LWE and cryptography

June 5, 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 χ

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 χ
- 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

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
- $\mathsf{Enc}_{pk} : \Sigma^* \times \mathcal{R} \to U$

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

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
- $\mathrm{Enc}_{pk} : \Sigma^* \times \mathcal{R} \to U$
- sk: "secret key", used for decryption
- $\mathrm{Dec}_{sk} : U \to \Sigma^*$
- $\mathrm{Dec}_{sk} \circ \mathrm{Enc}_{pk}(x, r) = x$ 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 $\mathrm{Enc}_{pk}(x)$ can evaluate $\mathrm{Enc}_{pk}(f(x))$.

- 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 (Bitdecomp)

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 (Bitdecomp)

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

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

GSW's tools (Bitdecomp)

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

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

GSW's tools (Bitdecomp)

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

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

GSW's tools (Flatten)

$\mathrm{Flatten}(110_2, 101_2, 1_2, 11_2,$
$110_2, 101_2, 1_2, 11_2, 110_2, 101_2, 1_2, 11_2)$

GSW's tools (Flatten)

$$\mathsf{Flatten}(110_2, 101_2, 1_2, 11_2,\\ 110_2, 101_2, 1_2, 11_2, 110_2, 101_2, 1_2, 11_2)\\ = \mathsf{BitDecomp} \circ \mathsf{BitDecomp}^{-1}(110_2, 101_2, 1_2, 11_2,\\ 110_2, 101_2, 1_2, 11_2, 110_2, 101_2, 1_2, 11_2)$$

GSW's tools (Flatten)

$\mathrm{Flatten}(110_2, 101_2, 1_2, 11_2,$
$110_2, 101_2, 1_2, 11_2, 110_2, 101_2, 1_2, 11_2)$
$= \mathrm{BitDecomp} \circ \mathrm{BitDecomp}^{-1}(110_2, 101_2, 1_2, 11_2,$
$110_2, 101_2, 1_2, 11_2, 110_2, 101_2, 1_2, 11_2)$
$= \mathrm{BitDecomp}(1000_2 \times 110_2 + 100_2 \times 101_2 + 10_2 \times 1_2 + 11_2,$
$1000_2 \times 110_2 + 100_2 \times 101_2 + 10_2 \times 1_2 + 11_2, 1000_2 \times 110_2 +$
$100_2 \times 101_2 + 10_2 \times 1_2 + 11_2)$

GSW's tools (Flatten)

$\mathrm{Flatten}(110_2, 101_2, 1_2, 11_2,$
$110_2, 101_2, 1_2, 11_2, 110_2, 101_2, 1_2, 11_2)$
$= \mathrm{BitDecomp} \circ \mathrm{BitDecomp}^{-1}(110_2, 101_2, 1_2, 11_2,$
$110_2, 101_2, 1_2, 11_2, 110_2, 101_2, 1_2, 11_2)$
$= \mathrm{BitDecomp}(1000_2 \times 110_2 + 100_2 \times 101_2 + 10_2 \times 1_2 + 11_2,$
$1000_2 \times 110_2 + 100_2 \times 101_2 + 10_2 \times 1_2 + 11_2, 1000_2 \times 110_2 +$
$100_2 \times 101_2 + 10_2 \times 1_2 + 11_2)$
$= \mathrm{BitDecomp}(110000_2 + 10100_2 + 10_2 + 11_2,$
$110000_2 + 10100_2 + 10_2 + 11_2, 110000_2 + 10100_2 + 10_2 + 11_2)$

GSW's tools (Flatten)

$\text{Flatten}(110_2, 101_2, 1_2, 11_2,$
$110_2, 101_2, 1_2, 11_2, 110_2, 101_2, 1_2, 11_2)$
$= \text{BitDecomp} \circ \text{BitDecomp}^{-1}(110_2, 101_2, 1_2, 11_2,$
$110_2, 101_2, 1_2, 11_2, 110_2, 101_2, 1_2, 11_2)$
$= \text{BitDecomp}(1000_2 \times 110_2 + 100_2 \times 101_2 + 10_2 \times 1_2 + 11_2,$
$1000_2 \times 110_2 + 100_2 \times 101_2 + 10_2 \times 1_2 + 11_2, 1000_2 \times 110_2 +$
$100_2 \times 101_2 + 10_2 \times 1_2 + 11_2)$
$= \text{BitDecomp}(110000_2 + 10100_2 + 10_2 + 11_2,$
$110000_2 + 10100_2 + 10_2 + 11_2, 110000_2 + 10100_2 + 10_2 + 11_2)$
$= \text{BitDecomp}(1001001_2, 1001001_2, 1001001_2)$

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.