Learning with Error and GSW's Homomorphic Encryption

June 19, 2019

Motivation

- LWE was an attempt to solve lattice problems

Motivation

- LWE was an attempt to solve lattice problems
- Worst-case lattice problems reduce to LWE

Motivation

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

Motivation

- LWE was an attempt to solve lattice problems
- Worst-case lattice problems reduce to LWE
- Need of new encryption schemes that are at least as hard to break as solving 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

- Assuming $q \in O(poly(n))$ is prime

Search to decision reduction

- Assuming $q \in O(poly(n))$ is prime
- Inductive process; solve variable-by-variable

Search to decision reduction

- Assuming $q \in O(poly(n))$ is prime
- Inductive process; solve variable-by-variable
- Take k, a guess for s_1

Search to decision reduction

- Assuming $q \in O(poly(n))$ is prime
- Inductive process; solve variable-by-variable
- Take k, a guess for s_1
- Pick r uniformly at random

Search to decision reduction

- Assuming $q \in O(poly(n))$ is prime
- Inductive process; solve variable-by-variable
- Take k, a guess for s_1
- Pick r uniformly at random
- Map $(\vec{a}, b)$ to $(\vec{a} + (r, 0, 0, \ldots, 0), b + rk)$

Search to decision reduction

- Assuming $q \in O(poly(n))$ is prime
- Inductive process; solve variable-by-variable
- Take k, a guess for s_1
- Pick r uniformly at random
- Map $(\vec{a}, b)$ to $(\vec{a} + (r, 0, 0, \ldots, 0), b + rk)$
- The above maps $A_{\vec{s},\chi}$ to itself if $k = s_1$, and to the uniform distribution otherwise.

Search to decision reduction

- Assuming $q \in O(poly(n))$ is prime
- Inductive process; solve variable-by-variable
- Take k, a guess for s_1
- Pick r uniformly at random
- Map $(\vec{a}, b)$ to $(\vec{a} + (r, 0, 0, \ldots, 0), b + rk)$
- The above maps $A_{\vec{s},\chi}$ to itself if $k = s_1$, and to the uniform distribution otherwise.
- Check using the blackbox for decision version. Try another k until the guess is correct.

GSW's LWE formulation

- Redefine $\vec{a}_i$ as $b_i||\vec{a}_i$

GSW's LWE formulation

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

GSW's LWE formulation

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

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
- $\mathrm{Enc}_{pk} : \Sigma^* \times \mathcal{R} \to U$
- *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 $\text{Enc}_{pk}(x)$ 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 $\text{Enc}_{pk}(x)$ can evaluate $\text{Enc}_{pk}(f(x))$.

- 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

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 $q = 2^4$.

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

GSW's tools

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

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

GSW's tools

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

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

GSW's tools

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

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

GSW's tools

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

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

GSW's tools (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)

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

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

GSW's tools (Flatten)

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

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

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

GSW's tools (Flatten)

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

GSW's tools (Flatten)

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

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

GSW's tools (Flatten)

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

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

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

GSW's tools cont.

Some basic properties

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

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

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

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 $\mu \text{Powersof2}(sk) + R \cdot pk \cdot sk$ are $\approx \mu, 2\mu, \ldots, 2^{l-2}\mu, 2^{l-1}\mu$

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 $\mu\text{Powersof2}(sk) + R \cdot pk \cdot sk$ are $\approx \mu, 2\mu, \ldots, 2^{l-2}\mu, 2^{l-1}\mu$
- 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 $\mu \text{Powersof2}(sk) + R \cdot pk \cdot sk$ are $\approx \mu, 2\mu, \ldots, 2^{l-2}\mu, 2^{l-1}\mu$
- 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 $\mu\text{Powersof2}(sk) + R \cdot pk \cdot sk$ are $\approx \mu, 2\mu, \ldots, 2^{l-2}\mu, 2^{l-1}\mu$
- 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 $\mu\text{Powersof2}(sk) + R \cdot pk \cdot sk$ are $\approx \mu, 2\mu, \ldots, 2^{l-2}\mu, 2^{l-1}\mu$
- 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

- If $C = \mathsf{Enc}_{pk}(\mu, r)$ hides μ, so does $\mathsf{BitDecomp}^{-1}(C)$, since C can be derived from it.

GSW's Construction - Security

- If $C = \text{Enc}_{pk}(\mu, r)$ hides μ, so does $\text{BitDecomp}^{-1}(C)$, since C can be derived from it.
- $\text{BitDecomp}^{-1}(C) = \mu \cdot \text{BitDecomp}^{-1}(I) + R \cdot A$

GSW's Construction - Security

- If $C = \text{Enc}_{pk}(\mu, r)$ hides μ, so does $\text{BitDecomp}^{-1}(C)$, since C can be derived from it.
- $\text{BitDecomp}^{-1}(C) = \mu \cdot \text{BitDecomp}^{-1}(I) + R \cdot A$
- Fact: The joint distribution $(A, R \cdot A)$ is indistinguishable from uniform, if $m > 2nl$

GSW's Construction - 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 $M_\alpha = \text{Flatten}(\alpha I)$
- Output $\text{Flatten}(M_\alpha \cdot C)$

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

- Simply add the ciphertexts

GSW's Construction - Addition

- Simply add the ciphertexts
- Error increases by a factor of 2

GSW's Construction - Multiplication

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

GSW's Construction - Multiplication

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

GSW's Construction - Multiplication

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

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.