CS 6501 Week 13: Advanced Lattice-Based Primitives

So far, we have shown how to build symmetric crypto and public-key crypto from standard lattice assumptions (e.g., SIS and LWE)

But it turns out, lattices have much additional structure $\Rightarrow$ enable many new advanced functionalities not known to follow from many other standard assumptions (e.g., discre log, factoring, pairings, etc.)

We will begin by studying *fully homomorphic encryption* (FHE)
↳ encryption scheme that supports arbitrary computation on encrypted data [very useful for outsourced computation]

Abstractly: given encryption ct_x of value x under some public key, can we derive from that an encryption of $f(x)$ for an arbitrary function f?
- So far, we have seen examples of encryption schemes that support one type of operation (e.g., addition) on ciphertexts
  - ElGamal encryption (in the exponent): homomorphic with respect to addition [HW3]
  - Regev encryption: homomorphic with respect to addition
- For FHE, need homomorphism with respect to two operations: addition and multiplication
  Major open problem in cryptography (dates back to late 1970s!) — first solved by Stanford student Craig Gentry in 2009
  ↳ revolutionized lattice-based cryptography!
  ↳ very surprising this is possible: encryption needs to "scramble" messages to be secure, but homomorphism requires preserving structure to enable arbitrary computation

General blueprint: 1. Build somewhat homomorphic encryption (SWHE) — encryption scheme that supports bounded number of homomorphic operations
2. Bootstrap SWHE to FHE (essentially a way to "refresh" ciphertext)

Focus will be on building SWHE (has all of the ingredients for realizing FHE)
↳ In particular, will present Gentry-Sahai-Waters (GSW) construction (conceptually simplest scheme, though not the most concretely efficient)
"3rd generation of FHE"

Starting point: Regev's encryption scheme:

Setup(1^λ): $\tilde{A} \xleftarrow{R} \mathbb{Z}_q^{n \times m}$ $\quad A = \begin{bmatrix} \tilde{A} \\ \tilde{s}^T \tilde{A} + e^T \end{bmatrix} \in \mathbb{Z}_q^{(n+1)\times m}$

$\tilde{s} \xleftarrow{R} \mathbb{Z}_q^n$

$e \leftarrow \chi^m$ $\quad s = \begin{bmatrix} -\tilde{s} \\ 1 \end{bmatrix} \in \mathbb{Z}_q^{n+1}$

Output $pk = A$ and $sk = s$

Observation: $s^T A = -\tilde{s}^T \tilde{A} + \tilde{s}^T \tilde{A} + e^T = e^T \approx 0^m$

Encrypt(pk, μ): Write $pk = A \in \mathbb{Z}_q^{(n+1)\times m}$ and sample $R \xleftarrow{R} \{0,1\}^{m \times m}$

$$C = AR + \mu \cdot \lfloor \tfrac{q}{2} \rfloor \cdot I_{(n+1)\times m}$$

$I_{(n+1)\times m} = \left(\underbrace{\begin{matrix} 1 & & 0 \\ & \ddots & \\ 0 & & 1 \end{matrix}}_{\substack{(n+1)\times(n+1) \\ \text{identity matrix}}} \middle| \; 0 \right)$

Decrypt(sk, c): Write $sk = s$. Compute $s^T C$ and output 0 if $|(s^T C)_{n+1}| < \frac{q}{4}$ and 1 if $|(s^T C)_{n+1}| > \frac{q}{4}$
$(n+1)^{st}$ component of $s^T C$, interpreted as value between $-\frac{q}{2}$ and $\frac{q}{2}$

Correctness: $s^T C = s^T AR + x \cdot \lfloor \frac{q}{2} \rfloor \cdot s^T I_{(n+1)\times m}$
$= e^T R + x \cdot \lfloor \frac{q}{2} \rfloor \cdot s^T$
$\approx x \cdot \lfloor \frac{q}{2} \rfloor \cdot s^T$

Observe: the vector s (i.e., the secret key) is an approximate left-eigenvector of the matrix C (i.e., the ciphertext) with associated eigenvalue $x \cdot \lfloor \frac{q}{2} \rfloor$ (i.e., the "encoded" message)

Security: Same as proof for Regev encryption (two hybrids: LWE, then LHL)

Observe: We can pad A with rows of all-zeroes so it is a square matrix (over $\mathbb{Z}_q^{m\times m}$) and pad s accordingly as well

For the ciphertext, we just embed the message in the first $(n+1)$ components

Then, correctness and security follow as before (scheme has not changed), and the message is simply the "noisy" eigenvalue associated with s (the "noisy" eigenvector)

Why is this view useful? Because eigenvalues add and multiply:

- Suppose x_1 is a (left) eigenvalue of C_1 with associated eigenvector s
- Suppose x_2 is a (left) eigenvalue of C_2 with associated eigenvector s

Then: $s^T(C_1+C_2) = s^TC_1 + s^TC_2 = x_1 s^T + x_2 s^T = (x_1+x_2)s^T$

$s^T C_1 C_2 = x_1 \cdot s^T C_2 = x_1 x_2 s^T$

Enables homomorphic operations!

Unfortunately, this intuition does not directly translate to our setting:

Correctness: $s^TC = x\cdot \lfloor \frac{q}{2} \rfloor \cdot s^T + e^TR$

Addition: $s^T(C_1+C_2) = x_1 \cdot \lfloor \frac{q}{2} \rfloor \cdot s^T + e^TR_1 + x_2 \cdot \lfloor \frac{q}{2} \rfloor \cdot s^T + e^TR_2$

$= (x_1+x_2)\cdot \lfloor \frac{q}{2} \rfloor \cdot s^T + e^T(R_1+R_2)$ Works as long as R_1+R_2 is small! (As long as $B \ll q$, this will be OK.)

Multiplication: $s^TC_1C_2 = (x_1 \cdot \lfloor \frac{q}{2} \rfloor \cdot s^T + e^TR_1)C_2 = x_1 \cdot \lfloor \frac{q}{2} \rfloor \cdot s^TC_2 + e^TR_1C_2$

$$= \underbrace{x_1 \cdot \lfloor \tfrac{q}{2} \rfloor \cdot \left(x_2 \cdot \lfloor \tfrac{q}{2} \rfloor + e^TR_2\right)}_{\text{not quite what we wanted due to the message encoding, but should be fixable...}} + \underbrace{e^TR_1C_2}_{\text{this is large since } C_2 \text{ is not short!}}$$

↳ Correctness fails for multiplication!

Need a new trick: the gadget matrix G (e.g., the powers-of-two matrix)

- One issue above is noise growth (and bit-decomposition is effective way of generating short matrices).

The GSW FHE scheme:

Setup(1^λ): $\tilde{A} \xleftarrow{R} \mathbb{Z}_q^{n\times m}$

$\tilde{s} \xleftarrow{R} \mathbb{Z}_q^n$

$e \leftarrow \chi^m$

$A = \begin{bmatrix} \tilde{A} \\ \tilde{s}^T\tilde{A} + e^T \end{bmatrix} \in \mathbb{Z}_q^{(n+1)\times m}$

$s = \begin{bmatrix} -\tilde{s} \\ 1 \end{bmatrix} \in \mathbb{Z}_q^{n+1}$

Same as Regev's encryption scheme!

output $pk = A$ and $sk = s$

Encrypt(pk, μ): Write $pk = A \in \mathbb{Z}_q^{(n+1)\times m}$ and sample $R \xleftarrow{R} \{0,1\}^{m\times m}$

Output $C = AR + \mu \cdot G$

Decrypt(sk, C): Write $sk = s$. Compute s^TC and output 0 if $|(s^TC)_m| < \frac{q}{4}$ and 1 if $|(s^TC)_m| > \frac{q}{4}$

Correctness: $s^TC = s^TAR + \mu \cdot s^TG = \mu \cdot s^TG + e^TR$

By construction of G, $[s^TG]_m = 2^{\lfloor \log q \rfloor}$, so if $\|e^TR\| < \frac{q}{4}$, then correctness goes through

GSW invariant: let $C = AR + \mu \cdot G$ for some $\mu \in \{0,1\}$. Then,

$$s^T C = s^T(AR + \mu \cdot G) = -e^T R + \mu \cdot s^T G$$

last components of $s^T G$ is large $(\sim 2^{\lfloor \log q \rfloor})$ if $x = 1$ and small if $x = 0$

Homomorphic addition: $s^T(C_1 + C_2) = s^T(AR_1 + \mu_1 G) + s^T(AR_2 + \mu_2 \cdot G) = \underbrace{e^T(R_1 + R_2)}_{\text{new error in ciphertext also adds}} + (\mu_1 + \mu_2) \cdot s^T G$

Homomorphic multiplication:

$$\begin{aligned} s^T(C_1 G^{-1}(C_2)) &= s^T(AR_1 + \mu_1 G) \cdot G^{-1}(C_2) = s^T(AR_1 G^{-1}(C_2) + \mu_1 \cdot C_2) \\ &= s^T(AR_1 G^{-1}(C_2) + \mu_1 AR_2 + \mu_1\mu_2 \cdot G) \\ &= e^T \underbrace{(R_1 G^{-1}(C_2) + \mu_1 R_2)} + \mu_1\mu_2 \cdot s^T G \end{aligned}$$

new error only increases modestly since $x_1 \in \{0,1\}$ and $G^{-1}(C_2)$ is short: if $\|R_1\|_\infty, \|R_2\|_\infty \leq B$, then

$$\|R_1 G^{-1}(C_2) + x_1 R_2\|_\infty \leq B(m+1)$$

Conclusion: If we want to support circuits of multiplicative depth d, we need to choose $q = m^{O(d)}$ to accomodate the multiplications. Observe that in this case, $\log q = O(d \log m)$, so the number of bits in the ciphertext scales linearly with the depth of the circuit. [Note: generally, there is a lot of flexibility when choosing lattice parameters]

Semantic security follows by same argument as Regev. Homomorphic operations possible by structure of gadget matrix!

From SWHE to FHE. The above construction requires imposing an a priori bound on the multiplicative depth of the computation. To obtain fully homomorphic encryption, we apply Gentry's brilliant insight of bootstrapping.

High-level idea. Suppose we have SWHE with following properties:

1. We can evaluate functions with multiplicative depth d
2. The decryption function can be implemented by a circuit with multiplicative depth $d' < d$

Then, we can build an FHE scheme as follows:

- Public key of FHE scheme is public key of SWHE scheme and an encryption of the SWHE decryption key under the SWHE public key
- We now describe a ciphertext-refreshing procedure:
  - For each SWHE ciphertext, we can associate a "noise" level that keeps track of how many more homomorphic operations can be performed on the ciphertext (while maintaining correctness).
    - ↳ for instance, we can evaluate depth-d circuits on fresh ciphertexts; after evaluating a single multiplication, we can only evaluate circuits of depth-$(d-1)$ and so on...
  - The refresh procedure takes any valid ciphertext and produces one that supports depth-$(d-d')$ homomorphism; since $d > d'$, this enables unbounded (i.e., arbitrary) computations on ciphertexts

Idea: Suppose $ct_x = \text{Encrypt}(pk, x)$.

Using the SWHE, we can compute $ct_{f(x)} = \text{Encrypt}(pk, f(x))$ for any f with multiplicative depth up to d

Given ct_x, we first compute

$$ct_{ct} = \text{Encrypt}(pk, ct_x)$$ [strictly speaking, encrypt bit by bit]

This is a fresh ciphertext so we can perform operations of depth up to d on ct_{ct}. Since the public key includes a copy of the decryption key (ct_{sk}), we can homomorphically evaluate the decryption function:

$$\left.\begin{aligned} ct_{ct} &= \text{Encrypt}(pk, ct_x) \\ ct_{sk} &= \text{Encrypt}(pk, sk) \end{aligned}\right\} \underbrace{\text{Encrypt}(pk, \text{Decrypt}(sk, ct))}_{\text{depth-}d'\text{ computation}} = \text{Encrypt}(pk, x)$$

This is a new encryption of m, and we can continue performing homomorphic operations on m (of depth $d - d'$)

Bootstrapping is a general technique that converts any SWHE that can evaluate its own decryption function (plus a little more) into an FHE scheme. Transformation requires additional circular security assumption (namely, that it is OK to publish an encryption of the scheme's own public key. [The GSW scheme supports bootstrapping — decryption is a threshold inner product ; choose parameters carefully]

Open problem : Build FHE from LWE (or another standard assumption) without the circular security assumption.

The GSW homomorphic operations have a lot of applications. We will describe three of them in the remaining weeks of this course : homomorphic signatures, attribute-based encryption, and non-interactive zero-knowledge.

Homomorphic signatures : Analog of homomorphic encryption for signatures

↳ given signature σ on input x, can compute signature σ_f on any function evaluation $f(x)$ where σ_f verifies with respect to function f and value $f(x)$

↳ useful for authenticating computations (e.g., cloud provider can prove that performed a particular computation correctly on signed data)

Syntax: $\mathrm{Setup}(1^\lambda) \to (sk, vk)$: Outputs a signing key sk and a verification key vk

$\mathrm{Sign}(sk, x) \to \sigma_x$: Output a signature on a message x

$\mathrm{Eval}(vk, \sigma_x, f) \to \sigma_{f(x)}$: Takes a signature on x and a function f and outputs a signature on $f(x)$ ← public operation that does not require knowledge of signing key

$\mathrm{Verify}(vk, f, y, \sigma_f) \to 0/1$: Checks whether signature σ_f is a signature on value y with respect to function f

Correctness :
$$\left.\begin{array}{l}(sk, vk) \leftarrow \mathrm{Setup}(1^\lambda) \\ \sigma_x \leftarrow \mathrm{Sign}(sk, x) \\ \sigma_{f(x)} \leftarrow \mathrm{Eval}(vk, \sigma_x, f)\end{array}\right. \implies \mathrm{Verify}(vk, f, f(x), \sigma_{f(x)}) = 1$$

(One-Time) Unforgeability :

adversary	challenger
	$(vk, sk) \leftarrow \mathrm{Setup}(1^\lambda)$
← vk
m →
← $\sigma \leftarrow \mathrm{Sign}(sk, m)$
(f, y, σ_y)

adversary wins if $y \neq f(x)$ but $\mathrm{Verify}(vk, f, y, \sigma_y) = 1$.

Intuitively : the adversary can always produce new signatures (by using the homomorphic properties of the underlying signature scheme), but cannot produce a new signature that does correspond to a valid computation on the signatures it is given

↳ Reason why homomorphic signatures are useful for authenticating computations

Compactness : signatures are "short" (depend essentially on the size of the output and the depth of the circuit) :

$$|\sigma_{f(x)}| = \mathrm{poly}(\lambda, d)$$

← depth of the circuit / computation performed on the signatures (can also consider other metrics like $|\sigma_{f(x)}| = \mathrm{poly}(\lambda, \log |C|)$)

↳ in particular, if we compute a signature over a large database (i.e., many signatures), the resulting signature that authenticates the computation can still be short

Starting point: Recall GPV signatures ("hash and sign")

$vk: A, \quad sk: td_A$

Signature on message m is a short vector $u \in \mathbb{Z}_q^m$ such that $Au = H(m)$

$H: m \to \mathbb{Z}_q^n$ (modeled as random oracle)

can view this as "target vector"

For homomorphic signatures, we will sign bit-by-bit. Suppose we are signing t-bit strings.

$vk: A, V_1, \dots, V_t \xleftarrow{R} \mathbb{Z}_q^{n \times m}$

$sk: td_A$

To sign an input $x \in \{0,1\}^t$, we will sample short $U_1, \dots, U_t \in \mathbb{Z}_q^{m \times m}$ where

$$AU_i = \underbrace{V_i + x_i G}_{\text{target matrix like in GPV}}$$

target matrix like in GPV signatures (notice that this is uniform since $V_i \xleftarrow{R} \mathbb{Z}_q^{n \times m}$ and we will only ever sign a single message)

HW5 will explore what goes wrong if we sign multiple messages

Signing directly corresponds to preimage sampling (using trapdoor for A).

Homomorphic operations are defined exactly as in GSW FHE:

GSW Ciphertext: $C = AR + x_i G$ — C: ciphertext (publicly known); R: encryption randomness

Homomorphic Signature: $AU_i = V_i + x_i G$ — U_i: signature; V_i: public component

Suppose we have
$$V_1 = AU_1 + x_1 G$$
$$V_2 = AU_2 + x_2 G$$
$$\Longrightarrow V_1 + V_2 = A(U_1 + U_2) + \underbrace{(x_1 + x_2)}_{\text{sum of } x_1 + x_2} G$$

$\Longrightarrow (U_1 + U_2)$ can be viewed as a signature on the sum $x_1 + x_2$ with respect to the public component $V_1 + V_2$

For multiplication, we use the analog of GSW multiplication:
$$V_1 = AU_1 + x_1 G$$
$$V_2 = AU_2 + x_2 G$$
$$\Longrightarrow \underbrace{V_1 G^{-1}(V_2)}_{\text{public component, depends only on public parameters}} = AU_1 G^{-1}(V_2) + x_1 AU_2 + x_1 x_2 G$$
$$= A\underbrace{\left(U_1 G^{-1}(V_2) + x_1 U_2\right)}_{\text{signature on } x_1x_2} + \underbrace{x_1 x_2}_{\text{product of } x_1 x_2} G$$

signature on $x_1 x_2$, depends on U_1, x_1, V_2, which are known to the evaluator

norm of signature grows by multiplicative factor $O(m)$

Summary: Suppose
$$V_1 = AU_1 + x_1 G$$
$$\vdots \qquad \vdots$$
$$V_t = AU_t + x_t G$$

V_i: public values; U_i, x_i: part of signatures

$\Longrightarrow$ given $V_1, \dots, V_t$, and circuit C, can compute V_C using homomorphic evaluation procedure described above

given $V_1, \dots, V_t$, signatures $U_1, \dots, U_t$, message $x \in \{0,1\}^t$, and circuit C, can compute U_C using homomorphic evaluation procedure described above

↳ these procedures then satisfy the following property:
$$V_C = AU_C + C(x) \cdot G$$
and $\|U_C\| \leq \beta \cdot m^{O(d)}$

↳ correspondingly, just need to set $q > \beta \cdot m^{O(d)}$

$[\log q > O(d \log m + \log \beta)$

Summary: To verify, compute V_c from $V_1, \ldots, V_t$ and check if $V_c = AU_c + y \cdot G$ and $\|U_c\| < \beta \cdot m^{O(d)}$

computed from C and public parameters

signature

claimed value