# LWE and cryptography

June 6, 2019

# Motivation

- Originally as an attempt to solve lattice problems

# Motivation

- Originally as an attempt to solve lattice problems
- Worst-case lattice problems reduce to LWE

# Motivation

- Originally as an attempt to solve lattice problems
- Worst-case lattice problems reduce to LWE
- Need new encryption schemes that are at least as hard to break as problems difficult for quantum computers

# Motivation

- Originally as an attempt to solve lattice problems
- Worst-case lattice problems reduce to LWE
- Need new encryption schemes that are at least as hard to break as problems difficult for quantum computers
- LWE fits the bill

# Intuition of LWE

$$14s_1 + 15s_2 + 5s_3 + 2s_4 \approx 8 \pmod{17}$$

$$13s_1 + 14s_2 + 14s_3 + 6s_4 \approx 16 \pmod{17}$$

$$6s_1 + 10s_2 + 13s_3 + s_4 \approx 3 \pmod{17}$$

$$10s_1 + 4s_2 + 12s_3 + 16s_4 \approx 12 \pmod{17}$$

$$9s_1 + 5s_2 + 9s_3 + 6s_4 \approx 9 \pmod{17}$$

$$3s_1 + 6s_2 + 4s_3 + 5s_4 \approx 16 \pmod{17}$$

$$\vdots$$

$$6s_1 + 7s_2 + 16s_3 + 2s_4 \approx 3 \pmod{17}$$

# Definition (LWE Distribution)

Let $A_{\vec{s},\chi}$ be a distribution on $\mathbb{Z}_q^n \times \mathbb{Z}_q$ as follows:

# Definition (LWE Distribution)

Let $A_{\vec{s},\chi}$ be a distribution on $\mathbb{Z}_q^n \times \mathbb{Z}_q$ as follows:

- Pick $\vec{a} \in \mathbb{Z}_q^n$ uniformly randomly

# Definition (LWE Distribution)

Let $A_{\vec{s},\chi}$ be a distribution on $\mathbb{Z}_q^n \times \mathbb{Z}_q$ as follows:

- Pick $\vec{a} \in \mathbb{Z}_q^n$ uniformly randomly
- Pick $e$ according to $\chi$

# Definition (LWE Distribution)

Let $A_{\vec{s},\chi}$ be a distribution on $\mathbb{Z}_q^n \times \mathbb{Z}_q$ as follows:

- Pick $\vec{a} \in \mathbb{Z}_q^n$ uniformly randomly
- Pick $e$ according to $\chi$
- Output $(\vec{a}, \langle \vec{a}, \vec{s} \rangle + e)$

# Definition (LWE)

Given samples from $A_{\vec{s},\chi}$,

- Search version: Find $\vec{s}$.

# Definition (LWE)

Given samples from $A_{\vec{s},\chi}$,

- Search version: Find $\vec{s}$.
- Decision version: Distinguish between $A_{\vec{s},\chi}$ and the uniform distribution.

# Search to decision reduction

# GSW's LWE formulation

1. Take the rows of $A$ as $b_i || \vec{a}_i$ instead

# GSW's LWE formulation

1. Take the rows of $A$ as $b_i||\vec{a}_i$ instead
2. Redefine $\vec{s}$ as $(1, -\vec{s})$

# GSW's LWE formulation

1. Take the rows of $A$ as $b_i||\vec{a}_i$ instead
2. Redefine $\vec{s}$ as $(1, -\vec{s})$
3. Either $A$ is uniform, or $\exists\vec{s}$ with first coefficient 1 s.t. $A\vec{s} = \vec{e}$, where $\vec{e}$ comes from $\chi$.

# Asymmetric cryptography

Two parties who may have never communicated before may securely exchange information, by using the tools below:

# Asymmetric cryptography

Two parties who may have never communicated before may securely exchange information, by using the tools below:

- $pk$: "public key", used for encryption

# Asymmetric cryptography

Two parties who may have never communicated before may securely exchange information, by using the tools below:

- *pk*: "public key", used for encryption
- $\mathrm{Enc}_{pk} : \Sigma^* \times \mathcal{R} \to U$

# Asymmetric cryptography

Two parties who may have never communicated before may securely exchange information, by using the tools below:

- *pk*: "public key", used for encryption
- $\text{Enc}_{pk} : \Sigma^* \times \mathcal{R} \to U$
- *sk*: "secret key", used for decryption

# Asymmetric cryptography

Two parties who may have never communicated before may securely exchange information, by using the tools below:

- *pk*: "public key", used for encryption
- $\mathsf{Enc}_{pk} : \Sigma^* \times \mathcal{R} \to U$
- *sk*: "secret key", used for decryption
- $\mathsf{Dec}_{sk} : U \to \Sigma^*$

# Asymmetric cryptography

Two parties who may have never communicated before may securely exchange information, by using the tools below:

- $pk$: "public key", used for encryption
- $\mathsf{Enc}_{pk} : \Sigma^* \times \mathcal{R} \to U$
- $sk$: "secret key", used for decryption
- $\mathsf{Dec}_{sk} : U \to \Sigma^*$
- $\mathsf{Dec}_{sk} \circ \mathsf{Enc}_{pk}(x, r) = x$ with overwhelming probability over $r$

# Asymmetric cryptography cont.

Typical use of such a scheme:

1. Alice generates $(pk, sk)$

# Asymmetric cryptography cont.

Typical use of such a scheme:

1. Alice generates $(pk, sk)$
2. Alice sends $pk$ to Bob

# Asymmetric cryptography cont.

Typical use of such a scheme:

1. Alice generates $(pk, sk)$
2. Alice sends $pk$ to Bob
3. Bob encrypts his message using $pk$

# Asymmetric cryptography cont.

Typical use of such a scheme:

1. Alice generates $(pk, sk)$
2. Alice sends $pk$ to Bob
3. Bob encrypts his message using $pk$
4. Bob sends the ciphertext to Alice

# Asymmetric cryptography cont.

Typical use of such a scheme:

1. Alice generates $(pk, sk)$
2. Alice sends $pk$ to Bob
3. Bob encrypts his message using $pk$
4. Bob sends the ciphertext to Alice
5. Alice decrypts it using $sk$

# Safety

What does it mean for an encryption scheme to be “safe”?

# Safety

What does it mean for an encryption scheme to be “safe”?

- Chosen plaintext attack (CPA): The ”intuitive” definition. An (efficient) adversary who’s able to encrypt anything shouldn’t be able to decrypt anything.

# Safety

What does it mean for an encryption scheme to be “safe”?

- Chosen plaintext attack (CPA): The ”intuitive” definition. An (efficient) adversary who’s able to encrypt anything shouldn’t be able to decrypt anything.
- Adaptive chosen-ciphertext attack (CCA2): A stronger definition. An (efficient) adversary who’s also able to decrypt anything but the target, still cannot decrypt the target.

# Safety (Malleability)

A cryptographic scheme is malleable if $\exists f : \Sigma^* \to \Sigma^*$ efficiently invertible, an entity given $pk$ and $\mathrm{Enc}_{pk}(x)$ can evaluate $\mathrm{Enc}_{pk}(f(x))$.

# Safety (Malleability)

A cryptographic scheme is malleable if $\exists f : \Sigma^* \to \Sigma^*$ efficiently invertible, an entity given $pk$ and $\text{Enc}_{pk}(x)$ can evaluate $\text{Enc}_{pk}(f(x))$.

- CCA2 implies non-malleability

# Safety (Malleability)

A cryptographic scheme is malleable if $\exists f : \Sigma^* \to \Sigma^*$ efficiently invertible, an entity given $pk$ and $\mathrm{Enc}_{pk}(x)$ can evaluate $\mathrm{Enc}_{pk}(f(x))$.

- CCA2 implies non-malleability
- Has many flavors (malleable under CPA vs CCA)

# Safety (Malleability)

A cryptographic scheme is malleable if $\exists f : \Sigma^* \to \Sigma^*$ efficiently invertible, an entity given $pk$ and $\mathrm{Enc}_{pk}(x)$ can evaluate $\mathrm{Enc}_{pk}(f(x))$.

- CCA2 implies non-malleability
- Has many flavors (malleable under CPA vs CCA)
- Is this always a bad property to have?

# Homomorphic encryption

Let someone else do the computation for you. Useful when that "someone else" is quantum!

# Homomorphic encryption

Let someone else do the computation for you. Useful when that "someone else" is quantum!

1. Alice generates $(pk, sk, evk)$

# Homomorphic encryption

Let someone else do the computation for you. Useful when that "someone else" is quantum!

1. Alice generates $(pk, sk, evk)$
2. Alice encrypts her message using $pk$

# Homomorphic encryption

Let someone else do the computation for you. Useful when that "someone else" is quantum!

1. Alice generates $(pk, sk, evk)$
2. Alice encrypts her message using $pk$
3. Alice sends $evk$ and the ciphertext to Bob

# Homomorphic encryption

Let someone else do the computation for you. Useful when that "someone else" is quantum!

1. Alice generates $(pk, sk, evk)$
2. Alice encrypts her message using $pk$
3. Alice sends $evk$ and the ciphertext to Bob
4. Bob runs computations on the ciphertext

# Homomorphic encryption

Let someone else do the computation for you. Useful when that "someone else" is quantum!

1. Alice generates $(pk, sk, evk)$
2. Alice encrypts her message using $pk$
3. Alice sends $evk$ and the ciphertext to Bob
4. Bob runs computations on the ciphertext
5. Bob sends the encrypted result back to Alice

# Homomorphic encryption

Let someone else do the computation for you. Useful when that "someone else" is quantum!

1. Alice generates $(pk, sk, evk)$
2. Alice encrypts her message using $pk$
3. Alice sends $evk$ and the ciphertext to Bob
4. Bob runs computations on the ciphertext
5. Bob sends the encrypted result back to Alice
6. Alice decrypts it using $sk$.

# Idea of GSW's homomorphic encryption

Intuitive idea as follows:

- private key $\vec{v}$ is a vector

# Idea of GSW's homomorphic encryption

Intuitive idea as follows:

- private key $\vec{v}$ is a vector
- ciphertexts are matrices with $\vec{v}$ approximately as an eigenvector

# Idea of GSW's homomorphic encryption

Intuitive idea as follows:

- private key $\vec{v}$ is a vector
- ciphertexts are matrices with $\vec{v}$ approximately as an eigenvector
- plaintexts are corresponding approximate eigenvalues

# Idea of GSW's homomorphic encryption

Intuitive idea as follows:

- private key $\vec{v}$ is a vector
- ciphertexts are matrices with $\vec{v}$ approximately as an eigenvector
- plaintexts are corresponding approximate eigenvalues
- $(C_1 + C_2)\vec{v} \approx (\lambda_1 + \lambda_2)\vec{v}$

# Idea of GSW's homomorphic encryption

Intuitive idea as follows:

- private key $\vec{v}$ is a vector
- ciphertexts are matrices with $\vec{v}$ approximately as an eigenvector
- plaintexts are corresponding approximate eigenvalues
- $(C_1 + C_2)\vec{v} \approx (\lambda_1 + \lambda_2)\vec{v}$
- $(C_1 C_2)\vec{v} \approx (\lambda_1 \lambda_2)\vec{v}$

# Idea of GSW's homomorphic encryption

Intuitive idea as follows:

- private key $\vec{v}$ is a vector
- ciphertexts are matrices with $\vec{v}$ approximately as an eigenvector
- plaintexts are corresponding approximate eigenvalues
- $(C_1 + C_2)\vec{v} \approx (\lambda_1 + \lambda_2)\vec{v}$
- $(C_1 C_2)\vec{v} \approx (\lambda_1 \lambda_2)\vec{v}$
- When plaintexts are booleans, $I_N - C_1 C_2$ encodes NAND.

# GSW's tools

Fix $q = 2^l$, $N = kl$, define the following functions on $\mathbb{Z}_q^*$. Easier to understand with examples with $l = 4$, $k = 3$, $N = 12$.

- $\text{BitDecomp}(1001_2, 0010_2, 1100_2) = (1, 0, 0, 1, 0, 0, 1, 0, 1, 1, 0, 0)$

# GSW's tools

Fix $q = 2^l$, $N = kl$, define the following functions on $\mathbb{Z}_q^*$. Easier to understand with examples with $l = 4$, $k = 3$, $N = 12$.

- $\mathrm{BitDecomp}(1001_2, 0010_2, 1100_2) = (1, 0, 0, 1, 0, 0, 1, 0, 1, 1, 0, 0)$
- $\mathrm{BitDecomp}^{-1}(1, 0, 0, 1, 0, 0, 1, 0, 1, 1, 0, 0) = (1001_2, 0010_2, 1100_2)$

# GSW's tools

Fix $q = 2^l$, $N = kl$, define the following functions on $\mathbb{Z}_q^*$. Easier to understand with examples with $l = 4$, $k = 3$, $N = 12$.

- $\mathrm{BitDecomp}(1001_2, 0010_2, 1100_2) = (1, 0, 0, 1, 0, 0, 1, 0, 1, 1, 0, 0)$
- $\mathrm{BitDecomp}^{-1}(1, 0, 0, 1, 0, 0, 1, 0, 1, 1, 0, 0) = (1001_2, 0010_2, 1100_2)$
- $\mathrm{BitDecomp}^{-1}(0, 0, 10_2, 0, 0, 1, 10_2, 1, 11_2, 0, 1, 1) = (0100_2, 1001_2, 1011_2)$

# GSW's tools

Fix $q = 2^l$, $N = kl$, define the following functions on $\mathbb{Z}_q^*$. Easier to understand with examples with $l = 4$, $k = 3$, $N = 12$.

- $\text{BitDecomp}(1001_2, 0010_2, 1100_2) = (1, 0, 0, 1, 0, 0, 1, 0, 1, 1, 0, 0)$
- $\text{BitDecomp}^{-1}(1, 0, 0, 1, 0, 0, 1, 0, 1, 1, 0, 0) = (1001_2, 0010_2, 1100_2)$
- $\text{BitDecomp}^{-1}(0, 0, 10_2, 0, 0, 1, 10_2, 1, 11_2, 0, 1, 1) = (0100_2, 1001_2, 1011_2)$
- $\text{Flatten} = \text{BitDecomp} \circ \text{BitDecomp}^{-1}$

# GSW's tools

Fix $q = 2^l$, $N = kl$, define the following functions on $\mathbb{Z}_q^*$. Easier to understand with examples with $l = 4$, $k = 3$, $N = 12$.

- $\text{BitDecomp}(1001_2, 0010_2, 1100_2) = (1, 0, 0, 1, 0, 0, 1, 0, 1, 1, 0, 0)$
- $\text{BitDecomp}^{-1}(1, 0, 0, 1, 0, 0, 1, 0, 1, 1, 0, 0) = (1001_2, 0010_2, 1100_2)$
- $\text{BitDecomp}^{-1}(0, 0, 10_2, 0, 0, 1, 10_2, 1, 11_2, 0, 1, 1) = (0100_2, 1001_2, 1011_2)$
- $\text{Flatten} = \text{BitDecomp} \circ \text{BitDecomp}^{-1}$
- $\text{Powersof2}(11_2, 0_2, 1_2) = (11_2, 110_2, 1100_2, 11000_2, 0_2, 0_2, 0_2, 0_2, 1_2, 10_2, 100_2, 1000_2)$

# GSW's tools (Flatten)

$$\begin{aligned}\mathrm{Flatten}(&110_2, 101_2, 1_2, 11_2,\\ &110_2, 101_2, 1_2, 11_2,\\ &110_2, 101_2, 1_2, 11_2)\end{aligned}$$

# GSW's tools (Flatten)

$$\begin{aligned}\text{Flatten}(&110_2, 101_2, 1_2, 11_2,\\ &110_2, 101_2, 1_2, 11_2,\\ &110_2, 101_2, 1_2, 11_2)\end{aligned}$$

$$\begin{aligned}= \text{BitDecomp} \circ \text{BitDecomp}^{-1}(&110_2, 101_2, 1_2, 11_2,\\ &110_2, 101_2, 1_2, 11_2,\\ &110_2, 101_2, 1_2, 11_2)\end{aligned}$$

# GSW's tools (Flatten)

$$\begin{aligned}\mathsf{Flatten}(&110_2, 101_2, 1_2, 11_2,\\ &110_2, 101_2, 1_2, 11_2,\\ &110_2, 101_2, 1_2, 11_2)\end{aligned}$$

$$\begin{aligned}= \mathsf{BitDecomp} \circ \mathsf{BitDecomp}^{-1}(&110_2, 101_2, 1_2, 11_2,\\ &110_2, 101_2, 1_2, 11_2,\\ &110_2, 101_2, 1_2, 11_2)\end{aligned}$$

$$\begin{aligned}= \mathsf{BitDecomp}(&110000_2 + 10100_2 + 10_2 + 11_2,\\ &110000_2 + 10100_2 + 10_2 + 11_2,\\ &110000_2 + 10100_2 + 10_2 + 11_2)\end{aligned}$$

# GSW's tools (Flatten)

$$\begin{aligned} = \mathrm{BitDecomp}(&1001001_2, \\ &1001001_2, \\ &1001001_2) \end{aligned}$$

# GSW's tools (Flatten)

$$= \mathrm{BitDecomp}(1001001_2,\\ 1001001_2,\\ 1001001_2)$$

$$= \mathrm{BitDecomp}(1001_2, 1001_2, 1001_2)$$

# GSW's tools (Flatten)

$$\begin{aligned} = \mathrm{BitDecomp}(&1001001_2,\\ &1001001_2,\\ &1001001_2) \end{aligned}$$

$$= \mathrm{BitDecomp}(1001_2, 1001_2, 1001_2)$$

$$= (1, 0, 0, 1, 1, 0, 0, 1, 1, 0, 0, 1)$$

# GSW's tools cont.

Some basic properties

- $\langle \mathrm{BitDecomp}(\vec{a}), \mathrm{Powersof2}(\vec{b}) \rangle = \langle \vec{a}, \vec{b} \rangle$

# GSW's tools cont.

Some basic properties

- $\langle \text{BitDecomp}(\vec{a}), \text{Powersof2}(\vec{b}) \rangle = \langle \vec{a}, \vec{b} \rangle$
- $\langle \vec{a}, \text{Powersof2}(\vec{b}) \rangle = \langle \text{BitDecomp}^{-1}(\vec{a}), \vec{b} \rangle$

# GSW's tools cont.

Some basic properties

- $\langle \text{BitDecomp}(\vec{a}), \text{Powersof2}(\vec{b}) \rangle = \langle \vec{a}, \vec{b} \rangle$
- $\langle \vec{a}, \text{Powersof2}(\vec{b}) \rangle = \langle \text{BitDecomp}^{-1}(\vec{a}), \vec{b} \rangle$
- $= \langle \text{Flatten}(\vec{a}), \text{Powersof2}(\vec{b}) \rangle$

# GSW's Construction - Setup

Choose the following parameters:

- $q$ of $\kappa(\lambda, L)$ bits

# GSW's Construction - Setup

Choose the following parameters:

- $q$ of $\kappa(\lambda, L)$ bits
- $n(\lambda, L)$ lattice dimension parameter

# GSW's Construction - Setup

Choose the following parameters:

- $q$ of $\kappa(\lambda, L)$ bits
- $n(\lambda, L)$ lattice dimension parameter
- Error distribution $\chi(\lambda, L)$

# GSW's Construction - Setup

Choose the following parameters:

- $q$ of $\kappa(\lambda, L)$ bits
- $n(\lambda, L)$ lattice dimension parameter
- Error distribution $\chi(\lambda, L)$
- $m(\lambda, L) \in O(n \log q)$

# GSW's Construction - Setup

Choose the following parameters:

- $q$ of $\kappa(\lambda, L)$ bits
- $n(\lambda, L)$ lattice dimension parameter
- Error distribution $\chi(\lambda, L)$
- $m(\lambda, L) \in O(n \log q)$
- $l = 1 + \log q$

# GSW's Construction - Setup

Choose the following parameters:

- $q$ of $\kappa(\lambda, L)$ bits
- $n(\lambda, L)$ lattice dimension parameter
- Error distribution $\chi(\lambda, L)$
- $m(\lambda, L) \in O(n \log q)$
- $l = 1 + \log q$
- $N = (n + 1) \times l$

# GSW's Construction - Secret Keygen

1. Sample $\vec{t} \leftarrow \mathbb{Z}_q^n$ uniformly. This represents the solution of the LWE system of equations.

# GSW's Construction - Secret Keygen

1. Sample $\vec{t} \leftarrow \mathbb{Z}_q^n$ uniformly. This represents the solution of the LWE system of equations.
2. Output $sk = \vec{s} = 1|| - \vec{t} \in \mathbb{Z}_q^{n+1}$

# GSW's Construction - Secret Keygen

1. Sample $\vec{t} \leftarrow \mathbb{Z}_q^n$ uniformly. This represents the solution of the LWE system of equations.
2. Output $sk = \vec{s} = 1|| - \vec{t} \in \mathbb{Z}_q^{n+1}$
3. Let $\vec{v} = \text{Powersof2}(\vec{s})$

# GSW's Construction - Public Keygen

1. Generate $B \leftarrow \mathbb{Z}_q^{m \times n}$ uniformly

# GSW's Construction - Public Keygen

1. Generate $B \leftarrow \mathbb{Z}_q^{m \times n}$ uniformly
2. Sample $\vec{e} \leftarrow \chi$

# GSW's Construction - Public Keygen

1. Generate $B \leftarrow \mathbb{Z}_q^{m \times n}$ uniformly
2. Sample $\vec{e} \leftarrow \chi$
3. Set $\vec{b} = B\vec{t} + \vec{e}$

# GSW's Construction - Public Keygen

1. Generate $B \leftarrow \mathbb{Z}_q^{m \times n}$ uniformly
2. Sample $\vec{e} \leftarrow \chi$
3. Set $\vec{b} = B\vec{t} + \vec{e}$
4. Set $pk$ as $\vec{b}$ on the first column, followed by the columns of $B$.

# GSW's Construction - Public Keygen

1. Generate $B \leftarrow \mathbb{Z}_q^{m \times n}$ uniformly
2. Sample $\vec{e} \leftarrow \chi$
3. Set $\vec{b} = B\vec{t} + \vec{e}$
4. Set $pk$ as $\vec{b}$ on the first column, followed by the columns of $B$.
5. Observe that $A\vec{s} = \vec{e}$

# GSW's Construction - Encryption

Input: $\mu$

1.

# References


- O. Regev. The Learning with Errors Problem.
- C. Gentry, A. Sahai, B. Waters. Homomorphic Encryption from Learning with Errors: Conceptually-Simpler, Asymptotically-Faster, Attribute-Based.