, f)$: Compute $B_f \leftarrow [B_1 | \ldots | B_\ell] \cdot H_f$, then use preimage sampling to obtain a vector $z \leftarrow \mathsf{SamplePre}\left([A|b_f], \begin{bmatrix} T \\ 0 \end{bmatrix}, p, \sigma\right)$. Output $\mathsf{sk}_f \leftarrow z$.

- $\mathsf{Encrypt}(\mathsf{mpk}, x, \mu)$: Sample the following:

$$\begin{aligned} &s \xleftarrow{R} \mathbb{Z}_q^n, e_1 \leftarrow \chi^m, e' \leftarrow \chi \\ &R_1, \ldots, R_\ell \xleftarrow{R} \{0,1\}^{m \times t} \\ &e_2 \leftarrow e_1^T [R_1 | \ldots | R_\ell] \end{aligned}$$

  Output $\mathsf{ct} = \left(s^T A + e_1^T, \;\; s^T [B_1 - x_1 G | \ldots | B_\ell - x_\ell G] + e_2^T, \;\; s^T p + e' + \mu \left\lfloor \frac{q}{2} \right\rceil, x\right)$.

- $\mathsf{Decrypt}(\mathsf{sk}_f, \mathsf{ct})$: Compute $\mathsf{ct}_3 - [\mathsf{ct}_1 | \mathsf{ct}_2 H_{f,x}] z$ and round.

Correctness. Suppose $f(x) = 0$. Observe that

$$\mathsf{ct}_2 H_{f,x} = \left(s^T [B_1 - x_1 G | \ldots | B_\ell - x_\ell G] + e_2^T]\right) H_{f,x} = s^T \left(B_f - f(x) \cdot G\right) + e_2^T H_{f,x} = s^T B_f + e_2^T H_{f,x}$$

Therefore,

$$[\mathsf{ct}_1 | \mathsf{ct}_2 H_{f,x}] z = \left(s^T [A | B_f] + [e_1^T | e_2^T H_{f,x}]\right) = s^T p + [e_1^T | e_2^T H_{f,x}] z$$

The decryption algorithm computes

$$\mathsf{ct}_3 - [\mathsf{ct}_1 | \mathsf{ct}_2 H_{f,x}] z = \left(s^T p + e' + \mu \left\lfloor \frac{q}{2} \right\rceil\right) - \left(s^T p + [e_1^T | e_2^T H_{f,x}] z\right) = \mu \cdot \left\lfloor \frac{q}{2} \right\rceil + e' - [e_1^T | e_2^T H_{f,x}] z$$

Observe that $e' - [e_1^T | e_2^T H_{f,x}] z$ is small since e', e_1, e_2 are all sampled from the error distribution and $||H_{f,x}|| < m^{O(d)}$, where d is the depth of the computation. Thus after rounding, we are simply left with $\mu \left\lfloor \frac{q}{2} \right\rceil$, from which we can easily extract μ.

Security. Proving security for this scheme is delicate. For the reduction to LWE, we need to simulate the ABE scheme, which in particular requires simulation of KeyGen for functions f such that $f(x) = 1$. Simulating this naively requires msk, the trapdoor for A, but access to this trapdoor breaks our LWE assumption!

In the reduction, it should be the case that by breaking ABE, the adversary provides meaningful insight that allows us to break LWE. To remedy this issue, we will shortly introduce a technique known as a *punctured trapdoor*, which allows us to properly produce secret keys only when $f(x) = 1$. Importantly, we are unable to produce keys for when $f(x) = 0$, and it is through this insight where the adversary provides meaningful insight.

In order to properly use a punctured trapdoor, we use the more relaxed notion for security, known as selective security. In this variation, the adversary must first declare the attribute before the public parameters are sent. It is still an open problem to construct an adaptively secure ABE scheme directly from LWE.

Selective security game. The game for selective semantic security works as follows:

1. The adversary sends the attribute, x, to the challenger.
2. The challenger generates $\mathsf{mpk}, \mathsf{msk} \leftarrow \mathsf{Setup}(1^\lambda)$ sends the public key mpk to the adversary.
3. The adversary sends a function f, and the challenger responds with $\mathsf{sk}_f \leftarrow \mathsf{KeyGen}(\mathsf{msk}, f)$. This step repeats as many times as the adversary likes.
4. The adversary sends μ_0, μ_1, and the challenger responds with $\mathsf{ct} \leftarrow \mathsf{Encrypt}(\mathsf{mpk}, x, \mu_b)$.
5. The adversary sends more functions f, and the challenger responds with $\mathsf{sk}_f \leftarrow \mathsf{KeyGen}(\mathsf{msk}, f)$. Again, this step repeats.
6. The adversary outputs a guess b' for which message was encrypted.

Additionally, we require that for all f that the adversary sends, $f(x) \neq 0$. Otherwise, the adversary could just decrypt the ciphertext using one of the provided secret keys.

Remark 16.2. The adversary is allowed to ask for secret keys twice – once before sending the message, and once after – for two different reasons. An adversary may want to choose their messages based on what the secret keys look like, and may want to ask for secret keys based on what the ciphertext looks like, so both steps are necessary to encapsulate security.

Also, observe that Setup is run *after* the attribute x is provided. As we will see shortly, this will allow us to use punctured trapdoors that embeds x into mpk.

Proving security. We will use a hybrid argument to prove selective security of our ABE scheme. To start, Hyb_0 will be the standard selective semantic security game described above.

For Hyb_1, we modify how B_i is computed in Hyb_0, as follows:

- In Step 2, instead of $\mathsf{mpk}, \mathsf{msk} \leftarrow \mathsf{Setup}(1^\lambda)$, the challenger computes

$$\begin{aligned}
&(A, T) \leftarrow \mathsf{TrapGen}(n, q) \\
&R_1, \ldots, R_\ell \xleftarrow{R} \{0,1\}^{m \times t} \\
&B_1 \xleftarrow{R} AR_1 + x_1 G, \ldots, B_\ell \xleftarrow{R} AR_\ell + x_\ell G \\
&p \xleftarrow{R} \mathbb{Z}_q^n \qquad \mathsf{mpk} \xleftarrow{R} (A, B_1, \ldots, B_\ell, p) \qquad \mathsf{msk} \leftarrow T
\end{aligned}$$

- To compute the challenge ciphertext, the challenger computes:

$$\begin{aligned}
&s \xleftarrow{R} \mathbb{Z}_q^n, e_1 \xleftarrow{R} \chi^m, e' \xleftarrow{R} \chi, e_2^T = e_1^T [R_1 | \ldots | R_\ell] \\
&\mathsf{ct} \xleftarrow{R} \left(s^T A + e_1^T, s^T [B_1 - x_1 G | \ldots | B_\ell - x_\ell G] + e_2^T, s^T p + e' + \mu \cdot \left\lfloor \frac{q}{2} \right\rceil, x \right)
\end{aligned}$$

Hyb_0 and Hyb_1 are statistically indistinguishable by a variant of LHL. In particular, these hybrids are indistinguishable if $(A, AR, e^T R) \overset{s}{\approx} (A, u, e^T R)$, which follows from LHL.

In Hyb_2, we compute KeyGen without using the trapdoor for A:

$$R_{f,x} \leftarrow [R_1|\dots|R_\ell]H_{f,x}$$
$$z_f \xleftarrow{R} \mathsf{SamplePre}\left([A|B_f]\begin{bmatrix}-R_{f,x}\\ I\end{bmatrix}, t, \sigma\right)$$

When σ is sufficiently large enough ($\sim m^{O(d)}$), then Hyb_1 and Hyb_2 are statistically indistinguishable by pre-image sampling. It is sufficient to show that $\begin{bmatrix}-R_{f,x}\\ I\end{bmatrix}$ is a short trapdoor for $[A|B_f]$. By homomorphic evaluation,

$$[B_1 - x_1 G|\dots|B_\ell - x_\ell G]H_{f,x} = B_f - f(x)G$$

When we restrict $f(x) = 1$, then the product above equals $B_f - G$. But since $B_i = AR_i + x_i G$, we have

$$[B_1 - x_1 G|\dots|B_\ell - x_\ell G]H_{f,x} = [AR_1|\dots|AR_\ell]H_{f,x} = AR_{f,x} = B - G$$

Thus, we see that $\begin{bmatrix}-R_{f,x}\\ I\end{bmatrix}$ is indeed a trapdoor:

$$[A|B_f]\begin{bmatrix}-R_{f,x}\\ I\end{bmatrix} = -AR_{f,x} + B_f = G$$

Moreover, $||R_{f,x}|| \le m^{O(d)}$, so $\begin{bmatrix}-R_{f,x}\\ I\end{bmatrix}$ is short as well.

Punctured trapdoors. Observe that the new KeyGen scheme does not use the trapdoor for A, but can only be used to generate keys if $f(x) = 1$. This technique of programming x into the public key to create a trapdoor only when $f(x) = 1$ is referred to as a punctured trapdoor.

Finally, in Hyb_3, we replace the ciphertext ct with $(z_1^T, z_1^T[R_1|\dots|R_\ell], z')$, where $z_1 \xleftarrow{R} \mathbb{Z}_q^m, z' \xleftarrow{R} \mathbb{Z}_q$.

Hyb_2 and Hyb_3 are computationally indistinguishable under LWE. To see this, let $([A|p], [z_1^T|z'])$ be the LWE challenge. Observe that we can use this A to compute the public key for $\mathsf{Hyb}_2/\mathsf{Hyb}_3$:

$$R_1, \dots, R_\ell \xleftarrow{R} \{0,1\}^{m\times t} \qquad p \xleftarrow{R} \mathbb{Z}_q^n \qquad B_i \leftarrow AR_i + X_i G \qquad \mathsf{mpk} \leftarrow (A, B_1, \dots, B_\ell, p)$$

Once we have mpk, we can generate secret keys using a punctured trapdoor. Finally, to simulate ciphertexts, we compute $\mathsf{ct} \leftarrow \left(z_1^T, z_1^T[R_1|\dots|R_\ell], z' + \mu_0 \cdot \left\lfloor \frac{q}{2} \right\rceil\right)$.

When $z_1^T = s^T A + e_1^T$ and $z' = s^T p + e'$, then we get the distribution for Hyb_2:

$$\begin{aligned} z_1^T[R_1|\dots|R_\ell] &= \left[s^T AR_1 + e^T R_1|\dots|s^T AR_\ell + e_1^T R_\ell\right] \\ &= s^T[AR_1|\dots|AR_\ell] + e_1^T[R_1|\dots|R_\ell] \\ &= s^T[B_1 - x_1 G|\dots|B_\ell - x_\ell G] + e_2^T \\ z' + \mu_0 \cdot \left\lfloor \frac{q}{2} \right\rceil &= s^T p + e' + \mu_0 \cdot \left\lfloor \frac{q}{2} \right\rceil \end{aligned}$$

And when z_1, z' are uniform random, then the ciphertext matches the distribution of Hyb_3.

CS 395T: Topics in Cryptography March 23, 2022

Lecture 17: Predicate Encryption

Lecturer: David Wu *Scribe: Rachit Garg*

In the last lecture, we saw how to construct an Attribute Based Encryption scheme from the LWE assumption. Our key proof component was programming x^* into the public key. This yields a trapdoor for $[\mathbf{A}|\mathbf{B}_f]$ whenever $f(x^*) = 1$, and ensures semantic security whenever $f(x^*) = 0$. Additionally recall that our construction contained the attribute x in the clear in the ciphertext. This allowed the decryption algorithm to compute the matrix $\mathbf{H}_{f,x}$ and hence compute the decryption correctly.

In this lecture, we will see how to construct Predicate Encryption scheme. Ciphertexts in such a scheme additionally *hide* the attribute. Attribute hiding can come in two flavors -

- Weak Attribute Hiding: Successful decryption also recovers attribute.
- Strong Attribute Hiding: Attribute remains hidden even if decryption succeeds (implies functional encryption).

Our focus will be on *weak* attribute hiding.

High level idea. Combine FHE with ABE. We will encrypt the attribute under ABE and homomorphically evaluate the predicate. However, if we simply encrypt the attribute, we need to ensure that the decryptor will be able to decrypt at some point and compute the underlying message. To do this, we will use a "dual-use" technique where the underlying schemes share a *common* secret key.

First, we will generalize our homomorphic evaluation relations to support matrix-valued computations. So far, we have the following equations, a function $f : \{0,1\}^\ell \to \{0,1\}$:

$$[\mathbf{B}_1|\dots|\mathbf{B}_\ell] \cdot \mathbf{H}_f = \mathbf{B}_f \qquad \text{Input-Independent Evaluation} \qquad (17.1)$$

$$[\mathbf{B}_1 - x_1\mathbf{G}|\dots|\mathbf{B}_\ell - x_\ell\mathbf{G}] \cdot \mathbf{H}_{f,x} = \mathbf{B}_f - f(x)\mathbf{G}. \qquad \text{Input-Dependent Evaluation} \qquad (17.2)$$

Suppose that $f : \{0,1\}^\ell \to \mathbb{Z}_q^{n\times m}$ is a matrix-valued function. Then we will describe an analogous relation:

$$[\mathbf{B}_1|\dots|\mathbf{B}_\ell] \cdot \mathbf{H}_f = \mathbf{B}_f \qquad \text{Input-Independent Evaluation} \qquad (17.3)$$

$$[\mathbf{B}_1 - x_1\mathbf{G}|\dots|\mathbf{B}_\ell - x_\ell\mathbf{G}] \cdot \mathbf{H}_{f,x} = \mathbf{B}_f - f(x). \qquad \text{Input-Dependent Evaluation} \qquad (17.4)$$

We will use a bit by bit approach. Let $f_{j,k} : \{0,1\}^\ell \to \{0,1\}$ be function that computes k^{th} bit of j^{th} entry of $f(x)$. Then,

$$\begin{aligned}[\mathbf{B}_1 - x_1\mathbf{G}|\dots|\mathbf{B}_\ell - x_\ell\mathbf{G}]\mathbf{H}_{f_{j,k},x} &= \mathbf{B}_{f_{j,k}} - [f(x)]_{j,k}\mathbf{G} \\ &= [\mathbf{B}_1|\dots|\mathbf{B}_\ell]\mathbf{H}_{f_{j,k}} - [f(x)]_{j,k}\mathbf{G}.\end{aligned}$$

Let $\mathbf{E}_j \in \mathbb{Z}_q^{n\times m}$ be the matrix that is 1 in position j (where j ranges over all $n \cdot m$ indices). Then we can write, $f(x) = \sum_{j\in[n\cdot m]} \sum_{k\in[\lceil \log q\rceil]} [f(x)]_{j,k} \cdot 2^k \mathbf{E}_j$. Using this expression, we can consider multiplying our matrix expression by $\mathbf{G}^{-1}(2^k\mathbf{E}_j)$ and getting,

$$\begin{aligned}\sum_{j,k}[\mathbf{B}_1 - x_1\mathbf{G}|\dots|\mathbf{B}_\ell - x_\ell\mathbf{G}] \cdot \mathbf{H}_{f_{j,k},x} \cdot \mathbf{G}^{-1}(2^k\mathbf{E}_j) &= \sum_{j,k}[\mathbf{B}_1|\dots|\mathbf{B}_\ell] \cdot \mathbf{H}_{f_{j,k}} \cdot \mathbf{G}^{-1}(2^k\mathbf{E}_j) - \sum_{j,k}[f(x)]_{j,k} \cdot \mathbf{G}\mathbf{G}^{-1}(2^k\mathbf{E}_j) \\ &= [\mathbf{B}_1|\dots|\mathbf{B}_\ell]\sum_{j,k}\mathbf{H}_{f_{j,k}} \cdot \mathbf{G}^{-1}(2^k\mathbf{E}_j) - f(x)\end{aligned}$$

Thus our final expression sets,

$$\mathbf{H}_f = \sum_{j,k}\mathbf{H}_{f_{j,k}} \cdot \mathbf{G}^{-1}(2^k\mathbf{E}_j)$$

$$\mathbf{H}_{f,x} = \sum_{j,k}\mathbf{H}_{f_{j,k},x} \cdot \mathbf{G}^{-1}(2^k\mathbf{E}_j)$$

Then we have our generalized equations 17.3,17.4 where $\|\mathbf{H}_f\|, \|\mathbf{H}_{f,x}\| \leq (n \log q)^{O(d)}$.

17.1 Predicate Encryption from LWE

Construction 17.1 (Predicate Encryption from LWE)**.** We construct the predicate encryption as follows:

- $\mathsf{Setup}(1^\lambda, 1^\ell)$: Set $(\mathbf{A}, \mathsf{td}) \leftarrow \mathsf{TrapGen}(n, q)$. Sample $\mathbf{B}_1 \ldots, \mathbf{B}_L \xleftarrow{R} \mathbb{Z}_q^{n \times m}, \mathbf{p} \xleftarrow{R} \mathbb{Z}_q^n$ where $L = \mathsf{poly}(\ell, n, \log q)$[1]. Output $\mathsf{mpk} = (\mathbf{A}, \mathbf{B}_1, \ldots, \mathbf{B}_L, \mathbf{p})$ and $\mathsf{msk} = \mathsf{td}$.
- $\mathsf{Encrypt}(\mathsf{mpk}, x, \mu)$: Sample $\mathbf{s} \xleftarrow{R} \mathbb{Z}_q^n$, $\mathbf{e} \leftarrow \chi^m$. Sample $\mathbf{R}_i \xleftarrow{R} \{0,1\}^{(n+1)\lceil \log q \rceil \times (n+1)\lceil \log q \rceil}$ and compute $\forall i \in [\ell]$, the GSW ciphertext, $\mathbf{T}_i = \begin{bmatrix} \mathbf{A} \\ \mathbf{s}^\mathsf{T}\mathbf{A} + \mathbf{e}^\mathsf{T} \end{bmatrix} \mathbf{R}_i + x_i \cdot \mathbf{G}$.

  Let $\mathsf{t}_1, \ldots, \mathsf{t}_L$ be the binary representaion of $\mathbf{T} = [\mathbf{T}_1 | \ldots | \mathbf{T}_\ell]$. We finally encode the bits of $\mathsf{t}_1, \ldots, \mathsf{t}_L$ and output, $\mathbf{c}_0^\mathsf{T} \leftarrow \mathbf{s}^\mathsf{T}\mathbf{A} + \mathbf{e}_0^\mathsf{T}$ where $\mathbf{e}_0 \leftarrow \chi^m$ and $\forall j \in [L], \mathbf{c}_j^\mathsf{T} = \mathbf{s}^\mathsf{T}[\mathbf{B}_j - \mathsf{t}_j \cdot \bar{\mathbf{G}}] + \mathbf{e}_j^\mathsf{T}$ where $\mathbf{e}_j \leftarrow \chi^m$. Note that $\bar{\mathbf{G}}$ is the gadget matrix *without* the last row, i.e. $\bar{\mathbf{G}} \in \mathbb{Z}_q^{n \times (n+1)\lceil \log q \rceil} = [\mathbf{G} \in \mathbb{Z}_q^{n \times n \lceil \log q \rceil} | \mathbf{0} \in 0^{n \times \lceil \log q \rceil}]$ and

$$\mathbf{G} = \begin{bmatrix} 1 & 2 & \ldots & 2^{\lceil \log q \rceil - 1} & & & & \\ & & & & \ldots & & & \\ & & & & & 1 & 2 & \ldots & 2^{\lceil \log q \rceil - 1} \end{bmatrix} = \mathbf{g}^\mathsf{T} \otimes \mathbf{I}_n$$

  is the gadget matrix. Finally, we compute $\mathsf{c}' = \mathbf{s}^\mathsf{T}\mathbf{p} + \mu \cdot \lfloor \frac{q}{2} \rceil + e'$ where $e' \leftarrow \chi$ and output $\mathsf{ct} = (\mathbf{T}, \mathbf{c}_0, \mathbf{c}_1, \ldots, \mathbf{c}_L, \mathsf{c}')$.
- $\mathsf{KeyGen}(\mathsf{msk} = \mathsf{td}, f)$: Let $\mathbf{T} = [\mathbf{T}_1 | \ldots | \mathbf{T}_\ell]$ be an encryption of $x = (x_1, \ldots, x_\ell)$. Let $\mathbf{T}_f = \mathsf{FHE.Eval}(f, \mathbf{T})$. Let $\hat{f}$ be a circuit that maps an input $\mathbf{T}$ to $\bar{\mathbf{T}}_f$ where $\mathbf{T}_f = \begin{bmatrix} \bar{\mathbf{T}}_f \\ \underline{\mathbf{t}}_f \end{bmatrix}$, i.e. the circuit homomorphically evaluates f on $\mathbf{T}$ and outputs all but the last row of $\mathbf{T}_f$.

  Use $\mathbf{B}_1, \ldots, \mathbf{B}_L, \hat{f}$ to compute $\mathbf{B}_{\hat{f}}$ from the input independent evaluation, i.e. compute $\mathbf{B}_{\hat{f}} = [\mathbf{B}_1 | \ldots | \mathbf{B}_L] \cdot \mathbf{H}_{\hat{f}}$. Use td to sample a short $\mathbf{z}_f$ such that $[\mathbf{A} | \mathbf{B}_{\hat{f}}]\mathbf{z}_f = \mathbf{p}$. Output the secret key sk_f as $\mathbf{z}_f$.
- $\mathsf{Decrypt}((\mathsf{sk}_f, f), \mathsf{ct})$: Homomorphically evaluate $\hat{f}$ on the encoding $[\mathbf{c}_1 | \ldots | \mathbf{c}_L]$ by computing the input dependent evaluation, i.e. $\mathbf{c}_{\hat{f}} \leftarrow [\mathbf{c}_1 | \ldots | \mathbf{c}_L] \cdot \mathbf{H}_{\hat{f}, \mathbf{T}}$. Homomorphically compute $\mathbf{T}_f \leftarrow \mathsf{FHE.Eval}(f, \mathbf{T})$ and let $\underline{\mathbf{t}}_f$ be the last row of $\mathbf{T}_f$.

  Compute $\mathsf{c}' - [\mathbf{c}_0 | \mathbf{c}_{\hat{f}} + \underline{\mathbf{t}}_f] \cdot \mathbf{z}_f$ and round the result.

Correctness. By construction,

$$\mathbf{c}_{\hat{f}} = [\mathbf{c}_1 | \ldots | \mathbf{c}_L] \cdot \mathbf{H}_{\hat{f}, \mathbf{T}} = \mathbf{s}^\mathsf{T}[\mathbf{B}_1 - \mathsf{t}_1 \bar{\mathbf{G}} | \ldots | \mathbf{B}_L - \mathsf{t}_L \bar{\mathbf{G}}]\mathbf{H}_{\hat{f}, \mathbf{T}} + [\mathbf{e}_1^\mathsf{T} | \ldots | \mathbf{e}_L^\mathsf{T}]\mathbf{H}_{\hat{f}, \mathbf{T}}.$$

Thus $\mathbf{c}_{\hat{f}} \approx \mathbf{s}^\mathsf{T}(\mathbf{B}_{\hat{f}} - \bar{\mathbf{T}}_f)$ (the error terms are small and captured in the approx notation). Finally, we have,

$$\begin{aligned} \mathbf{c}_{\hat{f}} + \underline{\mathbf{t}}_f &\approx \mathbf{s}^\mathsf{T}\mathbf{B}_{\hat{f}} - \mathbf{s}^\mathsf{T}\bar{\mathbf{T}}_f + \underline{\mathbf{t}}_f \\ &= \mathbf{s}^\mathsf{T}\mathbf{B}_{\hat{f}} + [-\mathbf{s}^\mathsf{T} | 1]\mathbf{T}_f \\ &\approx \mathbf{s}^\mathsf{T}\mathbf{B}_{\hat{f}} + f(x) \cdot [-\mathbf{s}^\mathsf{T} | 1]\mathbf{G} \end{aligned}$$

The last expression follows from GSW decryption. Recall that $\mathbf{T}_f$ is a GSW ciphertext encrypting $f(x)$ under the same secret $\mathbf{s}$. Note that when $f(x) = 0$, then $\mathbf{c}_{\hat{f}} + \underline{\mathbf{t}}_f \approx \mathbf{s}^\mathsf{T}\mathbf{B}_{\hat{f}}$ and $[\mathbf{c}_0 | \mathbf{c}_{\hat{f}} + \underline{\mathbf{t}}_f]\mathbf{z}_f \approx \mathbf{s}^\mathsf{T}[\mathbf{A} | \mathbf{B}_{\hat{f}}]\mathbf{z}_f = \mathbf{s}^\mathsf{T}\mathbf{p}$. Thus, $\mathsf{c}' - [\mathbf{c}_0 | \mathbf{c}_{\hat{f}} + \underline{\mathbf{t}}_f] \cdot \mathbf{z}_f \approx \mu \lfloor \frac{q}{2} \rceil$ and decryption succeeds.

[1] The exact polynomial will become clear based on our construction.

Security. Security follows by a similar argument as in ABE security (embed encryption of x^* into public parameters). Summarizing, the key idea behind our approach was using ABE evaluation (for matrix-valued relations), to compute,

$$\mathbf{s}^\mathsf{T}[\mathbf{B}_1 - x_1\mathbf{G}|\ldots|\mathbf{B}_\ell - x_\ell\mathbf{G}]\mathbf{H}_{f,x} = \mathbf{s}^\mathsf{T}(\mathbf{B}_f - f(x)\cdot\mathbf{G})$$

But this requires knowledge of x (to construct $\mathbf{H}_{f,x}$). To hide the attribute x, we encrypt x and homomorphically evaluate the FHE evaluation function, i.e. we compute $\mathsf{Encrypt}(f(x))$ from $\mathsf{Encrypt}(x)$. Now if $\mathbf{s}$ is also the GSW secret key, then $\mathbf{s}^\mathsf{T}$ efficiently implements GSW decryption. Using the same $\mathbf{s}$ for GSW and ABE is captured by this "dual use" approach.

CS 395T: Topics in Cryptography March 28, 2022

Lecture 18: Functional Encryption

Lecturer: David Wu *Scribe: Jiahui Liu*

18.1 Definition

A functional encryption scheme FE consists of the following algorithms:

- $\mathsf{Setup}(1^\lambda) \to (\mathsf{mpk}, \mathsf{msk})$: takes in a security parameter and outputs a master public key, a master secret key key pair $(\mathsf{mpk}, \mathsf{msk})$.
- $\mathsf{Encrypt}(\mathsf{mpk}, x) \to \mathsf{ct}_x$: takes in master public key and ciphertext x; outputs a ciphertext ct_x.
- $\mathsf{KeyGen}(\mathsf{msk}, f) \to \mathsf{sk}_f$: takes in the master secret key, a function f and outputs a functional key sk_f.
- $\mathsf{Decrypt}(\mathsf{sk}_f, \mathsf{ct}_x) \to f(x)$: takes in functional key sk_f and a ciphertext ct_x; outputs $f(x)$.

A functional encryption scheme should satisfy the following properties:

Correctness.

$$\Pr\left[\mathsf{Decrypt}(\mathsf{sk}_f, \mathsf{ct}_x) = f(x) \;\middle|\; \begin{array}{c}(\mathsf{mpk}, \mathsf{msk}) \leftarrow \mathsf{Setup}(1^\lambda),\\ \mathsf{ct}_x \leftarrow \mathsf{Encrypt}(\mathsf{mpk}, x),\\ \mathsf{sk}_f \leftarrow \mathsf{KeyGen}(\mathsf{msk}, f)\end{array}\right] \geq 1$$

Game-based security. There are different functional encryption security definitions. We will look at the game-based one.

We describe the following security game:

1. The challenger runs Setup to obtain $(\mathsf{mpk}, \mathsf{msk})$ and sends mpk to adversary $\mathcal{A}$.
2. $\mathcal{A}$ queries on functionality f of its own choice for polynomially many times; the challenger computes $\mathsf{sk}_f \leftarrow \mathsf{KeyGen}(\mathsf{msk}, f)$ and gives sk_f to $\mathcal{A}$.
3. $\mathcal{A}$ then provides x_0, x_1, where $f(x_0) = f(x_1)$ for all queried f as challenge messages. Challenger encrypts $\mathsf{ct}_b \leftarrow \mathsf{Encrypt}(\mathsf{mpk}, x_b)$ for a random b.
4. $\mathcal{A}$ queries again for sk_f on functionality f of its own choice for polynomially many times, where $f(x_0) = f(x_1)$ for all queried f.
5. $\mathcal{A}$ outputs a guess b'.

For any PPT adversary $\mathcal{A}$, there exists a negligible function $\mathsf{negl}(\cdot)$ such that for all λ, the followig holds:

$$|\Pr[b' = 1 | b = 0] - \Pr[b' = 1 | b = 1]| \leq \mathsf{negl}(\lambda)$$

.

Subtlety of FE definition. The definition above, when considering certain functionalities, can be satisfied by a trivial scheme: for instance, if the function is a OWP, then it's computatinally hard for the adversary to find $x_0 \neq x_1$ but $f(x_0) = f(x_1)$ for a start. But this definition is good enough for a broad range of settings.

FE is a very generic definition of encryption, PKE, ABE and predicate encryption can all be seen as FE with certain functional key.

18.2 Building block for FE: garbled circuits.

We next give a construction for **single-key** FE, where the adversary can only see a single functional key.

We will first introduce the main building block: Garbled circuits.

Garbled circuits can be used to realize two-party computation: Alice with input x and Bob with input y want to compute $f(x, y)$ without revealing their own input to each other.

We show a garbling protocol from [Yao86].

Garbling a single gate. The garbler (encoder) does the following:

- Consider an AND gate:

x_1	x_2	AND(x_1, x_2)
0	0	0
0	1	0
1	0	0
1	1	1

- For each wire i in the circuit: associate a pair of keys k_i^0, k_i^1. k_i^b is the key (of a symmetric encryption scheme) associated with wire i for wire value $b \in \{0, 1\}$. See Figure 18.1.

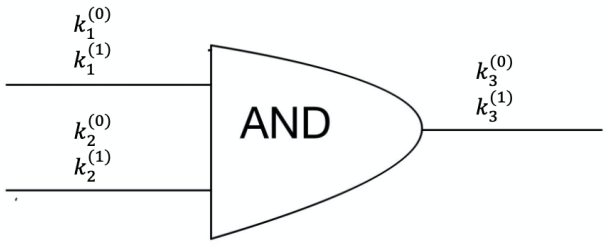


Figure 18.1: AND Gate with wire labels

- Prepare garbled truth table for each gate:
  1. Replace each entry of the truth table with corresponding key
  2. Encrypt output key with each of the input keys:

$$\begin{aligned}
\mathsf{ct}_{00} &\leftarrow \mathsf{Encrypt}(k_1^0, \mathsf{Encrypt}(k_2^0, k_3^0)) \\
\mathsf{ct}_{01} &\leftarrow \mathsf{Encrypt}(k_1^0, \mathsf{Encrypt}(k_2^1, k_3^0)) \\
\mathsf{ct}_{10} &\leftarrow \mathsf{Encrypt}(k_1^1, \mathsf{Encrypt}(k_2^0, k_3^0)) \\
\mathsf{ct}_{11} &\leftarrow \mathsf{Encrypt}(k_1^1, \mathsf{Encrypt}(k_2^1, k_3^1))
\end{aligned}$$

Then randomly shuffle positions of the ciphertexts.

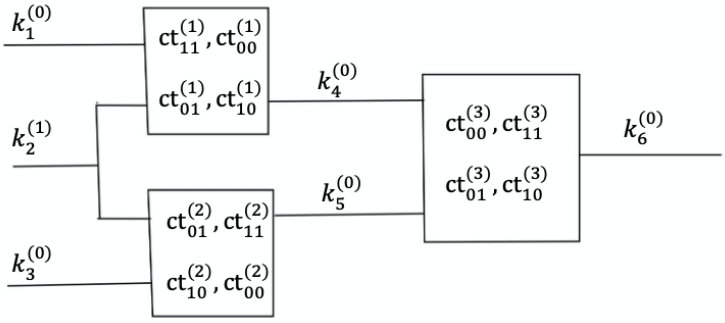
x_1	x_2	AND(x_1, x_2)
k_1^0	k_2^0	k_3^0
k_1^0	k_2^1	k_3^0
k_1^1	k_2^0	k_3^0
k_1^1	k_2^1	k_3^1

- Construct decoding table for output values:

$$k_3^0 \to 0; k_3^1 \to 1.$$

Alternatively, we can just encrypt output values instead of keys for output wires.

Garbling circuits. construct garbled table for each gate in the circuit, prepare decoding table for each output wire in the circuit.



Evaluating a garbled circuit. At each garbled gate of the circuit, the evaluator tries to decrypt each pf the 4 ciphertexts with the input keys and take the output key to be the ciphertext that decrypts. Then decode using the table.

The invariant in the evaluation: given keys for input wires of a gate, can derive the key corresponding to the output wire. This enables gate-by-gate evaluation of garbled circuit.

Note the security requirement here is: evaluator needs to obtain keys for its inputs only, but without revealing which keys it requests.

Garbling abstractions. The abstraction of a GC scheme is:

- $\mathsf{Garble}(1^\lambda, C) \to (\tilde{C}, \{L_i^b\}_{i\in[n],b\in\{0,1\}})$ where n is the number of input wires.
- $\mathsf{Eval}(\tilde{C}, \{L_i^b\}_{i\in[n]}) \to y$

A GC scheme should satisfy the following:

- **Correctness**: For all circuits $C : \{0,1\}^n \to \{0,1\}^m$ and all $x \in \{0,1\}^n$, for $(\tilde{C}, \{L_i^b\}_{i\in[n],b\in\{0,1\}}) \leftarrow \mathsf{Garble}(1^\lambda, C)$:

$$\Pr[\mathsf{Eval}(\tilde{C}, \{L_i^b\}_{i\in[n]}) = C(x)] = 1$$

- **Security**: There exists an efficient simulator S such that for all circuits $C : \{0,1\}^n \to \{0,1\}^m$ and all $x \in \{0,1\}^n$, for $(\tilde{C}, \{L_i^b\}_{i\in[n],b\in\{0,1\}}) \leftarrow \mathsf{Garble}(1^\lambda, C)$:

$$\{\tilde{C}, \{L_i^b\}_{i\in[n]}\} \approx_{\text{comp}} S(1^\lambda, C, C(x))$$

(We can also consider a notion where only $|C|$ is provided to S).

$\Rightarrow$ that is, the garbled circuit and *one set* of labels(keys) can be simulated just given the output value $C(x)$.

18.3 Using Garbled Circuits for Two-Party Computation

Oblivious transfer (OT). In an OT protocol, the sender has two messages (m_0, m_1); the receiver has a bit $b \in \{0,1\}$. At the end of the protocol, the receiver receives m_b and should learn nothing about m_{1-b}; the sender leanrs nothing about b. OT is necessary and sufficient for *general* MPC.

Yao's garbled circuit protocol. The overall protocol now proceeds as follows:

1. The Garbler holds input x and Evaluator holds input y.
2. (a) (Garbler Step 1) Prepare garbled circuit for C.
   (b) (Evaluator Step 1) Prepare OT queries for each bit of y.
3. Evaluator sends the OT labels for y to the Garbler.
4. (Garbler Step 2) Prepare OT responsese for the Evaluator's inputs. The messages will correspond to wire labels.
5. Garbler sends the OT responses for Evaluator's input, the garbled circuit and the labels for the garbler's own input.
6. (Evaluator Step 2) Evaluate the garbled circuit to learn $C(x, y)$.

Next lecture: from GC to single-key FE. Two party computation is interactive but is sufficient to realize single-key FE. It also can only realize single-key FE since GC is *not* reusable.

We will rely on using a "universal circuit" U that takes the circuit C and input x both as inputs and outputs $C(x)$: $U(C, x) = C(x)$.

Ciphertext will be a garbled circuit for U with wire labels for x. The function key for C will allow one to non-interactively receover the wire labels for C. Decryption will be a GC evaluation.

CS 395T: Topics in Cryptography March 30, 2022

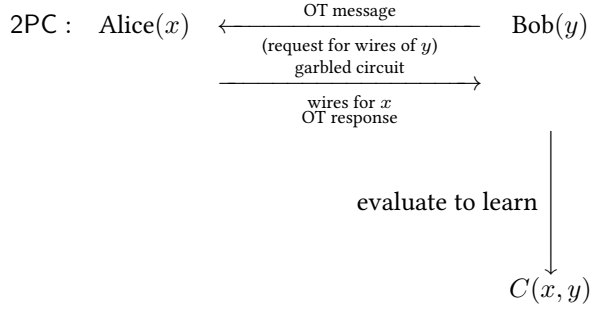
Lecture 19: Succinct Functional Encryption

Lecturer: David Wu *Scribe: Jonathan Li*

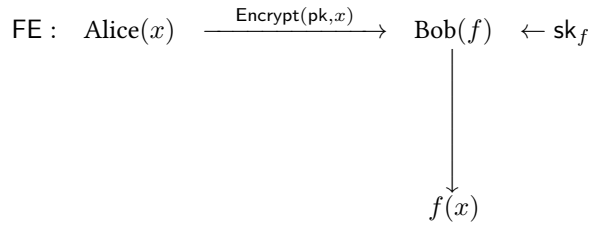
In the last lecture, we saw how to use a garbled circuit and oblivious transfer (OT) to get two-party computation (2PC). In this lecture, we will show that we can make this protocol non-interactive.

19.1 Functional Encryption from Public-Key Encryption

Suppose we start with our 2PC protocol.



Now consider this FE setting.



One way to evaluate something while hiding information is by garbling it. Thus, instead of sending $\mathsf{Encrypt}(\mathsf{mpk}, x)$, Alice could send a garbled circuit that has x embedded inside it. Bob can then attempt to evaluate the circuit to learn $f(x)$, but this requires knowledge of all the garbled gates and input labels. Alice also has no knowledge of f, so she has no idea what function she is going to garble.

The solution is to use a universal circuit. A **universal circuit** U takes as input (C, x) and outputs $C(x)$. We can also define a **restricted universal circuit** U_x that takes as input C and outputs $C(x)$. We modify the above so that Alice sends $(\tilde{U}, \{\ell_i^{(k)}\}) \leftarrow \mathsf{Garble}(1^\lambda, U_x)$ instead of $\mathsf{Encrypt}(\mathsf{mpk}, x)$. Alice cannot send all the wires labels since Bob could then evaluate the identity function to reveal Alice's entire input. In 2PC, this problem was solved through oblivious transfer. To make this protocol non-interactive, we use a public key encryption scheme.

Let $\ell = |C|$ denote the number of input bits to U associated with the circuit C. Let

$$\mathsf{mpk} : \begin{bmatrix} \mathsf{pk}_1^{(0)} & \cdots & \mathsf{pk}_\ell^{(0)} \\ \mathsf{pk}_1^{(1)} & \cdots & \mathsf{pk}_\ell^{(1)} \end{bmatrix} \quad \text{and} \quad \mathsf{msk} : \begin{bmatrix} \mathsf{sk}_1^{(0)} & \cdots & \mathsf{sk}_\ell^{(0)} \\ \mathsf{sk}_1^{(1)} & \cdots & \mathsf{sk}_\ell^{(1)} \end{bmatrix}$$

be a pair of public keys/secret keys associated with each input wire, and let $\mathsf{sk}_C : (\mathsf{sk}_1^{(C_1)}, \ldots, \mathsf{sk}_\ell^{(C_\ell)})$ consist of secret keys corresponding to the bits of C.

$\mathsf{Encrypt}(\mathsf{mpk}, x)$:

1. $(\tilde{U}, \{L_i^{(b)}\}_{i\in[\ell],b\in\{0,1\}}) \leftarrow \mathsf{Garble}(1^\lambda, U_x)$
2. $\mathsf{ct}_i^{(b)} \leftarrow \mathsf{Encrypt}(\mathsf{pk}_i^{(b)}, L_i^{(b)})$

$\mathsf{KeyGen}(\mathsf{msk}, C)$: give out $\mathsf{sk}_i^{(C_i)}$

$\mathsf{Decrypt}(\mathsf{sk}_c, \mathsf{ct}_x)$:

1. $L_i^{(C_i)} \leftarrow \mathsf{Decrypt}(\mathsf{sk}_i^{(C_i)}, \mathsf{ct}_i^{(C_i)})$
2. $y \leftarrow \mathsf{Eval}(\tilde{U}, \{L_i^{(C_i)}\})$

Correctness. Follows from garbled circuit correctness and PKE correctness.

Single-Key Security. Consider the challenge ciphertext

$$\mathsf{ct} = \begin{bmatrix} \mathsf{ct}_1^{(0)} & \cdots & \mathsf{ct}_\ell^{(0)} \\ \mathsf{ct}_1^{(1)} & \cdots & \mathsf{ct}_\ell^{(1)} \end{bmatrix}.$$

By PKE security, we can replace one entry in each column with an encryption of 0. Since KeyGen only gives out one key for each column, security follows by semantic security of PKE. By garbling security, there exists an efficient simulator $S(1^\lambda, 1^{|U_x|}, C(x)) \to (\tilde{U}, L_i^{(U_{x,i})})$ that can simulate ciphertexts given only $(C, C(x))$. In the indistinguishable-based FE security game, the adversary must choose x_0 and x_1 such that $C(x_0) = C(x_1)$. Thus it does not matter whether we started with x_0 or x_1.

Caveats.

- Single-key implies Q-key for any bounded $Q = \mathsf{poly}(\lambda)$ by a combinatoric construction known as "MPC-in-the-head", so ciphertexts scale with Q.
- It is difficult to achieve a fully collusion resistant FE.

Ideally, ciphertext size is independent of (or sublinear in) size of functions. We consider the following stronger notions:

1. **Succinct** FE: running time of encryption is independent of (or depends only on depth, so sublinear in) size of supported functions
2. **Compact** FE: running time of encryption is independent of (sublinear in) *both* size and output length of supported functions.

Single-key compact functional encryption for log-depth circuits implies indistinguishability obfuscation ("crypto-complete"). In particular, single-key compact FE implies fully collusion resistant FE.

Open Question. Can we get compact FE from LWE?

19.2 Succinct FE from Garbled Circuits, ABE, and FHE

We show how to construct succinct FE. The goal is to have the ciphertext size be sublinear in the size of the circuit being evaluated. We first tweak the ABE scheme to encrypt two messages μ_0 and μ_1:

- if $f(x) = 0$, decryption outputs μ_0, which can be built from vanilla ABE by encrypting μ_0 under f and μ_1 under $\overline{f}$.
- if $f(x) = 1$, decryption outputs μ_1.

Now the idea is to use FHE to encrypt x. Given an encryption of x, we can evaluate an encryption of $f(x)$. We also give out a garbled circuit that implements FHE decryption, i.e. takes $(\mathsf{sk}, \mathsf{ct})$ and outputs $\mathsf{Decrypt}(\mathsf{sk}, \mathsf{ct})$. We will use ABE to communicate the wire labels:

$$\mathsf{ct}_{\mathsf{ABE}} \leftarrow \mathsf{ABE.Encrypt}(\mathsf{mpk}_{\mathsf{ABE}}, \mathsf{ct}_x, L_i^{(0)}, L_i^{(1)}).$$

Thus the ciphertext consists of a garbled circuit for FHE decryption circuit, wire labels for FHE secret key, and ABE encryptions of wire labels of FHE ciphertext, where each bit uses an *independent* ABE scheme.

To generate a key for function f, give out ABE keys for each g_i, where g_i is defined as the function that takes ct and outputs the i^{th} bit of $\mathsf{FHE.Eval}(f, \mathsf{ct})$.

To decrypt,

1. use $\mathsf{sk}_{g_1}, \ldots, \mathsf{sk}_{g_\ell}$ to recover labels for $\mathsf{ct}_{f(x)} \leftarrow \mathsf{Eval}(f, \mathsf{ct}_x)$;
2. evaluate the garbled circuit to get $f(x) \leftarrow \mathsf{FHE.Decrypt}(\mathsf{sk}, \mathsf{ct}_{f(x)})$.

Succinctness. We analyse the size of each component of the ciphertext.

1. Garbled circuit for FHE decryption. This has size $\mathsf{poly}(\lambda, d)$ where d is the depth of the computation.
2. Wire labels for FHE secret key. This has size $|\mathsf{sk}| \cdot \mathsf{poly}(\lambda) = \mathsf{poly}(\lambda, d)$.
3. ABE encryptions of wire labels of FHE ciphertext. This has size $\ell \cdot \mathsf{poly}(\lambda, d) = \mathsf{poly}(\lambda, d)$.

It follows that the overall ciphertext size scales with depth rather than size, so this is succinct FE. Note, however, that this is not compact FE.

Single-Key Security. Follows from ABE security, then garbling security, then FHE security.

In summary, we can use garbled circuits and PKE to obtain single-key, non-succinct FE. We can also use garbled circuits, ABE, and FHE to obtain single-key, succinct FHE. Next week, we will revisit zero-knowledge and discuss how to get non-interactive zero-knowledge (NIZK) from LWE.

CS 395T: Topics in Cryptography April 11, 2022

Lecture 20: Designated-Verifier NIZKs

Lecturer: David Wu *Scribe: Kristin Sheridan*

20.1 Recap

- Functional encryption is a generalization of public key encryption in which we have fine-grained access control to encrypted data
- Steps that we went through along the way:
  - Attribute-based encryption
  - Predicate encryption
  - Functional encryption
- Now we switch focus to look at a dual of this problem, where we focus on *integrity* by looking at zero knowledge proofs
- Final goal over the next couple weeks: NIZK from LWE
  - This problem was solved in 2019, but it was open for a decade before that

20.2 Interactive Zero-Knowledge Protocol for Graph Hamiltonicity

First, recall a couple definitions that we will use going forward:

Definition 20.1. A graph is **Hamiltonian** if there exists a cycle that visits every node exactly once.

Definition 20.2. A Σ-protocol is a 3 round interactive honest-verifier zero knowledge (ZK) proof system. (Recall that an honest verifier does not choose its message in this proof maliciously.)

- The Hamiltonian cycle problem is NP-complete, and we will first construct a Σ-protocol for this problem and thus all of NP
- Last semester, we had a Σ-protocol for 3-coloring, which is also NP-complete, but the properties of this protocol, also known as *Blum's protocol*, will lend themselves better to our goal of designated verifier NIZKs
- Our scheme below assumes a statistically binding and computationally hiding commitment scheme, which we know we can get from previous lectures
- Blum's protocol:
  1. The prover samples a random permutation of the vertices in the graph $\pi \leftarrow Perm[V]$ and considers the graph constructed by applying this permutation to the vertex labels
  2. The prover commits to the edges in the permuted graph
     - Specifically, $\forall i, j \in [n]$, if $(i, j) \in E$, $C_{\pi(i),\pi(j)} \leftarrow Commit(1)$ and otherwise $C_{\pi(i),\pi(j)} \leftarrow Commit(0)$
  3. All commitments $C_{i,j}$ are sent to the verifier
  4. The verifier selects a random bit $b \in \{0, 1\}$ and sends it to the prover

5. If the prover receives the bit $b = 0$, it uses its witness that is a Hamiltonian cycle in the graph (which it must know if this is a good instance) and sends back openings for all edges in that cycle
6. If the prover receives bit $b = 1$, it sends openings for all edges it committed to and the permutation it used

- Completeness: by construction; if you're honest you can always answer correctly
- Soundness:
  - Suppose G does not have a Hamiltonian cycle, and we divide the possible scenarios into two exhaustive cases:
    1. The prover commits to a graph $G' \neq \pi(G)$ for some permutation π (ie it commits to a fake graph); in this case, if $b = 1$ the prover must fail because it except with small probability the openings will be to the same edges it committed to (due to statistical binding property this holds even for an unbounded prover), but since it committed to a bad graph there is no permutation to explain those openings/convince the verifier to accept. Thus, in this case Pr[prover succeeds] $\leq \frac{1}{2}$
    2. The prover commits to $G' = \pi(G)$ for some permutation π; in this case if $b = 0$ the prover cannot succeed since there does not exist a Hamiltonian cycle in the permuted graph. Except with negligible probability, it gives the verifier the same openings it committed to and thus cannot give it openings to a Hamiltonian cycle. Thus, we again get Pr[prover succeeds] $\leq \frac{1}{2}$
  - Thus, if G does not have a Hamiltonian cycle, then the probability of the prover succeeding is upper bounded by $1/2$ no matter what. If this is repeated λ times, then it is bounded by $1/2^{\lambda}$.
- Honest verifier zero knowledge (note that honest verifier that we need for our conversion to NIZK so we just focus on proving this)
  - The simulator will act as follows:
    1. Sample $b \xleftarrow{R} \{0, 1\}$.
    2. If $b = 0$, commit to all 1s for the $C_{i,j}$. For the last part, open any permutation of the n nodes you want
    3. if $b = 1$, choose a random permutation and commit to $\pi(G)$. For the last part, open all commitments
    4. Output the commitments $C_{i,j}$, b, and the appropriate openings for the chosen bit
  - Note that as usual in creating a ZK simulator, we reordered when certain things were picked, namely we picked b before making a commitment
  - We now just need that this distribution looks like the distribution of a protocol transcript to any computationally bounded verifier.
    - We see that the distribution of the choice of b is correct
    - When $b = 1$, the distribution of the commitments and openings are also correct, as we proceed exactly as in the protocol
    - When $b = 0$, the opened values do in fact reveal a random cycle; since the original protocol had a fixed cycle and randomly permuted the labels, this distribution is correct. Thus, we are just left with the distribution of the commitments; however, if the commitment to all 1s was computationally distinguishable from an appropriate commitment, the commitment scheme would not be computationally hiding, so we are left with the fact that the distribution we have here is computationally indistinguishable from the correct one

- Why do we look at this specific protocol? Note that if we repeat the protocol λ times for a graph that does *not* have a Hamiltonian cycle, then there is only one challenge (one choice of randomness by the verifier) on which the prover can successfully win, despite the fact that there are 2^{λ} challenges. In the 3-coloring case from last semester there were multiple challenges on which a verifier could win in each round, making the the number of successful challenges blow up with λ and making analysis harder for our ultimate goals

20.3 One-Time Designated-Verifier NIZK for Graph Hamiltonicity

Definition 20.3 (Designated verifier NIZK)**.** A designated verifier NIZK consists of the following algorithms:

$\text{Setup}(1^\lambda) \to \text{pk, sk}$

$\text{Prove}(\text{pk}, x, w) \to \pi$

$\text{Verify}(\text{sk}, x, \pi) \to b' \in \{0,1\}$

In this system, pk is a public key known to all, whereas sk is a secret key known only to the verifier (thus requiring that the verifier be "designated" in order to properly verify). All regular properties of honest verifier NIZKs must hold (ie correctness, soundness, HVZK).

The scenario of a DV-NIZK might be interesting in a setting where everyone publishes their own public key that others can use to prove secrets them, and everyone retains a matching secret key they use for their own verification. Note that our construction here can also be extended using a common random string to extend zero knowledge to a scenario in which the verifier chooses the public key maliciously, but we focus on honest verifiers here.

Now we have a construction of a one-time DV-NIZK from public key encryption (PKE). It is an analog to the one-key FE system we created from PKE.

- Recall that when we have a Σ protocol, the prover sends a commitment to verifier, the verifier sends challenge, and prover sends responses/openings
- Suppose the verifier's challenge is sampled from a polynomial sized space (for a single iteration in the original protocol there were only 2 possible challenges)
- The idea we will use is that we will have a public key for each possible challenge, and we will encrypt *all* possible responses but ensure that the verifier can only open appropriate ones. In particular this construction assumes that each round of the protocol has only two possible challenges (though can be easily extended to any polynomial number of challenges in each round) and λ rounds are completed.
  - $\text{Setup}(1^\lambda)$
    1. $\forall i \in [\lambda], \forall b \in \{0,1\}$: $(\mathsf{pk}_i^{(b)}, \mathsf{sk}_i^{(b)}) \leftarrow \mathsf{PKE.Setup}(1^\lambda)$
    2. $\text{pk} \leftarrow \begin{bmatrix} \mathsf{pk}_1^{(0)} & \mathsf{pk}_2^{(0)} & \cdots & \mathsf{pk}_\lambda^{(0)} \\ \mathsf{pk}_1^{(1)} & \mathsf{pk}_2^{(1)} & \cdots & \mathsf{pk}_\lambda^{(1)} \end{bmatrix}$
    3. Sample a challenge$b_1, \ldots, b_\lambda \leftarrow \{0,1\}^\lambda$ at random
    4. $\text{sk} \leftarrow (b_1, b_2, \ldots, b_\lambda, \mathsf{sk}_1^{(b_1)}, \mathsf{sk}_2^{(b_2)}, \ldots, \mathsf{sk}_\lambda^{(b_\lambda)})$
    5. Output (pk,sk)
  - $\text{Prove}(\text{pk}, x, w)$
    1. Following the Σ-protocol, construct the first message σ_i for λ simulations of the first round of the Σ-protocol.
    2. $z_i^{(b)}$ is set to be the response a prover would give in in the Σ-protocol with first message σ_i if the verifier picks challenge b
    3. $\mathsf{ct}_i^{(b)} \leftarrow \mathsf{PKE.Encrypt}(\text{pk}, z_i^{(b)})$
    4. Output $(\sigma_1, \sigma_2, \ldots, \sigma_\lambda, \mathsf{ct}_1^{(0)}, \mathsf{ct}_1^{(1)}, \ldots, , \mathsf{ct}_\lambda^{(1)})$
  - $\text{Verify}(\text{sk}=(b_1, b_2, \ldots, b_\lambda, \mathsf{sk}_1^{(b_1)}, \mathsf{sk}_2^{(b_2)}, \ldots, \mathsf{sk}_\lambda^{(b_\lambda)}), x, \pi = (\sigma_1, \sigma_2, \ldots, \sigma_\lambda, \mathsf{ct}_1^{(0)}, \mathsf{ct}_1^{(1)}, \ldots, , \mathsf{ct}_\lambda^{(1)}))$
    1. $\forall i \in [\lambda]$: $z_i^{(b_i)} \leftarrow \mathsf{PKE.Decrypt}(\mathsf{sk}_i^{(b_i)}, \mathsf{ct}_i^{(b_i)})$
    2. $\forall i \in [\lambda]$: Check that $(\sigma_i, b_i, z_i^{(b_i)}$ is an accepting transcript; output 1 if so and 0 otherwise
- Correctness: by construction

- Soundness: comes directly from soundness of the Σ protocol
- HVZK: we construct a simulator as usual, which we denote $S(1^\lambda, x)$
  1. Use HVZK simulator S' for the simulator protocol to generate λ transcripts $(\sigma_i, b_i, z_i^{(b_i)})$
  2. $\forall i \in [\lambda], \forall b \in \{0,1\}$: $(\mathsf{pk}_i^{b_i}, \mathsf{sk}_i^{(b_i)}) \leftarrow \mathsf{PKE.Setup}(1^\lambda)$; let $\mathsf{sk}_i \leftarrow \mathsf{sk}_i^{(b_i)}$
  3. $\mathsf{pk} \leftarrow \begin{bmatrix} \mathsf{pk}_1^{(0)} & \mathsf{pk}_2^{(0)} & \cdots & \mathsf{pk}_\lambda^{(0)} \\ \mathsf{pk}_1^{(1)} & \mathsf{pk}_2^{(1)} & \cdots & \mathsf{pk}_\lambda^{(1)} \end{bmatrix}$
  4. $\mathsf{sk} \leftarrow (b_1, b_2, \ldots, b_\lambda, \mathsf{sk}_1^{(b_1)}, \mathsf{sk}_2^{(b_2)}, \ldots, \mathsf{sk}_\lambda^{(b_\lambda)})$
  5. $\forall i \in [\lambda]$: $\mathsf{ct}_i^{(b_i)} \leftarrow \mathsf{PKE.Encrypt}(\mathsf{pk}_i^{(b_i)}, z_i)$ and $\mathsf{ct}_i^{(1-b_i)} \leftarrow \mathsf{PKE.Encrypt}(\mathsf{pk}_i^{(1-b_i)}), 0)$
  6. Output pk, sk, and $\pi \leftarrow (\sigma_1, \sigma_2, \ldots, \sigma_\lambda, \mathsf{ct}_1^{(0)}, \mathsf{ct}_1^{(1)}, \ldots, , \mathsf{ct}_\lambda^{(1)})$
- The only difference between the real transcript and this one is that the real transcript has all 0s encrypted for the responses that are not checked (the responses we don't hand out the secret keys for); however, PKE security says that we can't distinguish when we've encoded the 0s string vs another string (try replacing the messages one at a time via hybrids to get the whole proof)

Notably, the system we have here is one-time use only. A reusable system would let a single verifier check proofs on multiple inputs and retain all the same properties. However, we see that soundness fails, as in order for soundness to make sense in a version where a verifier is reused, we want to give oracle access for the verifier to the prover (to model the fact that it could see acceptances/rejections on previous tries). In particular, we can construct an attack on this concept of soundness in the following way, via a *verification rejection attack*:

- The prover starts with valid proof of a good statement and a grid of openings for this that are all honestly generated, and the verifier will accept
- The prover repeats this but for each i, it replaces $\mathsf{ct}_i^{(0)}$ with $\mathsf{PKE.Encrypt}(\mathsf{pk}_i^{(0)}, \perp)$ (keeping all other values the same)
  - If the verifier rejects this modified proof, we know that $b_i = 0$ since the verifier is actually decrypting the new ciphertext
  - If the verifier accepts this modified proof, we know that $b_i = 1$ since the verifier is not actually decrypting the new ciphertext

How can we extend this to get reusability? In the interactive setting, soundness came from having a fresh, unpredictable challenge every time, but the challenge has become fixed by becoming non-interactive. How can we get something fresh and unpredictable every time without letting the verifier contribute to the challenge?

We will do this in upcoming classes, but as a hint: replace public key encryption with attribute-based encryption to see if you get something useful and combine with a pseudorandom function.

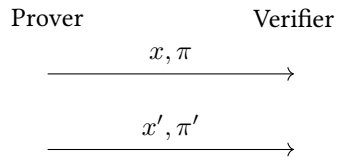
Lecture 21: Reusable Designated-Verifier NIZKs

Lecturer: David Wu *Scribe: Garrett Gu*

21.1 Reusable Designated-Verifier NIZKs

What we found in the previous lecture is that if the attacker is given oracle access to the verifier, they can craft fake proofs to leak the challenge bit by bit. Then the underlying Σ protocol is trivially broken, and the attacker can craft proofs of arbitrary statements. This is known as a verifier rejection attack.

We would like to construct a non-interactive protocol that has reusable soundness. In our setting, the verifier starts with a random string generated at setup time, and the prover sends statements and proofs for the verifier to check. So we end up with a problem: the verifier cannot participate in the protocol, but it must still somehow inject fresh randomness into the protocol as new statements come in.



Construction overview. Our basic idea is to use attribute-based encryption combined with a pseudorandom function to create a fresh challenge every time. Suppose we have a PRF:

$$F : K \times ([\lambda] \times \{0,1\}^n) \to \{0,1\} \tag{21.1}$$

Then $F(k,(i,x))$ denotes the ith challenge input for a proof of statement x. The reason why the base scheme fails is because the same challenge is used to verify both false and true statements. Thus, if we base the random strings on a PRF evaluated on the statement itself, information gained from verifying a true statement does not translate to gaining information about a false statement.

Note that the prover cannot be the one evaluating the PRF, since the prover would have to have the key and break randomness. But how can we get the verifier to derive the challenge from the PRF without participating in the scheme? The high-level idea is to use homomorphic evaluating to evaluate the PRF at verification time. We can use attribute-based encryption to do this.

Define the function f_{i^*}. This function will be used to enforce the invariant from last lecture that only one ciphertext from each column can be decrypted at a time, and only according to the corresponding bit in the challenge string.

$$f_{i^*}(x,i,b) := \begin{cases} 0 & \text{if } F(k,(i,x)) = b \text{ and } i = i^* \\ 1 & \text{otherwise} \end{cases} \tag{21.2}$$

21.2 Construction based on ABE

We construct the designated-verifier NIZK as follows:

- $\mathsf{Setup}(1^\lambda)$:
  - Let $k \xleftarrow{R} K$ where K is the keyspace of a PRF.
  - Let $(\mathsf{mpk}, \mathsf{msk}) \leftarrow \mathsf{ABE.Setup}(1^\lambda)$.
  - Let $\mathsf{sk}_i \leftarrow \mathsf{ABE.KeyGen}(\mathsf{msk}, f_{i,k})$ for each $i \in [\lambda]$.

- Output $\mathsf{pk} = \mathsf{mpk}$ and $\mathsf{sk} = (\mathsf{sk}_1, ...\mathsf{sk}_\lambda, k)$.

- $\mathsf{Prove}(\mathsf{pk}, x, w)$:
  - Construct the first message of the Σ-protocol, $\sigma_1, ...\sigma_\lambda$.
  - For each σ_i, compute responses $z_i^{(0)}$ and $z_i^{(1)}$ to be the responses associated with challenge bits 0 and 1 respectively.
  - Compute $\mathsf{ct}_i^{(b)} \leftarrow \mathsf{ABE.Encrypt}(\mathsf{mpk}, (x, i, b), z_i^{(b)})$.
  - Output $\pi = (\sigma_1, ..., \sigma_\lambda, \mathsf{ct}_1^{(0)}, \mathsf{ct}_1^{(1)}, ..., \mathsf{ct}_\lambda^{(0)}, \mathsf{ct}_\lambda^{(1)})$

- $\mathsf{Verify}(\mathsf{sk}, x, \pi)$:
  - Let $b_i \leftarrow F(k, (i, x))$ for each $i \in [\lambda]$.
  - Compute $z_i^{(b_i)} \leftarrow \mathsf{ABE.Decrypt}(\mathsf{sk}_i, \mathsf{ct}_i^{(b_i)})$ for each $i \in [\lambda]$.
  - Verify that $(\sigma_i, b_i, z_i^{(b_i)})$ is valid for each $i \in [\lambda]$.

Completeness. First we note that completeness holds for this scheme. By definition $f_i(x, i, b) = 0$ whenever $b_i = F(k, (i, x))$, so the verifier is able to decrypt $z_i^{(b_i)}$ for each $i \in [\lambda]$. Thus completeness follows from the underlying HVZK scheme.

Zero-knowledge. Now let's prove zero-knowledge of the protocol. First note that for all $i, j \in [\lambda]$, $f_i(x, j, b) = 1$ when $b = 1 - F(k, (i, x))$. Therefore, in each column, at least one entry can be replaced with the encryption of the all-zeros string by ABE security.

Soundness. Now we sketch an argument for soundness.

Prover (adversary) Verifier (challenger)

$\xrightarrow{x, \pi}$

If $x \notin \mathcal{L}$, then the verifier uses a uniform, unnpredictable challenge b to check π; thus, soundness reduces to Σ-protocol soundness

The challenge that was generated when $x \notin \mathcal{L}$ is just as good as the uniformly random challenge generated in the interactive case.

However, we actually hit a problem with soundness in this protocol. If the adversary is able to query the verification oracle, the output of the decryption oracle in the invocation of $\mathsf{ABE.Decrypt}$ could leak information about sk_i, which contains information about the PRF key.

Therefore, we must figure out a way to argue that the PRF key cannot be leaked through the $\mathsf{ABE.Decrypt}$call, even though the ciphertext is controlled by the adversary.

First note that if the prover were honest (all ciphertexts are honestly generated), we wouldn't have any issue. This follows by correctness of the ABE scheme, since if the key is able to decrypt at all, it would decrypt to the corresponding $z_i^{(b)}$ with probability 1, and thus we would never leak any information about the PRF key.

Now's a good time to bring up a major open question in cryptography. Given a black box CPA-secure cryptosystem, is it possible to construct a CCA-secure cryptosystem? In our case, we have a CPA-secure ABE encryption scheme, but is it possible to build a CCA-secure cryptosystem with this? We don't know the answer generally, but it turns out that the lattice-based ABE encryption we constructed can be easily tweaked to achieve CCA security.

Recall that an ABE ciphertext has form:

$$\begin{aligned}
&s^T A + \text{error} \\
&s^T[B_1 - x_1 G | ... | B_l - x_l G] + \text{error} \\
&s_T p + \mu \times \left\lfloor \frac{q}{2} \right\rceil + \text{error}
\end{aligned}$$

Now let the secret key for function f be T_f, which is a trapdoor for $[A|B_f]$ ($[A|B_f] \cdot T_f = G$), where $B_f = [B_1|...|B_l] \cdot H_f$.

Now decryption is performed as below:

$$\begin{aligned}
&(s^T[B_1 - x_1 G | ... | B_l - x_l G] + \text{error}) H_{f,x} \\
\approx &s^T(B_f - f(x) \cdot G) \\
= &S^T[A|B_f] \qquad \text{when } f(x) = 0
\end{aligned}$$

Using this and the trapdoor, we can compute

$$\begin{aligned}
&s^T[A|B_f] \cdot T_f + \text{error} \\
= &s^T G + \text{error}
\end{aligned}$$

And we can recover s, the encryption randomness, by solving LWE. We can also recover the errors in the ciphertext, and check that the errors are sufficiently small. If the errors are small enough, then we have correctness, since every decryption will be successful.

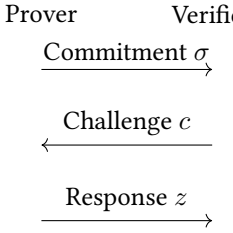
To summarize, decryption introduces a ciphertext validity check with the guarantees:

1. If validity check passes, then decryption with sk_f can be simulated by decrypting with mskor an all-zeroes key.
2. If the validity check does not pass, then decryption always returns $\perp$.

So we have erased the PRF key from the output of the decryption oracle, and we can replace the PRF with a black box. Thus, using our modified ABE encryption scheme, we can reduce soundness of the protocol to soundness of the underlying Σ-protocol.

21.3 Publicly-Verifiable NIZKs

Now, what about public verification. First, let's recall an approach from last semester on the random oracle model:



Now, to get rid of the interaction and get a NIZK, we can simply derive $c \leftarrow H(x, \sigma)$ where H is modeled as a random oracle. This challenge is only determined after the prover has selected a commitment and is therefore unpredictable.

But the ROM is a bit idealistic. In practice, we just instantiate the random oracle with a cryptographic hash function (like SHA-256). We cannot prove security since SHA-256 is not truly a random oracle, but we don't know of any attacks.

But can we identify some condition of a hash function that can be used to prove security?

This brings us to the notion of correlation-intractability, which we will explore in more depth next lecture.

Let $R(x, y)$ be a binary relation. We say that a hash function $H : X \to Y$ is correlation-intractable for the relation $R : X \times Y \to \{0, 1\}$ if no efficient adversary can find an $x \in X$ such that

$$R(x, H(x)) = 1$$

Intuitively, this correlation-intractability property is useful because for any statement $x \notin \mathcal{L}$, we can define a BadChallenge binary relation R_x where $R_x(\sigma, c) = 1$ if there exists some z such that

$$\mathsf{Verify}(x, (\sigma, c, z)) = 1$$

If we have a correlation-intractable hash function on this relation, then the adversary is prevented from finding a σ, c that causes the verification to pass, and we're good to go!

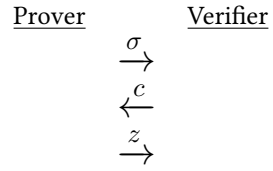
CS 395T: Topics in Cryptography April 11, 2022

Lecture 22: Correlation-Intractability and NIZKs

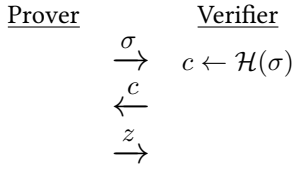
Lecturer: David Wu *Scribe: Alexander Burton*

22.1 Recap

Recall the general form of a Σ-protocol. The prover sends a first message σ (usually a description of the statement and initial commitments), the verifier sends a challenge c (typically, at random), and the prover responds to the challenge in z.



Given a Σ-protocol, we want to construct a non-interactive zero knowledge (NIZK) protocol. The sticking point is that we need a random challenge string for soundness, but the verifier cannot participate to generate the challenge. How can we ensure that the prover's challenge is random? One idea is to use the random oracle model, and apply the random oracle $\mathcal{H}$ to σ.



To avoid using a random oracle, it turns out we can use a hash function $H(\mathsf{hk}, -)$ with the following property.

Definition 22.1. Let $\mathcal{R} : X \times Y \to \{0,1\}$ be a binary relation. We say a hash function $H : X \to Y$ is *correlation-intractable* (c.i.) with respect to $\mathcal{R}$ if no efficient adversary can find $x \in X$ such that $\mathcal{R}(x, H(x)) = 1$.

22.2 NIZKs from Circular-Secure FHE

For $x \notin \mathcal{L}$, we define the "bad challenge" relation with $\mathcal{R}_x(\sigma, c) = 1$ iff there exists z such that

$$\mathsf{Verify}(x, (\sigma, c, z)) = 1.$$

It is clear that any H that is correlation-intractable for $\mathcal{R}_x$ results in information-theoretic soundness for the statement x. More explicitly, we have the following NIZK protocol:

Construction 22.2 (NIZK from Correlation-Intractable Hash Function)**.**

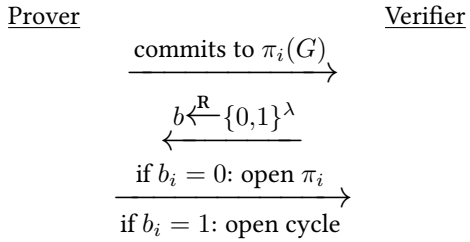
- $\mathsf{Setup}\left(1^\lambda\right)$: Sample $\mathsf{hk} \leftarrow \mathsf{CI.Setup}\left(1^\lambda\right)$.
- $\mathsf{Prove}(\mathsf{hk}, x, w)$: Compute $\sigma \leftarrow$ [Σ first message], $c \leftarrow H(\mathsf{hk}, \sigma)$, $z \leftarrow$ [Σ response for (σ, c)]. Output $\pi \leftarrow (\sigma, z)$.
- $\mathsf{Verify}(\mathsf{hk}, x, \pi = (\sigma, z))$: Compute $c \leftarrow H(\mathsf{hk}, \sigma)$, then use Σ verification for (σ, c, z).

Completeness follows from the base Σ-protocol, since given a true statement we can always respond to the challenge c, regardless of how it is generated. Soundness is immediate and information-theoretic (assuming $H(\mathsf{hk}, -)$ is c.i. for $\mathcal{R}_x$) since by definition there does not exist a z where $\mathsf{Verify}(x, (\sigma, c, z)) = 1$.

As is, zero-knowledge does not follow from zero-knowledge of the Σ-protocol: the Σ-protocol's simulator produces transcripts where c is sampled at random and independent of x, σ, but in the above protocol c depends on x, σ. To fix this, we generate a shift $\rho \xleftarrow{R} \{0,1\}^n$ to be included in the $\mathsf{crs} = (\mathsf{hk}, \rho)$. The challenge is then computed as $c \leftarrow H(\mathsf{hk}, \sigma) \oplus \rho$. The simulator then does the process in reverse, first obtaining (σ, c, z) from the base simulator, and then computing $\rho \leftarrow H(\mathsf{hk}, \sigma) \oplus c$. Observe that since c is uniform over $\{0,1\}^n$, so is ρ, so this induces the desired distribution.

22.2.1 Correlation-Intractability for Search Relations

It remains to explicitly construct a c.i. hash function (CIHF) for e.g. Blum's Protocol for graph hamiltonicity. Recall the general structure of (repeated) Blum's Protocol:



A key property that makes Blum's protocol easier to work with is that for every choice of first message σ, there is a unique bad challenge string out of the challenge space $\{0,1\}^\lambda$ (assuming statistically binding). To this end, we can define a so-called *search relation*: for any x, there exists a unique y such that $\mathcal{R}(x, y) = 1$; define $f(x) = y$. Clearly, the bad challenge relation for Blum's protocol induces a search relation. However, it is unclear how to ensure that the search relation is efficiently computable from the commitments. To fix this, we can use an extractable commitment scheme—one in which we can extract the committed values given a trapdoor. We have already constructed such a scheme in the past: GSW.

We can now attempt to construct a CIHF for a search relation $f : \{0,1\}^k \to \{0,1\}^n$ as follows.

Construction 22.3 (Candidate CIHF for Search Relations)**.** *(Warning: this construction is broken!)*

- $\mathsf{Setup}\left(1^\lambda, f\right)$: set $\mathsf{hk} \leftarrow f$.
- $\mathsf{Eval}(\mathsf{hk} = f, x)$: compute and output $H(\mathsf{hk}, x) := f(x) \oplus \left(0^{n-1} \| 1\right)$. (This is flipping the last bit of $f(x)$)

This construction is c.i. information theoertically. Indeed, $H(\mathsf{hk}, x) \neq f(x)$ by construction, so $\mathcal{R}(x, H(\mathsf{hk}, x)) = 0$ necessarily. However, it is not suitable for our purposes. The problem is that the hash key is public, which leaks f, completely breaking the hiding property necessary for soundness. However, a clever insight lets us fix this problem: we can use homomorphism to hide the evaluation of f!

Construction 22.4 (CIHF for Search Relations from Circular-Secure FHE)**.**

- $\mathsf{Setup}\left(1^\lambda, f\right)$: First sample
$$(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{FHE.KeyGen}\left(1^\lambda\right).$$
Compute $\mathsf{ct} \leftarrow \mathsf{FHE.Encrypt}\left(\mathsf{pk}, 0^{|f|}\right)$. Output $\mathsf{hk} \leftarrow (\mathsf{pk}, \mathsf{ct})$.
- $\mathsf{Eval}(\mathsf{hk} = (\mathsf{pk}, \mathsf{ct}), x)$: compute and output $H(hk, x) := \mathsf{FHE.Eval}(\mathsf{pk}, \mathcal{U}_x, \mathsf{ct})$. Here, for $x \in \{0,1\}^k$ we denote the universal circuit as $\mathcal{U}_x(f) \to f(x)$.

Claim 22.5. *The above construction is a CIHF for any efficient search relation f.*

Proof. We use a hybrid argument. Hyb_0 is the real game:

Adversary		Challenger
		$(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{FHE.KeyGen}\left(1^\lambda\right)$
		$\mathsf{ct} \leftarrow \mathsf{FHE.Encrypt}\left(\mathsf{pk}, 0^{\lvert f\rvert}\right)$
	$\xleftarrow{\mathsf{hk}=(\mathsf{pk},\mathsf{ct})}$	
	$\xrightarrow{x}$	Check $f(x) = H(\mathsf{hk}, x)$

Hyb_1 is constructed as follows. First define

$$f'(x) := \mathsf{FHE.Decrypt}(\mathsf{sk}, f(x)) \oplus \left(0^{n-1} \| 1\right),$$

where we treat the output of $\mathsf{FHE.Decrypt}$ as a bitstring. The game is identical, except we sample $\mathsf{ct} \leftarrow \mathsf{FHE.Encrypt}(\mathsf{pk}, f')$. We can see that $\mathsf{Hyb}_0 \overset{C}{\approx} \mathsf{Hyb}_1$ by circular security, since f' depends on sk.

We now claim that in Hyb_1, c.i. follows statistically: i.e. there does not exist x where $f(x) = H(\mathsf{hk}, x)$. Indeed, assume for the sake of contradiction such an x exists. Then

$$\begin{aligned} f(x) &= H(\mathsf{hk}, x) \\ &= \mathsf{FHE.Eval}(\mathsf{pk}, \mathcal{U}_x, \mathsf{ct}) \\ &= \text{some } \mathsf{FHE.Encrypt}\left(\mathsf{pk}, \mathcal{U}_x(f')\right) \\ &= \text{some } \mathsf{FHE.Encrypt}\left(\mathsf{pk}, f'(x)\right) \\ &= \text{some } \mathsf{FHE.Encrypt}\left(\mathsf{pk}, \mathsf{FHE.Decrypt}\left(\mathsf{sk}, f(x)\right) \oplus \left(0^{n-1} \| 1\right)\right). \end{aligned}$$

Applying $\mathsf{FHE.Decrypt}(\mathsf{sk}, -)$ to both sides yields

$$\mathsf{FHE.Decrypt}(\mathsf{sk}, f(x)) = \mathsf{FHE.Decrypt}\left(\mathsf{sk}, f(x)\right) \oplus \left(0^{n-1} \| 1\right),$$

but this is a contradiction. □

Observe that unlike the broken construction, f is hidden because $(\mathsf{pk}, \mathsf{ct})$ are independent of f in the real security game.

Lecture 23: NIZKs from LWE

Lecturer: David Wu *Scribe: Steven Xu*

23.1 Wrapup of NIZKs from FHE

Now that we have a CIHF, all we have to do is show that the search function $f(x)$ is efficiently computable (given a certain trapdoor) when using an extractable commitment (GSW). The GSW extractable commitment scheme works as follows. First, we sample a GSW keypair $(\mathrm{pk}, \mathrm{sk}) = (A, s)$:

$$A = \left[\frac{\overline{A}}{\overline{s}^T\overline{A} + \overline{e}^T}\right], s = [-\overline{s} \mid 1] \text{ where } \overline{A} \xleftarrow{R} \mathbb{Z}_q^{n\times m}, \overline{s} \xleftarrow{R} \mathbb{Z}_q^n, \overline{e} \xleftarrow{R} \chi^m.$$

The verifier is sent the public key A. For a bit μ, the commitment is $AR + \mu \cdot G$ for $R \xleftarrow{R} \{0,1\}^{m\times m}$—the same as the GSW encryption of μ. To open a commitment, the prover reveals the value of R, allowing the verifier to compute $(AR + \mu \cdot G) - (AR) = \mu \cdot G$ and therefore μ.

Our variant of Blum's protocol for a graph (V, E) then works as follows:

1. Prover samples a permutation of the nodes $\pi \xleftarrow{R} \mathrm{Perm}[V]$, as well as a GSW keypair (A, s).
2. Prover sends the public key A and commits to π as well as the edges $\{e_{ij}\}_{ij}$ of the permuted graph, yielding c_π and $\{c_{ij}\}_{ij}$ respectively.
3. The verifier chooses a random bit $b \xleftarrow{R} \{0,1\}$ which it sends to the prover.
4. If $b = 0$, the prover opens π as well as all the edges. If $b = 1$, the prover opens only the edges which are a part of the Hamiltonian cycle.

In the repeated version of the above protocol, we can then compute the bad challenge given the trapdoor. First, we use the trapdoor to decrypt all the commitments, giving us π and $\{e_{ij}\}_{ij}$. Then for each run, the bad challenge bit is 0 if the graph permutation π is consistent with the committed edges $\{e_{ij}\}_{ij}$, and 1 otherwise.

23.2 CIHFs from SIS

As it turns out, we can construct a CIHF without using the circular security assumption required by FHE. Below is a construction of a CIHF using only the SIS assumption. Note that the NIZK protocol as a whole still requires the LWE assumption to guarantee the security of the commitment scheme.

First, we define some notation. Given $B_1, ..., B_l$ and $f : \{0,1\}^l \to \{0,1\}$ we write

$$[B_1 \mid ... \mid B_l] \cdot H_f = B_f,\ [B_1 - x_1 G \mid ... \mid B_l - x_l G] \cdot H_{f,x} = B_f - f(x)G.$$

If $g : \{0,1\}^l \to \{0,1\}^t$ has t output bits, we write g_t to denote the tth bit of g, and

$$H_g = [H_{g_1} \mid ... \mid H_{g_t}],\ H_{g,x} = [H_{g_{1,x}} \mid ... \mid H_{g_{t,x}}],\ [B_1 \mid ... \mid B_l] \cdot H_g = [B_{g_1} \mid ... \mid B_{g_t}] = B_g,$$
$$[B_1 - x_1 G \mid ... \mid B_l - x_l G] \cdot H_{g,x} = [B_{g_1} - g_1(x)G \mid ... \mid B_{g_t} - g_t(x)G] = B_g - g(x) \otimes G.$$

Then our CIHF construction is

- Setup(1^λ): Sample $A \xleftarrow{R} \mathbb{Z}_q^{n\times m}$, $R_1, ..., R_l \xleftarrow{R} \{0,1\}^{m\times k}$, and $b \xleftarrow{R} \mathbb{Z}_q^n$. Let $B_i = AR_i$, and output $\mathrm{hk} = (b, B_1, ..., B_l)$. l will be the description length of our function f (i.e the universal circuit from last time) and $k = n\lfloor \log q \rfloor$. The B_i can be viewed as commitments to the all zeros string.

- Hash(hk, x): Compute $B_{U_x} = [B_1 \mid ... \mid B_l] \cdot H_{U_f} \in \mathbb{Z}n \times k^2$. Output $G^{-1}(b + B_{U_x} \cdot G^{-1}(z))$ for some $z \in \mathbb{Z}_q^{nk}$ which will be defined later. The output will make more sense once we finish the proof.

To prove correlation-intractability for f, our bad challenge function, we will use a hybrid argument. Hyb_0 will be the real game, and Hyb_1 will be the same except with $B_i = AR_i + f_iG$. The two are indistinguishable by the leftover hash lemma.

Suppose an adversary can find x such that Hash(hk, x) = $f(x)$ in Hyb_1. Then we have

$$\begin{aligned}
f(x) &= \mathrm{Hash}(\mathrm{hk}, x) = G^{-1}(b + B_{U_x} \cdot G^{-1}(z)) \\
G \cdot f(x) &= b + B_{U_x} \cdot G^{-1}(z) \\
&= b + (B_{U_x} - f(x) \otimes G + f(x) \otimes G) \cdot G^{-1}(z) \\
&= b + ([B_1 - f_1 G \mid ... \mid B_l - f_l G] \cdot H_{U_x,f} + f(x) \otimes G) \cdot G^{-1}(z) \\
&= b + A[R_1 \mid ... \mid R_l] \cdot H_{U_x,f} G^{-1}(z) + (f(x) \otimes G) G^{-1}(z) \\
&= b + Av + (f(x) \otimes G) G^{-1}(z).
\end{aligned}$$

Notice that since v is short, if we can somehow get $(f(x) \otimes G)G^{-1}(z) = G \cdot f(x)$ with a clever choice of z, then we would have $Av = -b$, an ISIS solution (recall that b is random).

We start by rearranging our nk-dimensional vector z into a $k \times n$ vector Z, going row first. Let Z_i^T denote the ith row of Z, and let $f^{(i)}(x)$ denote the ith output bit of $f(x)$. Then

$$\begin{aligned}
(f(x) \otimes G)G^{-1}(z) &= [f^{(1)} \cdot G \mid ... \mid f^{(k)} \cdot G] \cdot G^{-1}(z) \\
&= \left[f^{(1)} \cdot G \cdot G^{-1} \begin{bmatrix} z_1 \\ ... \\ z_l \end{bmatrix} \mid ... \mid f^{(k)} \cdot G \cdot G^{-1} \begin{bmatrix} z_{(k-1)l} \\ ... \\ z_{kl} \end{bmatrix} \right] \\
&= \left[f^{(1)} \cdot \begin{bmatrix} z_1 \\ ... \\ z_l \end{bmatrix} \mid ... \mid f^{(k)} \begin{bmatrix} z_{(k-1)l} \\ ... \\ z_{kl} \end{bmatrix} \right] \\
&= \begin{bmatrix} z_1 & ... & z_{(k-1)l} \\ ... & ... & ... \\ z_l & ... & z_{kl} \end{bmatrix} \cdot f(x).
\end{aligned}$$

This means that if we want $(f(x) \otimes G)G^{-1}(z) = G \cdot f(x)$, we can simply choose z such that the matrix of z_i's produced above is G. Thus our CIHF construction is secure under the SIS assumption.

CS 395T: Topics in Cryptography April 18, 2022

Lecture 24: Multi-Key Fully Homomorphic Encryption

Lecturer: David Wu *Scribe: Yingchen Wang*

Fully Homomorphic Encryption as we have seen so far, computation is possible for data encrypted under a single secret key. Any user can generate public key and secret key pair (pk, sk), where they can encrypt a data and under the public key pk: ct = Encrypt(pk,m). Later user can give the ct to someone, who can update the ciphertext with arbitrary function: ct' = Eval(f,ct). The ct' can be decrypted by the user under the secret key sk: Decrypt(sk, ct'), which will give the user f(m).

The above is the single user setting for Fully Homomorphic Encryption. However in practice it is not always going to be the case that one party always hold the secret key for Fully Homomorphic Encryption scheme. In fact, in many settings it can be problematic for one party to always hold the secret key, because the capability of one decryption scheme is controlled by one party completely. Ideally, we would want to have multiple parties, where each of them has the ownership of their own data and still we enable computation on top of that. What if we have ciphertext encrypted under different keys?

24.1 Application: Two-Round Multiparty Computation

Suppose we have 3 parties: P_1, P_2, and P_3, where the 3-party setting is a simplified version of thousands-party setting:

1. Three parties will each have their own input: $P_1(x_1)$, $P_2(x_2)$, $P_3(x_3)$.
2. Then each pair of parties will exchange a sequence of messages so that in the end all of the parties learn $f(x_1, x_2, x_3)$, where f is an arbitrary function.

An application of Multi-Party computation can be auction system. In Denmark, sugar beet auction is run with as a Multi-Party computation, where all prices are ensures to be hidden until all bidders submitted their prices.

24.2 Two-Party Computation from FHE

For 2-party Multi-Party computation, FHE is sufficient. Suppose we have 2 parties: P_1, P_2.

1. Party 1 will generate (pk, sk) for their FHE scheme, and send over (pk, Encrypt(pk, x_1)) to Party 2.
2. Party 2 will homomorphically compute the function with x_2: Encrypt(pk, $f(x_1, x_2)$) and send it to Party 1.
3. Party 1 can decrypt Encrypt(pk, $f(x_1, x_2)$) with sk to learn $f(x_1, x_2)$.

Party 2 in this case rely on Party 1 to learn the output $f(x_1, x_2)$. Can we have a protocol that ensures that either all parties learn the output or no party learns the output? This is a notion called **fairness**. Achieve fairness for Multi-Party computation in a general setting is impossible. We will show a weaker setting, where all parties behave honestly.

What if there are 3 parties and then they try to extend the above protocol by:

1. Party 1 will generate (pk, sk) for their FHE scheme, and send over (pk, Encrypt(pk, x_1)) to Party 2.
2. Party 2 will compute the encryption of x_2: Encrypt(pk, x_2) and send it with (pk, Encrypt(pk, x_1)) to Party 3.
3. Party 3 will homomorphically compute the function with x_3: Encrypt(pk, $f(x_1, x_2, x_3)$).

The issue with above scheme is that: If Party 1 and 3 collude, Party 3 can give Party 1 Encrypt(pk, x_2). Since Party 1 has the sk, Party 2's input is revealed.

24.3 Multi-Key Fully Homomorphic Encryption

Setting.

1. Each party P_i will sample their own public key and secret key pk_i, sk_i. Then they will generate a ciphertext ct_i which is the encryption of their input under their own key: Encrypt(pk_i, x_i). The ct_i is broadcast to everyone.
2. Every party P_1, ..., P_N will homomorphically function f on ct_1, ..., ct_N, which produces an encryption of $f(x_1, ..., x_N)$.
3. Now parties decrypt to learn $f(x_1, ..., x_N)$.

Now the question is: what to decrypt? which key to use? We don't want a single party or a subset of parties to be able to decrypt with their own key. The decryption should require sk_1, ..., sk_N.

The goal is: We want to build a FHE scheme where anyone can generate their own public key, and publish their own ciphertext. Anybody can take any collection of ciphertext encrypted under independently generated keys, and still is able to produce a ciphertext that somehow can be decrypted.

Syntax. We start by defining the syntax of a multi-key FHE scheme:

- Setup(1^λ) $\to$ crs, where crs is a uniform random string.
- KeyGen(crs) $\to$ (pk,sk)
- Encrypt(pk, x) $\to$ ct
- Eval(pk_1, ..., pk_N, ct_1, ..., ct_N, $\mathcal{C}$) $\to$ ct'
- Decrypt(sk_1, ..., sk_N, ct') $\to$ x

Correctness. For all x_1, ..., x_N, and all circuit $\mathcal{C}$, if you :

- crs $\leftarrow$ Setup(1^λ)
- (pk_i, sk_i) $\leftarrow$ KeyGen(crs) for all i
- $ct_i \leftarrow$ Encrypt(pk_i, x_i)
- ct' $\leftarrow$ Eval(pk_1, ..., pk_N, ct_1, ..., ct_N, $\mathcal{C}$)
- Decrypt(sk_1, ..., sk_N, ct') $\to$ f(x_1, ..., x_N)

Compactness. —ct'— = poly(λ, d, N), where d is the depth of the circuit.

Semantic security. We have an adversary and challenger, where

- Challenger generates crs and pk and set it to adversary.
- Adversary sends back x_0 and x_1.
- Challenger sends back Encrypt(src, pk, x_b), where b is a random coin.
- Adversary outputs b'. For efficient adversary $\mathcal{A}$: —Pr[b'=0 — b=1] - Pr[b'=1 — b=0]— = negl(λ)

Multi-key FHE Construction.

- The public key is
$$\mathsf{pk} = \mathbf{A} = \begin{bmatrix} \bar{\mathbf{A}} \\ \bar{\mathbf{s}}^\mathsf{T}\bar{\mathbf{A}} + \mathbf{e}^\mathsf{T} \end{bmatrix} \in \mathbb{Z}_q^{n \cdot m}$$
and the secret key is
$$\mathbf{s} = \mathbf{s}^\mathsf{T} = [\bar{\mathbf{s}}^\mathsf{T}|1] \in \mathbb{Z}_q^n,$$
where the GSW invariant is that $\mathbf{s}^\mathsf{T}\mathbf{A} = \mathbf{e}^\mathsf{T} \approx \mathbf{0}$

- The ciphertext ct : $\mathbf{C} = \mathbf{AR} + x \cdot \mathbf{G}$, where $\mathbf{R} \xleftarrow{R} \{0,1\}^{mt}$, and $t = n \log q$. The GSW decryption invariant is that $\mathbf{s}^\mathsf{T}\mathbf{C} \approx x \cdot \mathbf{s}^\mathsf{T}\mathbf{G}$.

- Suppose l ciphertext are given: $\mathbf{C}_1 = \mathbf{AR}_1 + x_1 \cdot \mathbf{G}$, ..., $\mathbf{C}_l = \mathbf{AR}_l + x_l \cdot \mathbf{G}$
$[\mathbf{C}_1|...|\mathbf{C}_l]\mathbf{H}_f = \mathbf{C}_f$, where $\mathbf{C}_f$ is the encryption of $f(x)$ because $[\mathbf{C}_1 - x_1\mathbf{G}|...|\mathbf{C}_l - x_l\mathbf{G}]\mathbf{H}_{f,x} = \mathbf{C}_f - f(x)\mathbf{G}$
Observe that $\mathbf{C}_f = [\mathbf{C}_1 - x_1\mathbf{G}|...|\mathbf{C}_l - x_l\mathbf{G}]\mathbf{H}_{f,x} + f(x)\mathbf{G} =$
$[\mathbf{AR}_1|...|\mathbf{AR}_l]\mathbf{H}_{f,x} + f(x)\mathbf{G} =$
$\mathbf{A}[\mathbf{R}_1|...|\mathbf{R}_l]\mathbf{H}_{f,x} + f(x)\mathbf{G}$.
If $[\mathbf{R}_1|...|\mathbf{R}_l]\mathbf{H}_{f,x}$ is small, then $\mathbf{C}_f$ is a GSW ciphertext.

- Suppose now we have two GSW ciphertext encrypted under different public keys but sharing the same $\bar{\mathbf{A}}$, which is the crs. The LWE secret will be different (key idea).
$pk_1 : \mathbf{A}_1 = \begin{bmatrix} \bar{\mathbf{A}} \\ s_1^T\bar{\mathbf{A}} + e_1^T \end{bmatrix} \in \mathbb{Z}_q^{n*m}$, $sk_1 : s_1^T = [\bar{s_1^T}|1] \in \mathbb{Z}_q^n$

$pk_2 : \mathbf{A}_2 = \begin{bmatrix} \bar{\mathbf{A}} \\ s_2^T\bar{\mathbf{A}} + e_2^T \end{bmatrix} \in \mathbb{Z}_q^{n*m}$, $sk_2 : s_2^T = [\bar{s_2^T}|1] \in \mathbb{Z}_q^n$

$\mathbf{C}_1 = \mathbf{A}_1\mathbf{R}_1 + x_1 \cdot \mathbf{G}$, $\mathbf{C}_2 = \mathbf{A}_2\mathbf{R}_2 + x_2 \cdot \mathbf{G}$, from where we can get: $\mathbf{C}_1 + \mathbf{C}_2 = \mathbf{A}_1\mathbf{R}_1 + \mathbf{A}_2\mathbf{R}_2 + (x_1 + x_2) \cdot \mathbf{G}$ (we don't know how to decrypt)

- **Key idea**: "Expand" ciphertext so that they are encryptions under the joint secret key $\mathbf{s}^\mathsf{T} = [s_1^T|s_2^T]$ (can be generalized to N cases). Now it become a single-key scheme.
Given a ciphertext we can expand it: $\mathbf{C} \to \hat{\mathbf{C}}$ so that $\hat{\mathbf{C}}$ is a GSW ciphertext. $\mathbf{s}^\mathsf{T}\hat{\mathbf{C}} \approx x_1 + \mathbf{s}^\mathsf{T}\hat{\mathbf{G}}$, where $\hat{\mathbf{G}} = \begin{bmatrix} \mathbf{G} & 0 \\ 0 & \mathbf{G} \end{bmatrix}$. This is challenging because an encryption algorithm Encrypt(crs, pk, x) can never produce $\hat{\mathbf{C}}$ because $\hat{\mathbf{C}}$ depends s_2, which is sampled independently form the encryption algorithm.
We solve the problem by ask Encrypt to output a **hint** that combines pk_2 to produce $\hat{\mathbf{C}}$, so that during evaluation, we can use the hint together with pk_2 together with the ciphertext to produce $\hat{\mathbf{C}}$.
We can construct $\hat{\mathbf{C}} = \begin{bmatrix} \mathbf{C}_1 & \mathbf{X} \\ 0 & \mathbf{C}_1 \end{bmatrix}$, where $\mathbf{X}$ is the hint.
We need to satisfy the invariant: $[s_1^T|s_2^T] = \begin{bmatrix} \mathbf{C}_1 & \mathbf{X} \\ 0 & \mathbf{C}_1 \end{bmatrix} \approx [\mu \cdot s_1^T\mathbf{G}|s_1^T\mathbf{X} + s_2^T\mathbf{C}]$. We would want $s_1^T\mathbf{X} + s_2^T\mathbf{C}$ to be $\mu \cdot s_2^T\mathbf{G}$ so that the whole equation will equal to $\mu[s_1^T|s_2^T]\hat{\mathbf{G}}$.

- Again our **goal** here is: $s_1^T\mathbf{X} + s_2^T\mathbf{C} \approx \mu \cdot s_2^T\mathbf{G}$, where $\mathbf{C} = \mathbf{AR} + \mu\mathbf{G}$
We have $\mathbf{A}_1 = \begin{bmatrix} \bar{\mathbf{A}} \\ b_1^T \end{bmatrix}$, where $b_1^T = \bar{s_1^T}\bar{\mathbf{A}} + e_1^T$, $\mathbf{A}_2 = \begin{bmatrix} \bar{\mathbf{A}} \\ b_2^T \end{bmatrix}$, where $b_2^T = \bar{s_2^T}\bar{\mathbf{A}} + e_2^T$.
Now, $s_2^T\mathbf{C} = [-\bar{s_2^T}|1] \begin{bmatrix} \bar{\mathbf{A}} \\ b_1^T \end{bmatrix} \mathbf{R} + \mu \cdot s_2^T\mathbf{G} = -\bar{s_2^T}\bar{\mathbf{A}}\mathbf{R} + b_1^T\mathbf{R} + \mu \cdot s_2^T\mathbf{G} \approx (b_2^T - b_1^T)\mathbf{R} + \mu \cdot s_2^T\mathbf{G}$.
So that all we need to do is ensure $s_1^T\mathbf{X} \approx (b_2^T - b_1^T)\mathbf{R}$ (b_1 and b_2 and $\mathbf{R}$ are from public keys and encryption randomness). All we need to have is that $\mathbf{X}$ looks like a GSW ciphertext and decrypt to $(b_2^T - b_1^T)\mathbf{R}$ under s_1^T.

The hint will an encryption of components of $\mathbf{R}$, which homomorphically compute ciphertext that decrypt to $(b_2^T - b_1^T)\mathbf{R}$.

- To construct the hint $\mathbf{X}$, we first construct matrix $\mathbf{T} \in \{0,1\}^m$. Our goal is that given the encryption of $\mathbf{T}_{ij}$ and a public vector $v \in \mathbb{Z}_q^m$, we want to construct $\mathbf{C}$ such that $\mathbf{s}^\mathsf{T}\mathbf{C} \approx v^T\mathbf{T}$.

  Define $\mathbf{C} = \sum_{i\in m}\sum_{j\in m} \mathbf{C}_{ij} * \mathbf{G}^{-1}(\mathbf{Z}^{ij})$, where $\mathbf{C}_{ij} = \mathsf{Encrypt}(\mathsf{pk}, \mathbf{T}_{ij})$ and $\mathbf{Z} = \begin{bmatrix} 0^{(n-1)*m} \\ v_i e_j^T \end{bmatrix}$ (e_j is the j-th basis vector).

  Observe that:

$$\begin{aligned}
\mathbf{s}^\mathsf{T}\mathbf{C} &= \sum_{i,j\in m} \mathbf{s}^\mathsf{T}\mathbf{C}_{ij}\mathbf{G}^{-1}(\mathbf{Z}^{ij}) \\
&= \mathbf{T}_{ij}\mathbf{s}^\mathsf{T}\mathbf{G}\mathbf{G}^{-1}(\mathbf{Z}^{ij}) \\
&= \mathbf{s}^\mathsf{T}\sum_{i,j\in m} \mathbf{T}_{ij}\mathbf{Z}^{ij} \\
&= \sum_{i,j\in m} \mathbf{T}_{ij}[-\mathbf{s}^\mathsf{T}|1]\begin{bmatrix} 0^{(n-1)*m} \\ v_i\mathbf{e}_j^\mathsf{T} \end{bmatrix} = \sum_{i,j\in m} v_i\mathbf{T}_{ij}\mathbf{e}_j^\mathsf{T} = \sum_{i\in m} v_i\mathbf{t}_i^\mathsf{T} = \mathbf{v}^\mathsf{T}\mathbf{T},
\end{aligned}$$

  where $\mathbf{t}_i^\mathsf{T}$ denotes the i^{th} row of $\mathbf{T}$.

Lecture 25: Homomorphic Secret Sharing

Lecturer: David Wu *Scribe: Charlotte LeMay*

25.1 Review of Multi-Key FHE

Recall Multi-Key Fully Homomorphic Encryption discussed in the last lecture:

- $\mathsf{Setup}(1^\lambda) \to \mathsf{crs}$
- $\mathsf{KeyGen}(\mathsf{crs}) \to \mathsf{pk}, \mathsf{sk}$

In our construction, Setup outputs $\mathsf{crs} = \overline{A} \xleftarrow{R} \mathbb{Z}_q^{(n-1)\times m}$, and KeyGen samples $\overline{s} \xleftarrow{R} \mathbb{Z}_q^n$, $e \leftarrow \chi^m$, and outputs

$$\mathsf{pk} = \begin{bmatrix} \overline{A} \\ \overline{s}^T\overline{A} + e^T \end{bmatrix} = A, \ \mathsf{sk} = \begin{bmatrix} -\overline{s}^T \mid 1 \end{bmatrix} = s^T.$$

Each of the N participants in the scheme will receive crs, construct their own keys $(A_i, s_i) = (\mathsf{pk}_i, \mathsf{sk}_i) \leftarrow \mathsf{KeyGen}(\mathsf{crs})$, and encrypt their messages x_i by sampling $R_i \xleftarrow{R} \{0,1\}^{m \times n \log q}$ and setting $C_i = A_i R_i + x_i \cdot G$. Each participant will publish their public key pk_i. Then for all $j \in [n] \setminus \{i\}$, participant i will use pk_j to construct X_{ij} such that $s_i^T X_{ij} + s_j^T C_i \approx x_i \cdot s_j^T G$, using the method described in the last lecture. Participant i will expand their ciphertext C_i into the following matrix:

$$\hat{C}_i = \begin{bmatrix} C_i & 0 & \cdots & 0 & \cdots & 0 \\ 0 & C_i & \cdots & 0 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots & \ddots & \vdots \\ X_{i1} & X_{i2} & \cdots & C_i & \cdots & X_{iN} \\ \vdots & \vdots & \ddots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & 0 & \cdots & C_i \end{bmatrix},$$

where the i-th row contains the X blocks. This construction ensures that

$$\begin{aligned} \left[s_1^T \middle| s_2^T \middle| \cdots \middle| s_N^T\right] \hat{C}_i &= \left[s_1^T C_i + s_i^T X_{i1} \middle| s_2^T C_i + s_i^T X_{i2} \middle| \cdots \middle| s_i^T C_i \middle| \cdots \middle| s_N^T C_i + s_i^T X_{iN}\right] \\ &\approx \left[x_i \cdot s_1^T G \middle| x_i \cdot s_2^T G \middle| \cdots \middle| x_i \cdot s_i^T \cdot G \middle| \cdots \middle| x_i \cdot s_N^T G\right] = x_i \cdot \left[s_1^T \middle| s_2^T \middle| \cdots \middle| s_N^T\right] G, \end{aligned}$$

We want each party's ciphertext to encrypt the corresponding x_i under secret key $s = \left[s_1^T \middle| s_2^T \middle| \cdots \middle| s_N^T\right]$. To achieve this, let

$$w = \begin{bmatrix} 0 \\ 0 \\ \vdots \\ 0 \\ q/2 \end{bmatrix} \in \mathbb{Z}_q^{nN}$$

and $\hat{\mathsf{ct}}_i = \hat{C}_i \cdot G^{-1}(w)$, and so that

$$s^T \hat{\mathsf{ct}}_i = s^T \hat{C}_i \cdot G^{-}1(w) \approx x_i \cdot s^T G G^{-1}(w) = x_i \cdot s^T w = q/2 \cdot x_i,$$

where the last equality holds because the last component of s is 1. Therefore $\hat{\mathsf{ct}}_i$ is an encryption of x_i, so each participant may publish $\hat{\mathsf{ct}}_i$.

The parties can then use all the published $\hat{\mathsf{ct}}_i$s to homomorphically evaluate $f(x_1, ..., x_N)$, where f is whichever function they wish to evaluate on their inputs. Suppose the outcome of this evaluation is $\hat{\mathsf{ct}}$.

Each participant must now decrypt $\hat{\mathsf{ct}}$, which would be possible with full knowledge of s:

$$s^T \hat{\mathsf{ct}} \approx f(x_1, ..., x_n) \cdot q/2.$$

This result could be rounded to find the output value. However, we cannot simply allow each participant to know s. This would leak the secret keys of all of the original messages, which would allow each participant to learn the original inputs. This entirely defeats the purpose of multi-party computation, which is supposed to allow input data to remain private.

We get around this by noting that $\hat{\mathsf{ct}}$ can be separated into blocks:

$$s^T \hat{\mathsf{ct}} = \left[s_1^T \middle| s_2^T \middle| \cdots \middle| s_N^T\right]^T \begin{bmatrix} \hat{C}'_1 \\ \hat{C}'_2 \\ \vdots \\ \hat{C}'_N \end{bmatrix} = s_1^T \hat{C}'_1 + s_2^T \hat{C}'_2 + \cdots + s_N^T \hat{C}'_N.$$

Therefore we can have each participant publish $s_i^T \hat{C}'_i$, and then each participant can add the published values to get the answer.

Slight technicality: Publishing $s_i^T \hat{C}'_i$ leaks information about the noise, so each participant will instead sample a new error e_{smudge} uniformly from some interval $[-B, B]$, and publish $s_i^T \hat{C}_i + e_{smudge}$.

Summary of protocol:

1. Someone trustworthy[1] runs $\mathsf{crs} \leftarrow \mathsf{Setup}(1^\lambda)$ and publishes crs.
2. Each participant runs $\mathsf{pk}_i, \mathsf{sk}_i \leftarrow \mathsf{KeyGen}(\mathsf{crs})$ and publishes pk_i.
3. Each participant constructs $C_i = A_i R_i + x_i \cdot G$, uses the public keys to expand this to a ciphertext $\hat{\mathsf{ct}}_i = \hat{C}_i \cdot G^{-1}(w)$, and publishes $\hat{\mathsf{ct}}_i$.
4. Each participant homomorphically evaluates f on $\hat{\mathsf{ct}}_1, ..., \hat{\mathsf{ct}}_N$ to obtain $\hat{ct}$. They then extract their particular block $\hat{C}'_i$, sample $e_{smudge} \xleftarrow{R} [-B, B]$, and publish $s_i^T \hat{C}'_i + e_{smudge}$.
5. The participants sum $s_1^T \hat{C}'_1 + \cdots + s_N^T \hat{C}'_N$ and round to obtain $f(x_1, ..., x_N)$.

25.2 Secret Sharing

Secret sharing is a scenario in which we have a particular secret s that we don't want any individual to know; instead we wish to require the voluntary participation of n participants who hold "shares" of the secret to recover s. This situation has applications for certificate authorities, nuclear launch codes, and the Coca-Cola recipe.

One generalization is t-out-of-n secret sharing, in which any subset of t share-holders can recover the secret, but no subset of $t-1$ or fewer share-holders can recover it.

We denote the sharing algorithm by $s_1, ..., s_n \leftarrow \mathsf{Share}(1^\lambda, 1^n, s)$, where the input s is the secret to be shared, and $s_1, ..., s_n$ are the shares to be distributed. Our security notion for t-out-of-n secret sharing is as follows:

Security: There exists a simulator S such that for all $T \subseteq [n]$ with $|T| < t$ and all messages s,

$$\left\{\{s_i\}_{i \in T} : (s_1, ..., s_n) \leftarrow \mathsf{Share}(1^\lambda, 1^n, s)\right\} \; apps \; \left\{S(1^\lambda, s^n, |s|, T)\right\}.$$

[1] The ghost of Abraham Lincoln, perhaps?

This is a case in which statistical indistinguishability is actually achievable!

In the n-out-of-n case, we can simply make the shares XOR to the secret:

- $\mathsf{Share}(1^\lambda, 1^n, s)$: Sample $s_1, ..., s_{n-1} \xleftarrow{R} \{0,1\}^\ell$, set $s_n \leftarrow s \oplus s_1 \oplus s_2 \oplus \cdots \oplus s_{n-1}$, and output $(s_1, ..., s_n)$.
- $\mathsf{Reconstruct}(s_1, ..., s_n)$: Output $s \leftarrow s_1 \oplus s_2 \oplus \cdots \oplus s_n$.

The security of this scheme is essentially the security of the one-time pad.

In the t-out-of-n case, we exploit some properties of polynomials. The following t-out-of-n secret sharing scheme (with secrets taken as elements of a field $\mathbb{Z}_p$, with p prime) is a disguised version of a Reed-Solomon Code, with which either you are familiar or you aren't (your humble scribe isn't), but those that were familiar were pretty excited about it:

- $\mathsf{Share}(1^\lambda, 1^n, t, s)$: Sample a polynomial $f \leftarrow \mathbb{Z}_p[x]$ with degree $t-1$ are $f(0) = s$:

$$f(x) = s + f_1 x + f_2 x^2 + \cdots + f_{t-1} x^{t-1},$$

  where $f_1, ..., f_{t-1} \xleftarrow{R} \mathbb{Z}_p$. Set $s_i \leftarrow (i, f(i))$ for $i \in [n]$, and output $(s_1, ..., s_n)$.
- $\mathsf{Reconstruct}(\{s_i\}_{i \in T} : |T| = t)$: Given the points $(i_1, y_1), ..., (i_t, y_t)$, interpolate the unique polynomial g of degree $t-1$ such that $g(i_j) = y_j$ for all $j \in T$, and output $g(0)$.

Security here follows from the fact that we cannot reconstruct a degree $t-1$ polynomial with fewer than t points.

The Reconstruct step is performed very quickly with a *Number Theoretic Transform*, which is a variant of the Fast Fourier Transform that can interpolate polynomials from points in quasi-linear time. Everyone should go look up the Number Theoretic Transform algorithm and admire its elegance.

25.3 Homomorphic Secret Sharing

As is the theme of the course, we would like to be able to perform computations on shared secrets, and this is the purpose of Homomorphic Secret Sharing (HSS). For example, we might start with x, share it among 3 participants as x_1, x_2, x_3, apply the function f separately to each share to get y_1, y_2, y_3, then recombine the shares to get $y_1 + y_2 + y_3 = f(x)$. (Specifically we are concerned with schemes of this kind that use *additive reconstruction*.)

It turns out that the multi-key FHE we have described above implies a 2-party HSS protocol. Then, separately, 2-party HSS can be used to build n-party HSS. However, our construction will not work to build n-party HSS directly from multi-key FHE (this is a homework exercise).

Construction of 2-party HSS from multi-key FHE:

- $\mathsf{Share}(1^\lambda, xin\{0,1\}^\ell)$: Run $\mathsf{crs} \leftarrow \mathsf{MK-FHE.Setup}(1^\lambda)$ and sample keys $(\mathsf{pk}_1, \mathsf{sk}_1)$ and $(\mathsf{pk}_2, \mathsf{sk}_2)$ from $\mathsf{MK-FHE.KeyGen}(\mathsf{crs})$. Sample $x_1 \xleftarrow{R} \{0,1\}^\ell$ and set $x_2 \leftarrow x \oplus x_1$. Compute $\mathsf{ct}_1 \leftarrow \mathsf{MK-FHE.Encrypt}(\mathsf{pk}_1, x_1)$ and $\mathsf{ct}_2 \leftarrow \mathsf{MK-FHE.Encrypt}(\mathsf{pk}_2, x_2)$, and output the shares $(\mathsf{pk}_1, \mathsf{pk}_2, \mathsf{ct}_1, \mathsf{ct}_2, \mathsf{sk}_1)$ and $(\mathsf{pk}_1, \mathsf{pk}_2, \mathsf{ct}_1, \mathsf{ct}_2, \mathsf{sk}_2)$.

To evaluate a function on the shares, use the homomorphic evaluation of the $\mathsf{MK-FHE}$ scheme on ct_1 and ct_2.

The current multi-key FHE scheme will give shares y_1 and y_2 such that $y_1 + y_2 = f(x) \cdot q/2 + \mathsf{error}$, and therefore $\mathsf{round}(y_1 + y_2) = f(x)$. In order to get additive reconstruction, we want to be able to split this rounded quantity up as $\mathsf{round}(y_1 + y_2) = \mathsf{round}(y_1) + \mathsf{round}(y_2)$. This is the part that won't work for $n > 2$. We will see next lecture how to make this splitting work with high probability. (Probability over what? You'll just have to wait and see.)

CS 395T: Topics in Cryptography April 25, 2022

Lecture 26: Distributed Point Functions

Lecturer: David Wu *Scribe: Nitesh Kartha*

26.1 Recap: Homomorphic Secret Sharing

First, recall **two-party homomorphic secret sharing**: two parties, P_1 and P_2 have secret shares x_1 and x_2 respectively where x_1, x_2 are secret shares of x. Each party can non-interactively evaluate functions f of x_1 and x_2 such that $f(x_1) + f(x_2) = f(x)$, possibly over modulus q.

In a prior lecture, we essentially showed how to get HSS from multi-key FHE, where shares are ciphertexts and evaluation is homomorphic evalutations on those ciphertexts and partial decryption. For more information, refer to the lecture on multi-key FHE.

However, using multi-key FHE requires rounding in order to achieve $f(x)$ which is not ideal for secret sharing. Therefore, we claim the following:

Claim 26.1 ($\mathsf{round}(x + y) = \mathsf{round}(x) + \mathsf{round}(y)$ when x, y are individually uniform over $\mathcal{Z}_p$)**.**

Suppose $t_1 + t_2 = \frac{q}{2} * f(x) + e \pmod q$ (like it would be for multi-key FHE) where t_1, t_2 are uniform and e is small. Then, $\mathsf{round}(t_1 + t_2) = \mathsf{round}(t_1) + \mathsf{round}(t_2)$ with high probability. Refer to the lecture for a proof by picture (essentially as long as the error does not result in the value being on the opposite side of the "rounding boundary", this claim holds and this will happen with high probability).

However, one big issue remains: t_1 **and** t_2 **are not uniform** but are a result of multi-key FHE which will not provide a uniform random value. Thus, we cannot apply the claim directly. But, we can add a secret share for 0.

More formally, we sample $\delta \xleftarrow{R} \mathcal{Z}_p$ and we give δ to P_1 and $-\delta$ to P_2. Then, P_1 computes $s_1^T \hat{C} + \delta$ and P_2 computes $s_1^T \hat{C} + (-\delta)$. Therefore, the sum of the two values is still $\frac{q}{2} * f(x) + e$ BUT each share is uniform over $\mathcal{Z}_p$. Thus, we have shown that rounding commutes with addition for two-parties as well as two-party HSS but as a homework exercise, you can show that this claim about rounding does not hold for three parties.

26.2 Function Secret Sharing

Now, we will focus on a particular case where we will get concrete efficiency. First, we will talk about a dual notion of HSS: **function secret sharing (FSS)**. Informally FSS takes a description of a function f and split this into two function shares f_1 and f_2 which can compute $f_1(x)$ and $f_2(x)$ such that $f_1(x) + f_2(x) = f(x)$ for a common input x. You can go from HSS to FSS and vice versa through a universal circuit.

We will show how to use FSS to get private database queries. First, here's an example: imagine that you have a database that is replicated across two database servers and a client wants to perform some query on both databases (i.e. select "top 10 restaurants" where "category = Mediterranean", compute "average price" where data = "tomorrow" and from = "Austin and to = "NYC", etc.). Our goal is for the client to send this query but to *hide sensitive attributes* (i.e. the parameters of the queries like "Mediterranean" or "Austin or "NYC", etc.) Note that this scheme should work as long as one of the database servers is not corrupted.

We will describe an approach to handling simple statistical queries using FSS. For example, consider the following query: $\mathsf{COUNT}(\mathsf{column})$ where $x_1 = v_1 ... x_n = v_n$ where we wish to hide $v_1 .... v_n$. We can define a predicate $f(y_1, ..., y_n) = 1$ if $y_1 = v_1, ..., y_n = v_n$ and 0 otherwise. Using the additive property of FSS, we can break the predicate into secret shares f_1, f_2 and the client one share to each server which will sum over all the function evaluations of all entries of the database and send that back the client (i.e. $\sum f_1(x_1, ..., x_n)$) and then the client can sum the responses to obtain $\sum f(x_1, ..., x_n)$ which is equivalent to the result of the count query.

26.3 Distributed Point Functions

Therefore, the key primitive that we need is a way to FSS for point functions (a function where it returns 1 at a single point and 0 everywhere else, like the predicate we described above). Although we can do this using multi-key FHE, this function is simple enough to have a more optimized way of doing FSS (in a two party setting). This primitive is known as a **distributed point function (DPF)**.

The simplest construction is known as the $\sqrt{N}$ construction where N is the size of the domain of f and the keys are the size $\sqrt{N}$ (which is inefficient but can be used to generalize a $\log(N)$ construction.

Construction 26.2 ($\sqrt{N}$ DPF Construction)**.**

1. Let $l = \sqrt{N}$. Represent a domain element as (i, j) where $i, j \in [l]$ (essentially arranging the domain in a $l \times l$ grid).
2. Suppose we want to share $f_{i*,j*}$ where $i*, j*$ is the target point. First, we will sample l seeds $s_1, ..., s_l$ and using a PRG which goes from $\{0,1\}^\lambda \to \{0,1\}^l$, we will get l bit-strings of length l. More simply, we will obtain N random bits compressed into the l seeds of length λ which is much much smaller than l. The first share of $f_{i*,j*}$ will be the seeds $s_1, ..., s_l$. However, note that this share does not depend on $i*, j*$ which is why we will need a second share.
3. For the second share, everything is the same for seeds s_i^2 where $i \neq i*$. For when $i = i*$, we will have a random seed $s_{i*}^2 \xleftarrow{R} \{0,1\}^\lambda$. We will also publish $w := \mathsf{PRG}(s_{i*}) \oplus \mathsf{PRG}(s_{i*}^2) \oplus e_{j*}$ and provide that to each share. Note that each party cannot determine any information about $i*, j*$ from w with just their share since it is binded by a PRG evaluation of a seed that is not known to them.
4. In addition, you will secret share l bits. $b_1 ... b_l$ is given to the first server where $b_i \xleftarrow{R} \{0,1\}$ and the second server will be given the same b_i when $i \neq i*$ for b_i^2 but then flip the bit for b_{i*}^2.
5. Evaluation: $\mathsf{PRG}(s_i) \oplus b_i * w$ for both parties.

When $i \neq i*$, the first server will have $\mathsf{PRG}(s_i) \oplus b_i * w$ and the second server will also have $\mathsf{PRG}(s_i) \oplus b_i * w$ (since $s_i = s_i^2$ and $b_i = b_i^2$ when $i \neq i*$) which xors to 0. When $i = i*$, the first server will have $\mathsf{PRG}(s_{i*}) \oplus b_{i*} * w$ and the second server will have $\mathsf{PRG}(s_{i*}^2) \oplus b_{i*}^2 * w$ which when xor-ed will get $\mathsf{PRG}(s_{i*}) \oplus \mathsf{PRG}(s_{i*}^2) \oplus w$ which is equivalent to e_{j*}.

The size of the share is $(\lambda + 2) * l$ or $(\lambda + 2) * \sqrt{N}$. If we were to try to go to key sizes beyond $\sqrt{N}$ (i.e. $\log(n)$), you can naively do recursive composition since the shares themselves are point functions since they only differ at one point which will, but this will get you to poly-logarithmic key sizes. However, you can view each share as a tree that differs only in the $i*$ branch. Note that for every other branch, the state of the tree is the same for both shares. On $i*$'s branch, we can program the value to whatever we want using a value like w above.

Using this idea, you can make the root of the tree a different PRG seed for each share which will then expand to the child-nodes as well as a factor b such that $b \oplus b' = 1$ where b' is for the other share. The correction word w is chosen such that $0^\lambda || s_1 \oplus s_1' || 1 \oplus b_0 \oplus b_0' || b_1 \oplus b_1'$ which is then blinded by $G(s) \oplus G(s')$. This results in a correction word for each level of the tree is $2\lambda + 2$ and there are $\log(N)$ levels, resulting in a share size of $O(\log N)$.

Open questions. Finally, to conclude, we will pose some open questions:

- **Can we do a similar construction (i.e. concretely efficient) for more than 2 parties?** The current construction describe does not work for more than 2 parties since the other parties can collude to figure out what $i*$ is as well as the requirement of more than 2 branches of the tree which can lead to different values and computations when generalized to $\log(N)$. The best construction for k parties from one-way functions has a share size of $2^k * \sqrt{N} * poly(\lambda)$ which is exponential.

- **Can we do better with k parties if we require an honest majority compared to only one honest host?** In the constructions above, we only require one party to be honest. It is unknown if relaxing this condition to being a honest majority will lead to a better k party construction.
- **What is the most expressive functions that we can effectively secret share without having to use multi-key FHE?** This construction was dealing with point functions (which can be expanded to interval functions) but is it possible to effectively secret share even more complex functions? This would allow us to support more expressive queries in the private database query example mentioned above.

CS 395T: Topics in Cryptography May 2, 2022

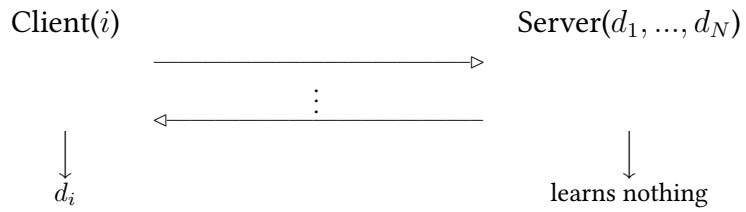
Lecture 27: Private Information Retrieval

Lecturer: David Wu *Scribe: Aya Abdelgawad*

27.1 Premise

Private information retrieval (PIR) is the process by which clients can access records from a server without revealing exactly which record they want. The properties we want to achieve are

- **Correctness:** the client learns the desired database record d_i.
- **Security:** the (potentially malicious) server learns nothing about i.



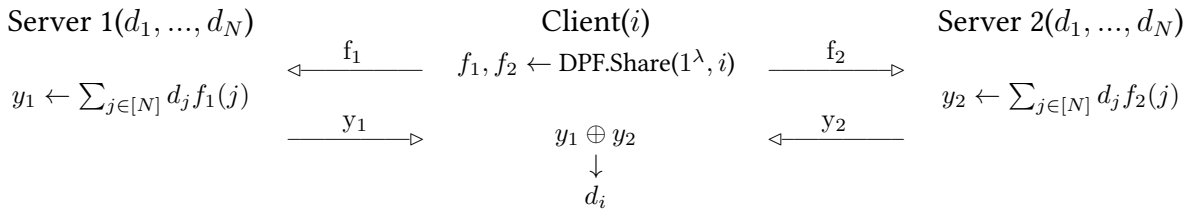
Remark 27.1. We do *not* require privacy for the server's database; otherwise, this would be oblivious transfer (OT). Thus, there is a trivial PIR solution for the client to download the full database ($O(N)$ response size). However, our goal is to minimize the size of the server's response to the client, so we want to build more efficient constructions. If we can do this, we can use PIR as the basic building block for privacy-preserving protocols, with potential applications in

- private DNS lookups
- safe browsing
- private contact tracing
- contact discovery
- anonymous messaging

Remark 27.2. In order to maintain the security property, the server must do work linear in the size of the database (if it does sublinear work, there are some records that will be ignored, revealing those were not d_i). A current open question is if it is possible to do some linear preprocessing to encode the database so that the protocol takes sublinear time per request.

27.2 Potential Constructions

Construction 27.3. Secret-Sharing in the Multi-Server Setting
In this setting, we assume that we have a two non-colluding servers, each with a copy of the database. Then we can use homomorphic secret-sharing on a distributed point function to obtain $f_1, f_2 \leftarrow$DPF.Share($1^\lambda, i$). Since f_1, f_2 perfectly hide i, the client can send each server a share.



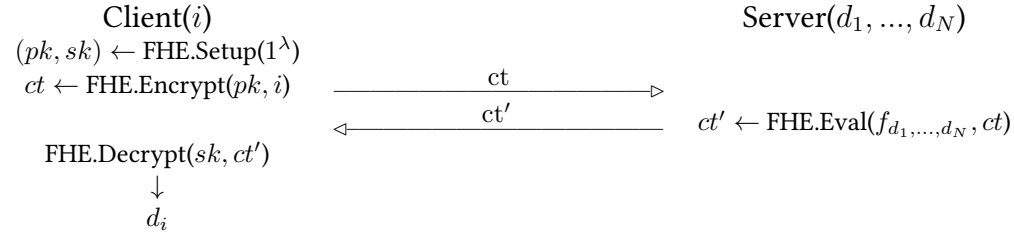
For this construction, the communication sizes are almost optimal with respect to N, with a

- query of size $O(\log N) \cdot \mathsf{poly}(\lambda)$
- response of size $2|d_i|$

While the server must do linear work with respect to the size of the database, it is possible to amortize with some preprocessing.

Remark 27.4. It is possible to achieve information-theoretic constructions using locally decodable codes, but with the cost that communication sizes blow up to $O(N^\epsilon)$ with $0 < \epsilon < 1$.

Construction 27.5. FHE in Single-Server Setting
In this setting, we have only one server hosting the database. Let $f_{d_1,\dots,d_n} = \sum_{j\in[N]} d_j \cdot 1\{i = j\}$. Then we can use the following as a potential PIR construction:



The communication sizes are

- query: $O(\log N) \cdot \mathsf{poly}(\lambda)$
- response: $|d_i| \cdot \mathsf{poly}(\lambda, \log N)$ (note we can remove the $\log N$ factor by bootstrapping the FHE)

This construction is unlikely to achieve concrete efficiency, though, because it requires doing $O(\log N)$ multiplications per database element. With N usually being around $2^{20} - 2^{30}$, so the multiplicative depth is high, leading to poor concrete efficiency.

Construction 27.6. Kushilevitz-Ostrovsky in the Single-Server Setting
We can achieve better efficiency results using the Kushilevitz-Ostrovsky framework, getting communication down to $O(\sqrt{N})$ by interpreting the database as a grid multiplying it by FHE encryption of $\vec{e_i}$ (where i in this case represents the column of the database we are interested in) to get an encryption of the relevant column.

$$\begin{bmatrix} d_{11} & d_{12} & \cdots & d_{1\ell} \\ d_{21} & d_{22} & \cdots & d_{2\ell} \\ \vdots & \vdots & \ddots & \vdots \\ d_{\ell 1} & d_{\ell 2} & \cdots & d_{\ell\ell} \end{bmatrix} \times \text{FHE.Encrypt}\left(\begin{bmatrix} 0 \\ 1 \\ \vdots \\ 0 \end{bmatrix} \right) \to \text{FHE.Encrypt}\left(\begin{bmatrix} d_{1i} \\ d_{2i} \\ \vdots \\ d_{\ell i} \end{bmatrix} \right)$$

In this construction, additively homomorphic constructions are sufficient for computing the output. Note that both the query and response have size $O(\sqrt{N})$, but it is possible to decrease this by doing recursive computation or homomorphic multiplication (both approaches requiring fully homomorphic schemes). This is currently what concretely-efficient algorithms use today, but an open question is if we can further reduce the computational complexity by encoding the database in some way. Note the server still has to do work linear in the database size, but it is possible to amortize to $O(\sqrt{N})$ time with preprocessing.

27.3 Improving Efficiency of Lattice-Based Schemes with Rings

27.3.1 Ring-LWE

Recall that in the standard Regev encryption scheme, we have

$$pk = A = \begin{bmatrix} \overline{A} \\ \overline{s}^T\overline{A} + e^T \end{bmatrix} \in \mathbb{Z}_q^{n\times m}$$

a large public key, typically of size $O(n^2 \log q)$ (quadratic in the lattice dimension)

$$sk = s^T = [-\overline{s}^T | 1] \in \mathbb{Z}_q^n$$

$$ct = Ar + \begin{bmatrix} 0^{n-1} \\ \mu \cdot \left\lfloor \frac{q}{p} \right\rceil \end{bmatrix} \in \mathbb{Z}_q^n$$

need n elements of $\mathbb{Z}_q$ to encrypt 1 $\mathbb{Z}_p$ element (large blowup)

To improve efficiency, we can work over polynomial rings instead. The notation for this is

- $\mathbb{Z}[x]$: ring of polynomials with integer coefficients
- $\mathbb{Z}[x]/(x^{2^d}+1)$: ring of polynomials with integer coefficients, modulo $x^{2^d}+1$ (a cyclotomic polynomial)
  - Note that $-1 \cong x^{2^d} \mod x^{2^d}+1$

We can think of LWE as working over the ring $R = \mathbb{Z}$. Now, we consider the ring $R = \mathbb{Z}/(x^{2^d}+1)$ for Ring-LWE (RLWE). The RLWE assumption is that given

$$\begin{aligned} a &\xleftarrow{R} R_q \\ s &\xleftarrow{R} R_q \\ e &\leftarrow \chi \\ u &\xleftarrow{R} R_q \end{aligned}$$

(where χ is an analog of dicrete Gaussian in R_q), then the following distributions are computationally indistinguishable:

$$(a, sa+e) \quad \text{and} \quad (a, u).$$

We can use this assumption to build a Regev encryption scheme over rings.

$$\begin{aligned} sk &= s && \text{where} \quad s \xleftarrow{R} R_q \\ pk &= (a,b) && a \xleftarrow{R} R_q \\ & && e \leftarrow \chi \\ & && b \leftarrow sa + e \\ ct &= (ar, br + \mu \left\lfloor \tfrac{q}{2} \right\rceil) && r \xleftarrow{R} R_q \text{ and } \mu \in R_q \end{aligned}$$

With this scheme we get shorter public keys, faster key-generation, and less blowup in ciphertext (can encrypt 1 R_p element with 2 R_q elements). Another benefit comes from viewing view ring multiplication as matrix-vector multiplication. For example, if we are working over the ring $R = \mathbb{Z}[x]/(x^4+1)$ and have

$$a = 2x^3 + x^2 - 3x + 1, \quad s = x^3 - 2x + 2$$

then

$$as = (2x^3+x^2-3x+1)(x^3-2x+2) = \begin{bmatrix} 1 & -3 & 1 & 2 \\ -2 & 1 & -3 & 1 \\ -1 & -2 & 1 & -3 \\ 3 & -1 & -2 & 1 \end{bmatrix} \begin{bmatrix} 1 \\ 0 \\ -2 \\ 2 \end{bmatrix} = \begin{bmatrix} 3 \\ 6 \\ -9 \\ 1 \end{bmatrix} \begin{matrix} x^3 \\ x^2 \\ x \\ 1 \end{matrix}$$

Note that the right vector is uniform random, and so too is the first row of the left matrix. However, the other rows are related to that first one, so we are doing LWE on a structured matrix. This gives us the benefit that (with a suitable q) we can compute as in $O(d \log d)$ time using Fast Fourier Transform (much faster than matrix multiplication in normal LWE). However, this structured setting prevents us from reducing RLWE hardness to worst case lattice problems.

27.3.2 General Efficiency Improvements from Rings

Most constructions based on standard lattices can be directly translated into the ring setting with better concrete efficiency.

Exploiting Structure in the Ring Setting

Suppose we work over the ring $R = \mathbb{Z}[x]/(x^{2^d}+1)$ and suppose we choose the plaintext modulus $p = 1 \mod 2^{d+1}$. We can show that the polynomial $x^{2^d}+1$ factors mod p as

$$x^{2^d}+1 = \prod_{i\in[2^d]} (x-\alpha_i) \mod p$$

for $a_1, ..., a_{2^d} \in \mathbb{Z}_p$. Then, by the Chinese Remainder Theorem (CRT):

$$\begin{aligned} R/pR = \mathbb{Z}_p[x]/(x^{2^d}+1) &\cong \mathbb{Z}_p[x]/(x-\alpha_1) \times ... \times \mathbb{Z}_p[x]/(x-\alpha_{2^d}) \\ &\cong \mathbb{Z}_p^{2^d} \end{aligned}$$

Thus, the plaintext space is isomorphic to $\mathbb{Z}_p^{2^d}$. This means that additions and multiplications in R_p correspond to component-wise additions and multiplications in $\mathbb{Z}_p^{2^d}$, respectively. Single instruction multiple data (SIMD) gives us support for homomorphic evaluation, so if we encrypt a vector of 2^d integers, we can apply each homomorphic operation simultaneously on *all* 2^d elements.

Reducing Ciphertext Size via Modulus Switching

When we use FHE to evaluate a circuit C, we need to choose parameters so that the accumulated error is smaller than $\frac{q}{2p}$. One useful technique is to perform all computations with respect a modulus q and then rescale the final ciphertext to a smaller modulus $q' < q$:

1. Recall that $s^T c = \left\lfloor \frac{q}{p} \cdot \mu \right\rceil + e \mod q$. We replace $c \mapsto \left\lfloor \frac{q'}{q} \cdot c \right\rceil$ (and interpret c' and element of $\mathbb{Z}_{q'}$).
2. To analyze this, we consider an expression over the rationals:

   $$s^T c = \left\lfloor \frac{p}{q} \cdot \mu \right\rceil + e + kq \quad \text{for some integer } q$$

   We can write

   $$c' = \left\lfloor \frac{q'}{q} \cdot c \right\rceil = \frac{q'}{q} \cdot c + e' \quad \text{where } ||e'|| < 1/2 \text{ (over the rationals)}$$

3. Then,

   $$\begin{aligned} s^T c' &= \frac{q'}{q} \cdot s^T c + s^T e' \\ &= \frac{q'}{q}\left[\left(\frac{q}{p} \cdot \mu + e''\right) + e + kq \right] + s^T e' \\ &= \frac{q'}{p}\mu + \frac{q'}{q}(e''+e) + kq' + s^T e' \\ &\cong \frac{q'}{p}\mu + \frac{q'}{q}(e''+e) + s^T e' \mod q' \end{aligned}$$

   We require that the components of $\frac{q'}{q}(e''+e)+s^T e'$ be smaller than $\frac{q'}{2p}$. In particular, the secret key components need to be small (i.e., sampled from an error distribution). In addition, observe that the error is scaled down by $\frac{q'}{q}$.

Using this technique, we can rescale the ciphertexts to a smaller modulus q', resulting in concrete reductions in the size of communications.

CS 395T: Topics in Cryptography May 4, 2022

Lecture 28: Conclusion

Lecturer: David Wu *Scribe: Soham Roy*

28.1 Notes on RLWE

For the RLWE (ring LWE) problem, instead of operating over $\mathbb{Z}$ as in LWE, work over a polynomial ring $\mathbb{Z}[x] / \left(x^{2^d} + 1\right)$. This can also be viewed as LWE with a structured matrix. $(a, s \cdot a + e) \cong (a, u)$, where a and s are polynomials.

Regev encryption over polynomial ring looks like $pk = (a, b)$ where $a \xleftarrow{R} R_q$, $s \xleftarrow{R} R_q$, $e \xleftarrow{R} \chi$, and $b = sa + e$. Ciphertext $\text{ct} = \left(ar, b \cdot r + \mu \lfloor \frac{q}{2} \rceil\right)$, and $r \leftarrow \chi$. To encrypt a polynomial $\mu \in R_2$ (μ is degree 2^d), $|\text{ct}| = 2|R_q|$. $\frac{|\mu|}{|\text{ct}|} = \frac{|R_2|}{2|R_q|} = \frac{1}{2 \log q}$. This is a significantly smaller ciphertext than LWE, both theoretically and practically.

Another reason to use polynomial rings is for the additional algebraic structure. Polynomial multiplication is commutative, unlike with matrices. As far as we know, the commutative property does not compromise the RLWE assumption. Unfortunately, RLWE doesn't have worst-case reductions (only to a specific ring).

Suppose the plaintext space is $R = \mathbb{Z}[x] / \left(x^{2^d+1}\right)$ and the plaintext modulus $p \equiv 1 \pmod{2^{d+1}}$. The following comes out of Galois theory:

$$x^{2^d} + 1 = \prod_{i \in [2^d]} (x - \alpha_i) \text{ where } \alpha_i \in \mathbb{Z}_p$$

This allows for the use of the Chinese Remainder Theorem. Using RNS (residual number system), operations can be performed over a CRT decomposition of the ring modulus to better use registers that are smaller than necessary for big integer representations.

$$R_p \cong \mathbb{Z}_p[x] / (x - \alpha_1) \times \ldots \times \mathbb{Z}_p[x] / (x - \alpha_{2^d}) \cong \mathbb{Z}_p^{2^d}$$

By choosing the plaintext modulus wisely, operations can be carried out over a vector space instead of a polynomial. This technique is called "batching" in FHE or SIMD (single instruction multiple data) and can provide significant time savings. FHE can be done with polylog overhead.

28.2 Course summary

Advanced crypto $\rightarrow$ securing (confidentiality and integrity) computation

Homomorphic encryption - computing on encrypted data; Long believed to be impossible
Homomorphic signatures - computing on signed data
Zero knowledge for NP - integrity for computations
Functional encryption - fine-grained access to encrypted data

These capabilities are powered by lattice based cryptography. Lattice based cryptography comes from the study of the SIS and LWE assumptions.
SIS: $A \in \mathbb{Z}_q^{n \times m}$, find $\vec{x}$ s.t. $A\vec{x} = \vec{0}$ and $\|\vec{x}\| \leq \beta$
LWE: $A \xleftarrow{R} \mathbb{Z}_q^{n \times m}$, $s \xleftarrow{R} \mathbb{Z}_q^n$, $e \xleftarrow{R} \chi^m$, $(A, s^T A + e^T) \cong (A, b)$ where $b \xleftarrow{R} \mathbb{Z}_q^n$
Hardness of LWE implies hardness of SIS and vice versa.
Both SIS and LWE have ring based analogues: RSIS and RLWE respectively.

Two major equations that underpin lattice based cryptography.

Given $A_1, ..., A_l \in \mathbb{Z}_q^{n \times m}$ and $f : \{0,1\}^l \to \{0,1\}$.

$$\exists H_f \text{ where } \|H_f\| = (n \log q)^{O(d)} \text{ s.t. } [A_1|...|A_l] \cdot H_f = A_f$$

$$\exists H_{f,x} \text{ where } \|H_{f,x}\| = (n \log q)^{O(d)} \text{ s.t. } [A_1 - x_1 G|...|A_l - x_l G] \cdot H_{f,x} = A_f - f(x) \cdot G$$

Some of the uses of these two equations:

	H_f	$H_{f,x}$
FHE	homomorphic evaluation	correctness analysis
HS	verification	homomorphic evaluation on signature
ABE	key-generation	decryption

Bibliography

[Ajt98] Miklós Ajtai. The shortest vector problem in L_2 is *NP*-hard for randomized reductions (extended abstract). In *STOC*, pages 10–19, 1998.

[ALNS20] Divesh Aggarwal, Jianwei Li, Phong Q. Nguyen, and Noah Stephens-Davidowitz. Slide reduction, revisited - filling the gaps in SVP approximation. In *CRYPTO*, pages 274–295, 2020.

[ALS21] Divesh Aggarwal, Zeyong Li, and Noah Stephens-Davidowitz. A $2^{n/2}$-time algorithm for $\sqrt{n}$-svp and $\sqrt{n}$-hermite svp, and an improved time-approximation tradeoff for (H)SVP. In *EUROCRYPT*, pages 467–497, 2021.

[AR04] Dorit Aharonov and Oded Regev. Lattice problems in NP cap conp. In *FOCS*, pages 362–371, 2004.

[Din00] Irit Dinur. Approximating SVP_∞ to within almost-polynomial factors is np-hard. In *CIAC*, pages 263–276, 2000.

[GG00] Oded Goldreich and Shafi Goldwasser. On the limits of nonapproximability of lattice problems. *J. Comput. Syst. Sci.*, 60(3):540–563, 2000.

[HR07] Ishay Haviv and Oded Regev. Tensor-based hardness of the shortest vector problem to within almost polynomial factors. In *STOC*, pages 469–477, 2007.

[Kho04] Subhash Khot. Hardness of approximating the shortest vector problem in lattices. In *FOCS*, pages 126–135, 2004.

[LLL82] Arjen K Lenstra, Hendrik Willem Lenstra, and László Lovász. Factoring polynomials with rational coefficients. *Mathematische annalen*, 261:515–534, 1982.

[Mic98] Daniele Micciancio. The shortest vector in a lattice is hard to approximate to within some constant. In *FOCS*, pages 92–98, 1998.

[Pei08] Chris Peikert. Limits on the hardness of lattice problems in l_p norms. *Comput. Complex.*, 17(2):300–351, 2008.

[Sch87] Claus-Peter Schnorr. A hierarchy of polynomial time lattice basis reduction algorithms. *Theor. Comput. Sci.*, 53:201–224, 1987.

[WLW15] Wei Wei, Mingjie Liu, and Xiaoyun Wang. Finding shortest lattice vectors in the presence of gaps. In *CT-RSA*, pages 239–257, 2015.