Learning with Error and GSW's Homomorphic Encryption

June 28, 2019

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 problem)

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

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

Definition (LWE problem)

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

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

Definition (LWE problem)

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

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

Definition (LWE problem)

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

- Search version: Find $\vec{s}$.
- Decision version: Distinguish between $A_{\vec{s},\chi}$ and the uniform distribution.
- Both are difficult
- Decision isn't much easier than search

Search to decision reduction

Works when $q \in O(poly(n))$ is prime.

- Inductive process; solve variable-by-variable

Search to decision reduction

Works when $q \in O(poly(n))$ is prime.

- Inductive process; solve variable-by-variable
- Guess $s_1 = k$

Search to decision reduction

Works when $q \in O(poly(n))$ is prime.

- Inductive process; solve variable-by-variable
- Guess $s_1 = k$
- Pick r at random

Search to decision reduction

Works when $q \in O(poly(n))$ is prime.

- Inductive process; solve variable-by-variable
- Guess $s_1 = k$
- Pick r at random
- Map $(\vec{a}, b)$ to $\left(\vec{a} + \begin{pmatrix} r \\ 0 \\ 0 \\ \vdots \\ 0 \end{pmatrix}, b + rk\right)$

Search to decision reduction

Works when $q \in O(poly(n))$ is prime.

- Inductive process; solve variable-by-variable
- Guess $s_1 = k$
- Pick r at random
- Map $(\vec{a}, b)$ to $\left(\vec{a} + \begin{pmatrix} r \\ 0 \\ 0 \\ \vdots \\ 0 \end{pmatrix}, b + rk\right)$
- The above maps $A_{\vec{s},\chi}$ to itself if the guess was correct.

Search to decision reduction

Works when $q \in O(poly(n))$ is prime.

- Inductive process; solve variable-by-variable
- Guess $s_1 = k$
- Pick r at random
- Map $(\vec{a}, b)$ to $\left(\vec{a} + \begin{pmatrix} r \\ 0 \\ 0 \\ \vdots \\ 0 \end{pmatrix}, b + rk\right)$
- The above maps $A_{\vec{s},\chi}$ to itself if the guess was correct.
- Maps to the uniform distribution otherwise.

Search to decision reduction

Works when $q \in O(poly(n))$ is prime.

- Inductive process; solve variable-by-variable
- Guess $s_1 = k$
- Pick r at random
- Map $(\vec{a}, b)$ to $\left(\vec{a} + \begin{pmatrix} r \\ 0 \\ 0 \\ \vdots \\ 0 \end{pmatrix}, b + rk\right)$
- The above maps $A_{\vec{s},\chi}$ to itself if the guess was correct.
- Maps to the uniform distribution otherwise.
- Check using the oracle. Try another k until the guess is correct.

GSW's LWE formulation

- Define A_i as $b_i||\vec{a}_i$

GSW's LWE formulation

- Define A_i as $b_i||\vec{a}_i$
- Redefine $\vec{s}$ as $(1, -\vec{s})$

GSW's LWE formulation

- Define A_i as $b_i||\vec{a}_i$
- Redefine $\vec{s}$ as $(1, -\vec{s})$
- $A\vec{s} = \vec{e}$,

Asymmetric cryptography

Allows secure communication by using the tools below:

Asymmetric cryptography

Allows secure communication by using the tools below:

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

Asymmetric cryptography

Allows secure communication by using the tools below:

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

Asymmetric cryptography

Allows secure communication 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

Allows secure communication 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

Allows secure communication 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
- $\text{Dec}_{sk} : U \to \Sigma^*$.
- $\text{Dec}_{sk} \circ \text{Enc}_{pk}(x, r) = x$ 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

Security

- What does it mean for an encryption scheme to be secure?

Security

- What does it mean for an encryption scheme to be secure?
- Chosen plaintext attack (CPA): The "intuitive" definition. An (efficient) adversary who's able to encrypt anything shouldn't be able to decrypt anything.

Security

- What does it mean for an encryption scheme to be secure?
- Chosen plaintext attack (CPA): The "intuitive" definition. An (efficient) adversary who's able to encrypt anything shouldn't be able to decrypt anything.
- Ciphertext indistinguishability

Homomorphic encryption

- Allows computation on ciphertext without decrypting

Homomorphic encryption

- Allows computation on ciphertext without decrypting
- Given function f and $\mathrm{Enc}(x)$, allows computing $\mathrm{Enc}(f(x))$.

Homomorphic encryption

- Allows computation on ciphertext without decrypting
- Given function f and $\mathrm{Enc}(x)$, allows computing $\mathrm{Enc}(f(x))$.
- May involve an evaluation key.

Homomorphic encryption

- Allows computation on ciphertext without decrypting
- Given function f and $\mathrm{Enc}(x)$, allows computing $\mathrm{Enc}(f(x))$.
- May involve an evaluation key.
- Application includes quantum computing

Idea of GSW's homomorphic encryption

Intuitive idea as follows:

- Using the "approximate eigenvector": $C\vec{v} \approx \lambda\vec{v}$

Idea of GSW's homomorphic encryption

Intuitive idea as follows:

- Using the "approximate eigenvector": $C\vec{v} \approx \lambda\vec{v}$
- $\vec{v}$ is the private key

Idea of GSW's homomorphic encryption

Intuitive idea as follows:

- Using the "approximate eigenvector": $C\vec{v} \approx \lambda\vec{v}$
- $\vec{v}$ is the private key
- C is the ciphertext

Idea of GSW's homomorphic encryption

Intuitive idea as follows:

- Using the "approximate eigenvector": $C\vec{v} \approx \lambda\vec{v}$
- $\vec{v}$ is the private key
- C is the ciphertext
- λ is the plaintext

Idea of GSW's homomorphic encryption

Intuitive idea as follows:

- Using the "approximate eigenvector": $C\vec{v} \approx \lambda\vec{v}$
- $\vec{v}$ is the private key
- C is the ciphertext
- λ is the plaintext
- $(C_1 + C_2)\vec{v} \approx (\lambda_1 + \lambda_2)\vec{v}$

Idea of GSW's homomorphic encryption

Intuitive idea as follows:

- Using the "approximate eigenvector": $C\vec{v} \approx \lambda\vec{v}$
- $\vec{v}$ is the private key
- C is the ciphertext
- λ is the plaintext
- $(C_1 + C_2)\vec{v} \approx (\lambda_1 + \lambda_2)\vec{v}$
- $(C_1 C_2)\vec{v} \approx (\lambda_1\lambda_2)\vec{v}$

Idea of GSW's homomorphic encryption

Intuitive idea as follows:

- Using the "approximate eigenvector": $C\vec{v} \approx \lambda\vec{v}$
- $\vec{v}$ is the private key
- C is the ciphertext
- λ is the plaintext
- $(C_1 + C_2)\vec{v} \approx (\lambda_1 + \lambda_2)\vec{v}$
- $(C_1 C_2)\vec{v} \approx (\lambda_1\lambda_2)\vec{v}$
- When plaintexts are booleans, $I_N - C_1 C_2$ encodes NAND.

GSW's tools

Define the following functions on $\mathbb{Z}_q^*$. Easier to understand with examples. Take $q = 2^4$.

GSW's tools

Define the following functions on $\mathbb{Z}_q^*$. Easier to understand with examples. Take $q = 2^4$.

- $\text{Powersof2}(1_2, 11_2) =$
  $(1000_2, 100_2, 10_2, 1_2, 1000_2, 1100_2, 110_2, 11_2)$

GSW's tools

Define the following functions on $\mathbb{Z}_q^*$. Easier to understand with examples. Take $q = 2^4$.

- $\text{Powersof2}(1_2, 11_2) = (1000_2, 100_2, 10_2, 1_2, 1000_2, 1100_2, 110_2, 11_2)$
- $\text{BitDecomp}(1001_2, 0010_2) = (1, 0, 0, 1, 0, 0, 1, 0)$

GSW's tools

Define the following functions on $\mathbb{Z}_q^*$. Easier to understand with examples. Take $q = 2^4$.

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

GSW's tools

Define the following functions on $\mathbb{Z}_q^*$. Easier to understand with examples. Take $q = 2^4$.

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

GSW's tools

Define the following functions on $\mathbb{Z}_q^*$. Easier to understand with examples. Take $q = 2^4$.

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

GSW's tools (Flatten)

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

GSW's tools (Flatten)

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

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

GSW's tools (Flatten)

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

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

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

GSW's tools (Flatten)

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

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

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

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

GSW's tools cont.

Some basic properties

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

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:

- Modulus $q = 2^l$ (to simplify some proofs)

GSW's Construction - Setup

Choose the following parameters:

- Modulus $q = 2^l$ (to simplify some proofs)
- Lattice dimension n

GSW's Construction - Setup

Choose the following parameters:

- Modulus $q = 2^l$ (to simplify some proofs)
- Lattice dimension n
- Error distribution $\chi(\lambda, L)$

GSW's Construction - Setup

Choose the following parameters:

- Modulus $q = 2^l$ (to simplify some proofs)
- Lattice dimension n
- Error distribution $\chi(\lambda, L)$
- $m \in O(n \log q)$

GSW's Construction - Secret Keygen

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

GSW's Construction - Secret Keygen

1. Sample $\vec{s} \leftarrow \mathbb{Z}_q^n$ uniformly. This represents the solution of the LWE system of equations.
2. Output sk as 1 on the first coordinate, followed by $-\vec{s}$.

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 $\vec{e} \leftarrow \chi^m$

GSW's Construction - Public Keygen

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

GSW's Construction - Encryption

Input: μ

1. Sample $R \in \{0,1\}^{N \times m}$

GSW's Construction - Encryption

Input: μ

1. Sample $R \in \{0,1\}^{N \times m}$
2. Output $\text{Flatten}(\mu \cdot I + \text{BitDecomp}(R \cdot pk))$

GSW's Construction - Decryption

$$\text{Flatten}(\mu \cdot I + \text{BitDecomp}(R \cdot pk)) \cdot \text{Powersof2}(sk)$$

GSW's Construction - Decryption

$$\text{Flatten}(\mu \cdot I + \text{BitDecomp}(R \cdot pk)) \cdot \text{Powersof2}(sk)$$
$$= (\mu \cdot I + \text{BitDecomp}(R \cdot pk)) \cdot \text{Powersof2}(sk)$$

GSW's Construction - Decryption

$$
\begin{aligned}
&\text{Flatten}(\mu \cdot I + \text{BitDecomp}(R \cdot pk)) \cdot \text{Powersof2}(sk) \\
&= (\mu \cdot I + \text{BitDecomp}(R \cdot pk)) \cdot \text{Powersof2}(sk) \\
&= \mu \text{Powersof2}(sk) + R \cdot pk \cdot sk
\end{aligned}
$$

GSW's Construction - Decryption cont.

$\mu\text{Powersof2}(sk) + R \cdot pk \cdot sk$

- The second term is small.

GSW's Construction - Decryption cont.

$\mu \text{Powersof2}(sk) + R \cdot pk \cdot sk$

- The second term is small.
- The first coordinate of sk is 1.

GSW's Construction - Decryption cont.

$\mu\text{Powersof2}(sk) + R \cdot pk \cdot sk$

- The second term is small.
- The first coordinate of sk is 1.
- $\Rightarrow$ The first coordinates of the first term are $2^{l-1}\mu, 2^{l-2}\mu, \ldots, \mu$

GSW's Construction - Decryption cont.

$\mu\text{Powersof2}(sk) + R \cdot pk \cdot sk$

- The second term is small.
- The first coordinate of *sk* is 1.
- $\Rightarrow$ The first coordinates of the first term are $2^{l-1}\mu, 2^{l-2}\mu, \ldots, \mu$
- Recover μ's least significant bit by $LSB(\mu) = 2^{l-1}\mu$

GSW's Construction - Decryption cont.

$\mu \text{Powersof2}(sk) + R \cdot pk \cdot sk$

- The second term is small.
- The first coordinate of *sk* is 1.
- $\Rightarrow$ The first coordinates of the first term are $2^{l-1}\mu, 2^{l-2}\mu, \ldots, \mu$
- Recover μ's least significant bit by $LSB(\mu) = 2^{l-1}\mu$
- Recover μ's next bit by $2^{l-2}(\mu - LSB(\mu))$

GSW's Construction - Decryption cont.

$\mu\text{Powersof2}(sk) + R \cdot pk \cdot sk$

- The second term is small.
- The first coordinate of sk is 1.
- $\Rightarrow$ The first coordinates of the first term are $2^{l-1}\mu, 2^{l-2}\mu, \ldots, \mu$
- Recover μ's least significant bit by $LSB(\mu) = 2^{l-1}\mu$
- Recover μ's next bit by $2^{l-2}(\mu - LSB(\mu))$
- Similar for all other bits of μ.

GSW's Construction - Decryption cont.

$\mu\text{Powersof2}(sk) + R \cdot pk \cdot sk$

- The second term is small.
- The first coordinate of sk is 1.
- $\Rightarrow$ The first coordinates of the first term are $2^{l-1}\mu, 2^{l-2}\mu, \ldots, \mu$
- Recover μ's least significant bit by $LSB(\mu) = 2^{l-1}\mu$
- Recover μ's next bit by $2^{l-2}(\mu - LSB(\mu))$
- Similar for all other bits of μ.
- Decryption breaks down when the error reaches $q/4$.

GSW's Construction - Security

- $\text{BitDecomp}^{-1}(C) = \mu \cdot \text{BitDecomp}^{-1}(I) + R \cdot A$

GSW's Construction - Security

- $\text{BitDecomp}^{-1}(C) = \mu \cdot \text{BitDecomp}^{-1}(I) + R \cdot A$
- Fact: The joint distribution $(A, R \cdot A)$ is indistinguishable from uniform, if $m > 2nl$

GSW's Construction - Security

- $\text{BitDecomp}^{-1}(C) = \mu \cdot \text{BitDecomp}^{-1}(I) + R \cdot A$
- Fact: The joint distribution $(A, R \cdot A)$ is indistinguishable from uniform, if $m > 2nl$
- $\Rightarrow \text{BitDecomp}^{-1}(C)$ hides μ

GSW's Construction - Security

- $\text{BitDecomp}^{-1}(C) = \mu \cdot \text{BitDecomp}^{-1}(I) + R \cdot A$
- Fact: The joint distribution $(A, R \cdot A)$ is indistinguishable from uniform, if $m > 2nl$
- $\Rightarrow \text{BitDecomp}^{-1}(C)$ hides μ
- $C = \text{Flatten}(C) = \text{BitDecomp} \circ \text{BitDecomp}^{-1}(C)$ hides μ

GSW's Construction - NAND

- $(I - C_1 \cdot C_2)\vec{v} = (1 - \mu_1\mu_2)\vec{v} - \mu_2\vec{e}_1 - C_1\vec{e}_2$

GSW's Construction - NAND

- $(I - C_1 \cdot C_2)\vec{v} = (1 - \mu_1\mu_2)\vec{v} - \mu_2\vec{e}_1 - C_1\vec{e}_2$
- Error increased by a factor of $N + 1$.

GSW's Construction - NAND

- $(I - C_1 \cdot C_2)\vec{v} = (1 - \mu_1\mu_2)\vec{v} - \mu_2\vec{e}_1 - C_1\vec{e}_2$
- Error increased by a factor of $N + 1$.
- Final error increase by a factor of $(N + 1)^L$

GSW's Construction - Multiplying by constant

Input: C, α

- Set $M_\alpha = \text{Flatten}(\alpha I)$

GSW's Construction - Multiplying by constant

Input: C, α

- Set $M_\alpha = \mathrm{Flatten}(\alpha I)$
- Output $\mathrm{Flatten}(M_\alpha \cdot C)$

GSW's Construction - Multiplying by constant

Input: C, α

- Set $M_\alpha = \text{Flatten}(\alpha I)$
- Output $\text{Flatten}(M_\alpha \cdot C)$
- Observe $M_\alpha \cdot C\vec{v} = M_\alpha \cdot (\mu\vec{v} + \vec{e}) = \alpha\mu\vec{v} + M_\alpha \cdot e$

GSW's Construction - Multiplying by constant

Input: C, α

- Set $M_\alpha = \text{Flatten}(\alpha I)$
- Output $\text{Flatten}(M_\alpha \cdot C)$
- Observe $M_\alpha \cdot C\vec{v} = M_\alpha \cdot (\mu\vec{v} + \vec{e}) = \alpha\mu\vec{v} + M_\alpha \cdot e$
- Error increases by a factor of N

GSW's Construction - Addition

- Output Flatten($C_1 + C_2$)

GSW's Construction - Addition

- Output Flatten($C_1 + C_2$)
- Error increases by a factor of 2

GSW's Construction - Multiplication

- Output Flatten$(C_1 C_2)$

GSW's Construction - Multiplication

- Output Flatten($C_1 C_2$)
- $C_1 \cdot C_2\vec{v} = C_1(\mu_2\vec{v} + \vec{e}_2) = \mu_1\mu_2\vec{v} + \mu_2\vec{e}_1 + C_1\vec{e}_2$

GSW's Construction - Multiplication

- Output Flatten($C_1 C_2$)
- $C_1 \cdot C_2 \vec{v} = C_1(\mu_2 \vec{v} + \vec{e}_2) = \mu_1 \mu_2 \vec{v} + \mu_2 \vec{e}_1 + C_1 \vec{e}_2$
- Error increase depends on what's encrypted

GSW's Construction - Multiplication

- Output Flatten$(C_1 C_2)$
- $C_1 \cdot C_2 \vec{v} = C_1(\mu_2 \vec{v} + \vec{e}_2) = \mu_1 \mu_2 \vec{v} + \mu_2 \vec{e}_1 + C_1 \vec{e}_2$
- Error increase depends on what's encrypted
- May need to assume bounds on the values being computed

Choosing parameters

- $\frac{q}{B} > 4(N+1)^L$ for NANDs

Choosing parameters

- $\frac{q}{B} > 4(N+1)^L$ for NANDs
- $\frac{q}{B} > 4(N+T)^L$ for arithmetic circuit, where T is the upper bound on plaintexts

Choosing parameters

- $\frac{q}{B} > 4(N+1)^L$ for NANDs
- $\frac{q}{B} > 4(N+T)^L$ for arithmetic circuit, where T is the upper bound on plaintexts
- N increases linearly with $\log \frac{q}{B}$ for LWE's security

Choosing parameters

- $\frac{q}{B} > 4(N+1)^L$ for NANDs
- $\frac{q}{B} > 4(N+T)^L$ for arithmetic circuit, where T is the upper bound on plaintexts
- N increases linearly with $\log \frac{q}{B}$ for LWE's security
- Pick (q, B, N) accordingly

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.