Zero-Knowledge Proof of Decryption for FHE Ciphertexts

Tom Kneiphof

June 28, 2018

Scenario

- Multiple users with *secret* input.
- Compute some function on inputs.
- Everyone should be convinced that the output is indeed correct.
- Inputs must not be revealed!

Scenario

- Multiple users with *secret* input.
- Compute some function on inputs.
- Everyone should be convinced that the output is indeed correct.
- Inputs must not be revealed!

Scenario

- Multiple users with *secret* input.
- Compute some function on inputs.
- Everyone should be convinced that the output is indeed correct.
- Inputs must not be revealed!

Scenario

- Multiple users with *secret* input.
- Compute some function on inputs.
- Everyone should be convinced that the output is indeed correct.
- Inputs must not be revealed!

Multi-Party Computation

- Interactive protocol amongst n parties.
- Perform computation cooperatively (By some protocol).

Problem:

- Everybody must be online.
- Asynchronous setting.
- Large group setting.

Multi-Party Computation

- Interactive protocol amongst n parties.
- Perform computation cooperatively (By some protocol).

Problem:

- Everybody must be online.
- Asynchronous setting.
- Large group setting.

Multi-Party Computation

- Interactive protocol amongst n parties.
- Perform computation cooperatively (By some protocol).

Problem:

- Everybody must be online.
- Asynchronous setting.
- Large group setting.

Multi-Party Computation

- Interactive protocol amongst n parties.
- Perform computation cooperatively (By some protocol).

Problem:

- Everybody must be online.
- Asynchronous setting.
- Large group setting.

Multi-Party Computation

- Interactive protocol amongst n parties.
- Perform computation cooperatively (By some protocol).

Problem:

- Everybody must be online.
- Asynchronous setting.
- Large group setting.

Semi-Trusted Authority

- Authority is trusted to know the secret inputs.
- Authority is *not* trusted to perform correct computations.

- Use fully homomorphic encryption to perform computation on encrypted inputs.
- Get ciphertext of the output.
- Authority decrypts and announces output.
- Authority *proves* correctness of output.
- Secrets must not be revealed!

Semi-Trusted Authority

- Authority is trusted to know the secret inputs.
- Authority is *not* trusted to perform correct computations.

- Use fully homomorphic encryption to perform computation on encrypted inputs.
- Get ciphertext of the output.
- Authority decrypts and announces output.
- Authority *proves* correctness of output.
- Secrets must not be revealed!

Semi-Trusted Authority

- Authority is trusted to know the secret inputs.
- Authority is *not* trusted to perform correct computations.

- Use fully homomorphic encryption to perform computation on encrypted inputs.
- Get ciphertext of the output.
- Authority decrypts and announces output.
- Authority *proves* correctness of output.
- Secrets must not be revealed!

Semi-Trusted Authority

- Authority is trusted to know the secret inputs.
- Authority is *not* trusted to perform correct computations.

- Use fully homomorphic encryption to perform computation on encrypted inputs.
- Get ciphertext of the output.
- Authority decrypts and announces output.
- Authority *proves* correctness of output.
- Secrets must not be revealed!

Semi-Trusted Authority

- Authority is trusted to know the secret inputs.
- Authority is *not* trusted to perform correct computations.

- Use fully homomorphic encryption to perform computation on encrypted inputs.
- Get ciphertext of the output.
- Authority decrypts and announces output.
- Authority *proves* correctness of output.
- Secrets must not be revealed!

Semi-Trusted Authority

- Authority is trusted to know the secret inputs.
- Authority is *not* trusted to perform correct computations.

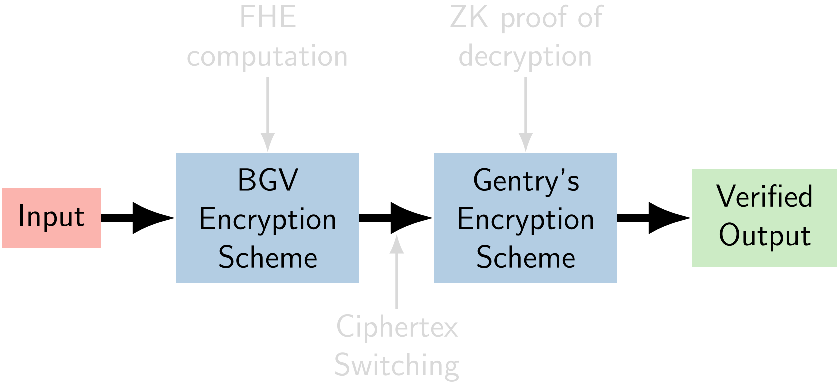
- Use fully homomorphic encryption to perform computation on encrypted inputs.
- Get ciphertext of the output.
- Authority decrypts and announces output.
- Authority *proves* correctness of output.
- Secrets must not be revealed!

Semi-Trusted Authority

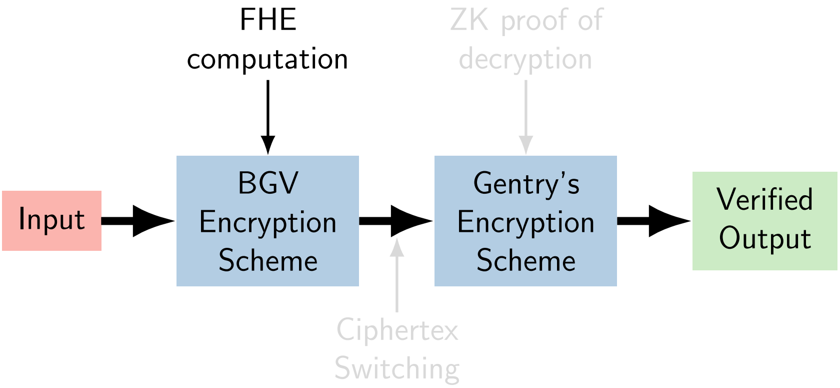
- Authority is trusted to know the secret inputs.
- Authority is *not* trusted to perform correct computations.

- Use fully homomorphic encryption to perform computation on encrypted inputs.
- Get ciphertext of the output.
- Authority decrypts and announces output.
- Authority *proves* correctness of output.
- Secrets must not be revealed!

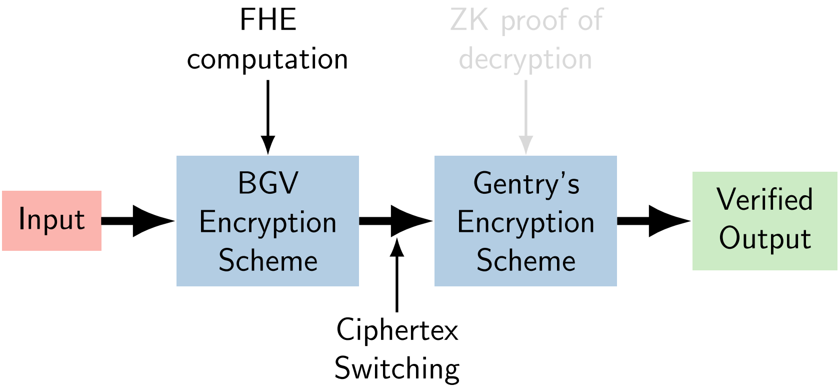
Framework


FHE
computation
ZK proof of
decryption
Input
BGV
Encryption
Scheme
Gentry's
Encryption
Scheme
Verified
Output
Ciphertex
Switching

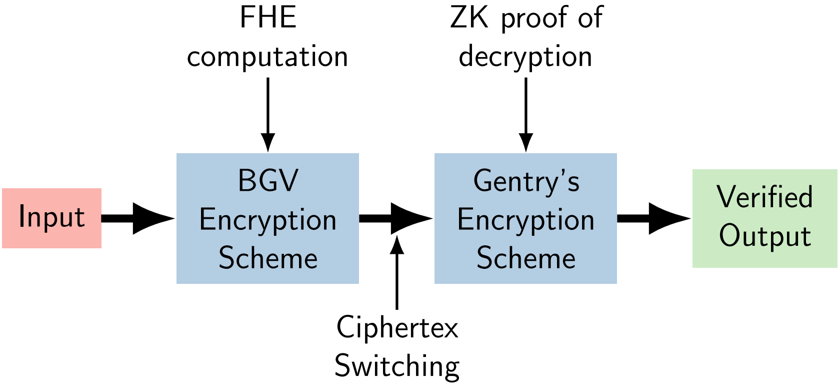
Framework


FHE computation
ZK proof of decryption
Input
BGV Encryption Scheme
Gentry's Encryption Scheme
Verified Output
Ciphertex Switching

Framework


FHE
computation
ZK proof of
decryption
Input
BGV
Encryption
Scheme
Gentry's
Encryption
Scheme
Verified
Output
Ciphertex
Switching

Framework


FHE
computation
ZK proof of
decryption
Input
BGV
Encryption
Scheme
Gentry's
Encryption
Scheme
Verified
Output
Ciphertex
Switching

Fully Homomorphic Encryption

Circuits

- Think of hardware circuits.
- Consist of gates (AND, OR, NAND, ...).
    - Here: Set of gates $\Gamma := \{\cdot, +\}$.
- Only consider functions that can be expressed as circuit of gates in Γ.

Circuits

- Think of hardware circuits.
- Consist of gates (AND, OR, NAND, ...).
  - Here: Set of gates $\Gamma := \{\cdot, +\}$.
- Only consider functions that can be expressed as circuit of gates in Γ.

Circuits

- Think of hardware circuits.
- Consist of gates (AND, OR, NAND, ...).
  - Here: Set of gates $\Gamma := \{\cdot, +\}$.
- Only consider functions that can be expressed as circuit of gates in Γ.

Circuits

- Think of hardware circuits.
- Consist of gates (AND, OR, NAND, ...).
  - Here: Set of gates $\Gamma := \{\cdot, +\}$.
- Only consider functions that can be expressed as circuit of gates in Γ.

Somewhat Homomorphic Encryption (SHE)

- $\text{KeyGen}_{\mathcal{E}}(1^\kappa) \to (sk, pk)$.
- $\text{Encrypt}_{\mathcal{E}}(pk, \pi) \to \psi$.
- $\text{Decrypt}_{\mathcal{E}}(sk, \psi) \to \pi$.

- Set of permitted circuits $\mathcal{C}_{\mathcal{E}}$.
- $\text{Evaluate}_{\mathcal{E}}(pk, C, \psi_1, \ldots \psi_t) \to \psi'$, $C \in \mathcal{C}_{\mathcal{E}}$.

Correctness:

For $C \in \mathcal{C}_{\mathcal{E}}$, plaintexts π_i and their encryption $\psi_i \leftarrow \text{Encrypt}_{\mathcal{E}}(pk, \pi_i)$, $1 \le i \le t$:

$$\psi' \leftarrow \text{Evaluate}_{\mathcal{E}}(pk, C, \psi_1, \ldots, \psi_t) \Rightarrow \text{Decrypt}_{\mathcal{E}}(sk, \psi') = C(\pi_1, \ldots, \pi_t).$$

- Ciphertext size and computation times in $\text{poly}(\kappa)$.

Somewhat Homomorphic Encryption (SHE)

- KeyGen$_{\mathcal{E}}(1^\kappa) \rightarrow (sk, pk)$.
- Encrypt$_{\mathcal{E}}(pk, \pi) \rightarrow \psi$.
- Decrypt$_{\mathcal{E}}(sk, \psi) \rightarrow \pi$.
- Set of permitted circuits $\mathcal{C}_{\mathcal{E}}$.
- Evaluate$_{\mathcal{E}}(pk, C, \psi_1, \ldots \psi_t) \rightarrow \psi'$, $C \in \mathcal{C}_{\mathcal{E}}$.

Correctness:

For $C \in \mathcal{C}_{\mathcal{E}}$, plaintexts π_i and their encryption $\psi_i \leftarrow$ Encrypt$_{\mathcal{E}}(pk, \pi_i)$, $1 \leq i \leq t$:

$$\psi' \leftarrow \text{Evaluate}_{\mathcal{E}}(pk, C, \psi_1, \ldots, \psi_t) \Rightarrow \text{Decrypt}_{\mathcal{E}}(sk, \psi') = C(\pi_1, \ldots, \pi_t).$$

- Ciphertext size and computation times in poly(κ).

Somewhat Homomorphic Encryption (SHE)

- KeyGen$_{\mathcal{E}}(1^\kappa) \to (sk, pk)$.
- Encrypt$_{\mathcal{E}}(pk, \pi) \to \psi$.
- Decrypt$_{\mathcal{E}}(sk, \psi) \to \pi$.
- Set of permitted circuits $\mathcal{C}_{\mathcal{E}}$.
- Evaluate$_{\mathcal{E}}(pk, C, \psi_1, \ldots \psi_t) \to \psi'$, $C \in \mathcal{C}_{\mathcal{E}}$.

Correctness:

For $C \in \mathcal{C}_{\mathcal{E}}$, plaintexts π_i and their encryption $\psi_i \leftarrow$ Encrypt$_{\mathcal{E}}(pk, \pi_i)$, $1 \le i \le t$:

$$\psi' \leftarrow \text{Evaluate}_{\mathcal{E}}(pk, C, \psi_1, \ldots, \psi_t) \Rightarrow \text{Decrypt}_{\mathcal{E}}(sk, \psi') = C(\pi_1, \ldots, \pi_t).$$

- Ciphertext size and computation times in poly(κ).

Somewhat Homomorphic Encryption (SHE)

- $\text{KeyGen}_{\mathcal{E}}(1^{\kappa}) \to (sk, pk)$.
- $\text{Encrypt}_{\mathcal{E}}(pk, \pi) \to \psi$.
- $\text{Decrypt}_{\mathcal{E}}(sk, \psi) \to \pi$.

- Set of permitted circuits $\mathcal{C}_{\mathcal{E}}$.
- $\text{Evaluate}_{\mathcal{E}}(pk, C, \psi_1, \ldots \psi_t) \to \psi'$, $C \in \mathcal{C}_{\mathcal{E}}$.

Correctness:

For $C \in \mathcal{C}_{\mathcal{E}}$, plaintexts π_i and their encryption $\psi_i \leftarrow \text{Encrypt}_{\mathcal{E}}(pk, \pi_i)$, $1 \leq i \leq t$:

$$\psi' \leftarrow \text{Evaluate}_{\mathcal{E}}(pk, C, \psi_1, \ldots, \psi_t) \Rightarrow \text{Decrypt}_{\mathcal{E}}(sk, \psi') = C(\pi_1, \ldots, \pi_t).$$

- Ciphertext size and computation times in $\text{poly}(\kappa)$.

Somewhat Homomorphic Encryption (SHE)

- $\text{KeyGen}_{\mathcal{E}}(1^{\kappa}) \to (sk, pk)$.
- $\text{Encrypt}_{\mathcal{E}}(pk, \pi) \to \psi$.
- $\text{Decrypt}_{\mathcal{E}}(sk, \psi) \to \pi$.

- Set of permitted circuits $\mathcal{C}_{\mathcal{E}}$.
- $\text{Evaluate}_{\mathcal{E}}(pk, C, \psi_1, \ldots \psi_t) \to \psi'$, $C \in \mathcal{C}_{\mathcal{E}}$.

Correctness:

For $C \in \mathcal{C}_{\mathcal{E}}$, plaintexts π_i and their encryption $\psi_i \leftarrow \text{Encrypt}_{\mathcal{E}}(pk, \pi_i)$, $1 \leq i \leq t$:

$$\psi' \leftarrow \text{Evaluate}_{\mathcal{E}}(pk, C, \psi_1, \ldots, \psi_t) \Rightarrow \text{Decrypt}_{\mathcal{E}}(sk, \psi') = C(\pi_1, \ldots, \pi_t).$$

- Ciphertext size and computation times in $\text{poly}(\kappa)$.

Somewhat Homomorphic Encryption (SHE)

- $\mathsf{KeyGen}_{\mathcal{E}}(1^\kappa) \to (sk, pk)$.
- $\mathsf{Encrypt}_{\mathcal{E}}(pk, \pi) \to \psi$.
- $\mathsf{Decrypt}_{\mathcal{E}}(sk, \psi) \to \pi$.

- Set of permitted circuits $\mathcal{C}_{\mathcal{E}}$.
- $\mathsf{Evaluate}_{\mathcal{E}}(pk, C, \psi_1, \ldots \psi_t) \to \psi'$, $C \in \mathcal{C}_{\mathcal{E}}$.

Correctness:

For $C \in \mathcal{C}_{\mathcal{E}}$, plaintexts π_i and their encryption $\psi_i \leftarrow \mathsf{Encrypt}_{\mathcal{E}}(pk, \pi_i)$, $1 \leq i \leq t$:

$$\psi' \leftarrow \mathsf{Evaluate}_{\mathcal{E}}(pk, C, \psi_1, \ldots, \psi_t) \Rightarrow \mathsf{Decrypt}_{\mathcal{E}}(sk, \psi') = C(\pi_1, \ldots, \pi_t).$$

- Ciphertext size and computation times in $\mathsf{poly}(\kappa)$.

Somewhat Homomorphic Encryption (SHE)

- KeyGen$_{\mathcal{E}}(1^\kappa) \to (sk, pk)$.
- Encrypt$_{\mathcal{E}}(pk, \pi) \to \psi$.
- Decrypt$_{\mathcal{E}}(sk, \psi) \to \pi$.

- Set of permitted circuits $\mathcal{C}_{\mathcal{E}}$.
- Evaluate$_{\mathcal{E}}(pk, C, \psi_1, \dots \psi_t) \to \psi'$, $C \in \mathcal{C}_{\mathcal{E}}$.

Correctness:

For $C \in \mathcal{C}_{\mathcal{E}}$, plaintexts π_i and their encryption $\psi_i \leftarrow$ Encrypt$_{\mathcal{E}}(pk, \pi_i)$, $1 \leq i \leq t$:

$$\psi' \leftarrow \text{Evaluate}_{\mathcal{E}}(pk, C, \psi_1, \dots, \psi_t) \Rightarrow \text{Decrypt}_{\mathcal{E}}(sk, \psi') = C(\pi_1, \dots, \pi_t).$$

- Ciphertext size and computation times in poly(κ).

Somewhat Homomorphic Encryption (SHE)

- $\mathsf{KeyGen}_{\mathcal{E}}(1^\kappa) \to (sk, pk)$.
- $\mathsf{Encrypt}_{\mathcal{E}}(pk, \pi) \to \psi$.
- $\mathsf{Decrypt}_{\mathcal{E}}(sk, \psi) \to \pi$.
- Set of permitted circuits $\mathcal{C}_{\mathcal{E}}$.
- $\mathsf{Evaluate}_{\mathcal{E}}(pk, C, \psi_1, \ldots \psi_t) \to \psi'$, $C \in \mathcal{C}_{\mathcal{E}}$.

Correctness:

For $C \in \mathcal{C}_{\mathcal{E}}$, plaintexts π_i and their encryption $\psi_i \leftarrow \mathsf{Encrypt}_{\mathcal{E}}(pk, \pi_i)$, $1 \leq i \leq t$:

$$\psi' \leftarrow \mathsf{Evaluate}_{\mathcal{E}}(pk, C, \psi_1, \ldots, \psi_t) \Rightarrow \mathsf{Decrypt}_{\mathcal{E}}(sk, \psi') = C(\pi_1, \ldots, \pi_t).$$

- Ciphertext size and computation times in $\mathsf{poly}(\kappa)$.

Fully Homomorphic Encryption (FHE)

Leveled Fully Homomorphic Encryption:

- $\mathcal{C}_{\mathcal{E}}$ contains all circuits of a user chosen circuit depth.
- Ciphertext size must be independent of circuit depth.

Fully Homomorphic Encryption:

- $\mathcal{C}_{\mathcal{E}}$ contains *all* circuits.

Fully Homomorphic Encryption (FHE)

Leveled Fully Homomorphic Encryption:

- $\mathcal{C}_\mathcal{E}$ contains all circuits of a user chosen circuit depth.
- Ciphertext size must be independent of circuit depth.

Fully Homomorphic Encryption:

- $\mathcal{C}_\mathcal{E}$ contains *all* circuits.

Fully Homomorphic Encryption (FHE)

Leveled Fully Homomorphic Encryption:

- $\mathcal{C}_{\mathcal{E}}$ contains all circuits of a user chosen circuit depth.
- Ciphertext size must be independent of circuit depth.

Fully Homomorphic Encryption:

- $\mathcal{C}_{\mathcal{E}}$ contains *all* circuits.

Bootstrapping

- The following encryption schemes contain “noise”.
- Can decrypt $\Leftrightarrow$ Noise small.
- Homomorphic operations $\rightarrow$ Noise grows $\rightarrow$ Can't decrypt.
- “Refresh” ciphertext after homomorphic operations.

Basic Idea:

- Encrypt ciphertext under *new* key.
- Evaluate decryption circuit homomorphically.

✓ Create (leveled) FHE scheme from SHE scheme.

Bootstrapping

- The following encryption schemes contain “noise”.
- Can decrypt $\Leftrightarrow$ Noise small.
- Homomorphic operations $\rightarrow$ Noise grows $\rightarrow$ Can’t decrypt.
- “Refresh” ciphertext after homomorphic operations.

Basic Idea:

- Encrypt ciphertext under *new* key.
- Evaluate decryption circuit homomorphically.

✓ Create (leveled) FHE scheme from SHE scheme.

Bootstrapping

- The following encryption schemes contain "noise".
- Can decrypt $\Leftrightarrow$ Noise small.
- Homomorphic operations $\rightarrow$ Noise grows $\rightarrow$ Can't decrypt.
- "Refresh" ciphertext after homomorphic operations.

Basic Idea:

- Encrypt ciphertext under *new* key.
- Evaluate decryption circuit homomorphically.

✓ Create (leveled) FHE scheme from SHE scheme.

Bootstrapping

- The following encryption schemes contain "noise".
- Can decrypt $\Leftrightarrow$ Noise small.
- Homomorphic operations $\rightarrow$ Noise grows $\rightarrow$ Can't decrypt.
- "Refresh" ciphertext after homomorphic operations.

Basic Idea:

- Encrypt ciphertext under *new* key.
- Evaluate decryption circuit homomorphically.

✓ Create (leveled) FHE scheme from SHE scheme.

Bootstrapping

- The following encryption schemes contain "noise".
- Can decrypt $\Leftrightarrow$ Noise small.
- Homomorphic operations $\rightarrow$ Noise grows $\rightarrow$ Can't decrypt.
- "Refresh" ciphertext after homomorphic operations.

Basic Idea:

- Encrypt ciphertext under *new* key.
- Evaluate decryption circuit homomorphically.

✓ Create (leveled) FHE scheme from SHE scheme.

Bootstrapping

- The following encryption schemes contain "noise".
- Can decrypt $\Leftrightarrow$ Noise small.
- Homomorphic operations $\rightarrow$ Noise grows $\rightarrow$ Can't decrypt.
- "Refresh" ciphertext after homomorphic operations.

Basic Idea:

- Encrypt ciphertext under *new* key.
- Evaluate decryption circuit homomorphically.

✓ Create (leveled) FHE scheme from SHE scheme.

Circular Security

Definition:

- SHE scheme $\mathcal{E}$ is circular secure, iff it is IND-CPA given encryptions of it secret key bits.

- Bootstrapping: Encrypt ciphertext under *same* key.
- Don't have to chain secret keys to get leveled FHE from bootstrapping.
- Get FHE scheme from single SHE secret key.

Circular Security

Definition:

- SHE scheme $\mathcal{E}$ is circular secure, iff it is IND-CPA given encryptions of it secret key bits.

- Bootstrapping: Encrypt ciphertext under *same* key.
- Don't have to chain secret keys to get leveled FHE from bootstrapping.
- Get FHE scheme from single SHE secret key.

Circular Security

Definition:

- SHE scheme $\mathcal{E}$ is circular secure, iff it is IND-CPA given encryptions of it secret key bits.

- Bootstrapping: Encrypt ciphertext under *same* key.
- Don't have to chain secret keys to get leveled FHE from bootstrapping.
- Get FHE scheme from single SHE secret key.

Circular Security

Definition:

- SHE scheme $\mathcal{E}$ is circular secure, iff it is IND-CPA given encryptions of it secret key bits.

- Bootstrapping: Encrypt ciphertext under *same* key.
- Don't have to chain secret keys to get leveled FHE from bootstrapping.
- Get FHE scheme from single SHE secret key.

Polynomial Rings

- Common ring $R = \mathbb{Z}[x]/\Phi(x)$.
- $\Phi(x) = x^d + 1$ with $d = 2^\delta$.
- Ring of polynomials with degree at most $d - 1$.
- $x^d \equiv -1 \mod \Phi(x)$.
- For $a \in \mathbb{Z}[x]/\Phi(x)$: coefficient vector $\mathbf{a} \in \mathbb{Z}^d$.
- Polynomial addition = vector addition.
- Multiplication looks similar to complex numbers (for $d = 2$)...

Polynomial Rings

- Common ring $R = \mathbb{Z}[x]/\Phi(x)$.
- $\Phi(x) = x^d + 1$ with $d = 2^\delta$.
- Ring of polynomials with degree at most $d - 1$.
- $x^d \equiv -1 \mod \Phi(x)$.
- For $a \in \mathbb{Z}[x]/\Phi(x)$: coefficient vector $\mathbf{a} \in \mathbb{Z}^d$.
- Polynomial addition = vector addition.
- Multiplication looks similar to complex numbers (for $d = 2$)...

Polynomial Rings

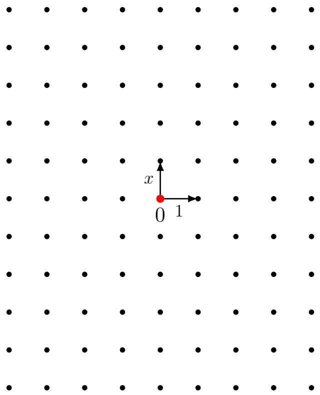
- Common ring $R = \mathbb{Z}[x]/\Phi(x)$.
- $\Phi(x) = x^d + 1$ with $d = 2^\delta$.
- Ring of polynomials with degree at most $d - 1$.
- $x^d \equiv -1 \mod \Phi(x)$.
- For $a \in \mathbb{Z}[x]/\Phi(x)$: coefficient vector $\mathbf{a} \in \mathbb{Z}^d$.
- Polynomial addition = vector addition.
- Multiplication looks similar to complex numbers (for $d = 2$)...

Polynomial Rings

- Common ring $R = \mathbb{Z}[x]/\Phi(x)$.
- $\Phi(x) = x^d + 1$ with $d = 2^\delta$.
- Ring of polynomials with degree at most $d - 1$.
- $x^d \equiv -1 \mod \Phi(x)$.
- For $a \in \mathbb{Z}[x]/\Phi(x)$: coefficient vector $\mathbf{a} \in \mathbb{Z}^d$.
- Polynomial addition = vector addition.
- Multiplication looks similar to complex numbers (for $d = 2$)...

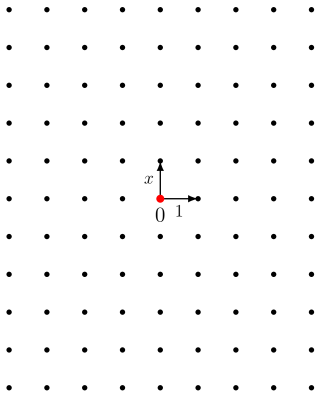
Polynomial Rings

- Common ring $R = \mathbb{Z}[x]/\Phi(x)$.
- $\Phi(x) = x^d + 1$ with $d = 2^\delta$.
- Ring of polynomials with degree at most $d - 1$.
- $x^d \equiv -1 \mod \Phi(x)$.
- For $a \in \mathbb{Z}[x]/\Phi(x)$: coefficient vector $\mathbf{a} \in \mathbb{Z}^d$.
- Polynomial addition $=$ vector addition.
- Multiplication looks similar to complex numbers (for $d = 2$)...



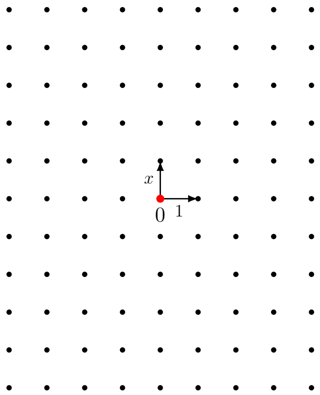
Polynomial Rings

- Common ring $R = \mathbb{Z}[x]/\Phi(x)$.
- $\Phi(x) = x^d + 1$ with $d = 2^\delta$.
- Ring of polynomials with degree at most $d - 1$.
- $x^d \equiv -1 \mod \Phi(x)$.
- For $a \in \mathbb{Z}[x]/\Phi(x)$: coefficient vector $\mathbf{a} \in \mathbb{Z}^d$.
- Polynomial addition = vector addition.
- Multiplication looks similar to complex numbers (for $d = 2$)...



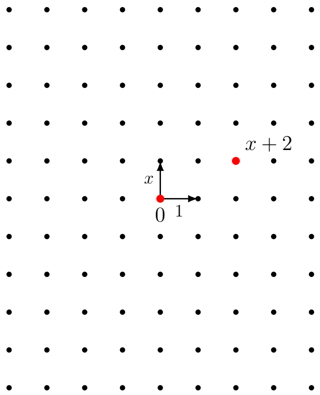
Polynomial Rings

- Common ring $R = \mathbb{Z}[x]/\Phi(x)$.
- $\Phi(x) = x^d + 1$ with $d = 2^\delta$.
- Ring of polynomials with degree at most $d - 1$.
- $x^d \equiv -1 \mod \Phi(x)$.
- For $a \in \mathbb{Z}[x]/\Phi(x)$: coefficient vector $\mathbf{a} \in \mathbb{Z}^d$.
- Polynomial addition $=$ vector addition.
- Multiplication looks similar to complex numbers (for $d = 2$)...



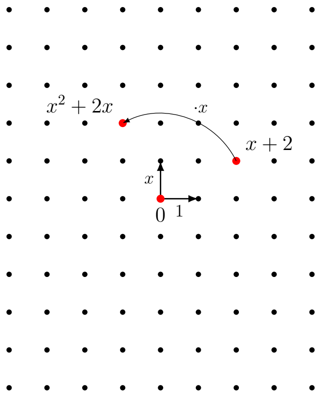
Polynomial Rings

- Common ring $R = \mathbb{Z}[x]/\Phi(x)$.
- $\Phi(x) = x^d + 1$ with $d = 2^\delta$.
- Ring of polynomials with degree at most $d - 1$.
- $x^d \equiv -1 \mod \Phi(x)$.
- For $a \in \mathbb{Z}[x]/\Phi(x)$: coefficient vector $\mathbf{a} \in \mathbb{Z}^d$.
- Polynomial addition = vector addition.
- Multiplication looks similar to complex numbers (for $d = 2$)...



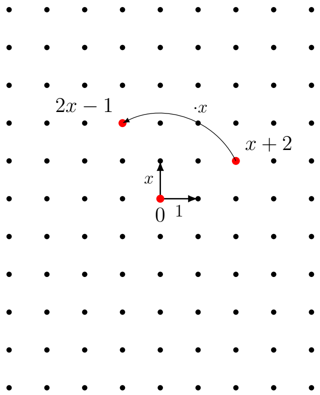
Polynomial Rings

- Common ring $R = \mathbb{Z}[x]/\Phi(x)$.
- $\Phi(x) = x^d + 1$ with $d = 2^\delta$.
- Ring of polynomials with degree at most $d-1$.
- $x^d \equiv -1 \mod \Phi(x)$.
- For $a \in \mathbb{Z}[x]/\Phi(x)$: coefficient vector $\mathbf{a} \in \mathbb{Z}^d$.
- Polynomial addition $=$ vector addition.
- Multiplication looks similar to complex numbers (for $d = 2$)...



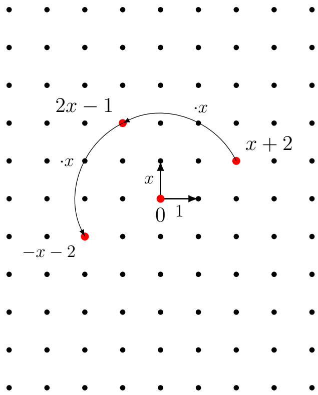
Polynomial Rings

- Common ring $R = \mathbb{Z}[x]/\Phi(x)$.
- $\Phi(x) = x^d + 1$ with $d = 2^\delta$.
- Ring of polynomials with degree at most $d - 1$.
- $x^d \equiv -1 \mod \Phi(x)$.
- For $a \in \mathbb{Z}[x]/\Phi(x)$: coefficient vector $\mathbf{a} \in \mathbb{Z}^d$.
- Polynomial addition = vector addition.
- Multiplication looks similar to complex numbers (for $d = 2$)...



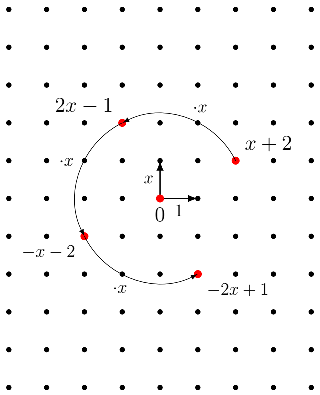
Polynomial Rings

- Common ring $R = \mathbb{Z}[x]/\Phi(x)$.
- $\Phi(x) = x^d + 1$ with $d = 2^\delta$.
- Ring of polynomials with degree at most $d - 1$.
- $x^d \equiv -1 \mod \Phi(x)$.
- For $a \in \mathbb{Z}[x]/\Phi(x)$: coefficient vector $\mathbf{a} \in \mathbb{Z}^d$.
- Polynomial addition = vector addition.
- Multiplication looks similar to complex numbers (for $d = 2$)...



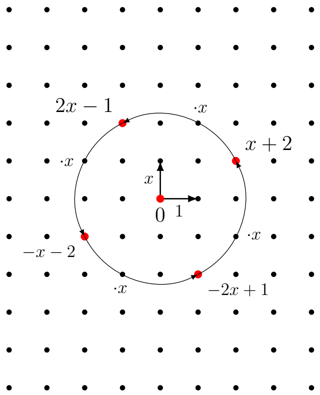
Polynomial Rings

- Common ring $R = \mathbb{Z}[x]/\Phi(x)$.
- $\Phi(x) = x^d + 1$ with $d = 2^\delta$.
- Ring of polynomials with degree at most $d - 1$.
- $x^d \equiv -1 \mod \Phi(x)$.
- For $a \in \mathbb{Z}[x]/\Phi(x)$: coefficient vector $\mathbf{a} \in \mathbb{Z}^d$.
- Polynomial addition $=$ vector addition.
- Multiplication looks similar to complex numbers (for $d = 2$)...



Polynomial Rings

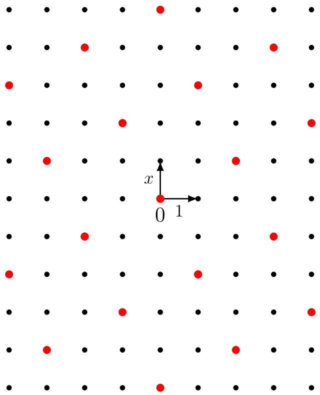
- Common ring $R = \mathbb{Z}[x]/\Phi(x)$.
- $\Phi(x) = x^d + 1$ with $d = 2^\delta$.
- Ring of polynomials with degree at most $d-1$.
- $x^d \equiv -1 \mod \Phi(x)$.
- For $a \in \mathbb{Z}[x]/\Phi(x)$: coefficient vector $\mathbf{a} \in \mathbb{Z}^d$.
- Polynomial addition $=$ vector addition.
- Multiplication looks similar to complex numbers (for $d = 2$)...



Ring Ideals

Definition:

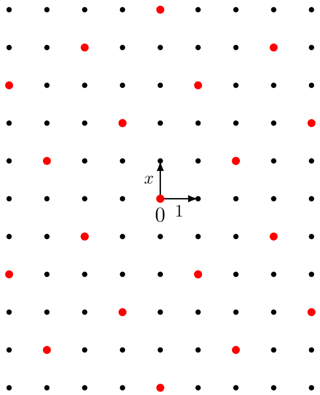
- $I \subset R$ is called *ideal* iff for all $a, b \in I$, $r \in R$:
  - $0 \in I$.
  - $a + b \in I$.
  - $r \cdot a \in I$.



Ring Ideals

Definition:

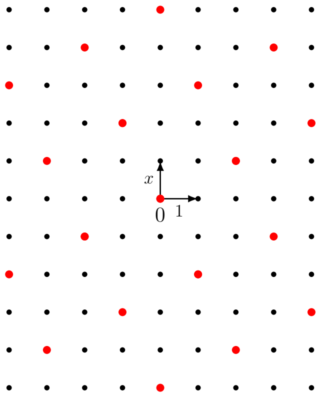
- $I \subset R$ is called *ideal* iff for all $a, b \in I$, $r \in R$:
  - $0 \in I$.
  - $a + b \in I$.
  - $r \cdot a \in I$.



Ring Ideals

Definition:

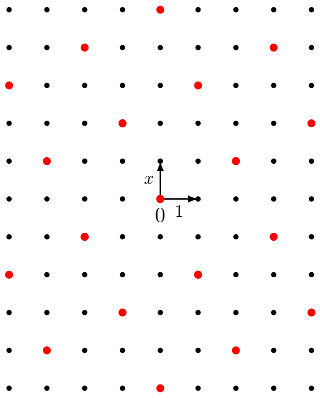
- $I \subset R$ is called *ideal* iff for all $a, b \in I$, $r \in R$:
  - $0 \in I$.
  - $a + b \in I$.
  - $r \cdot a \in I$.



Ring Ideals

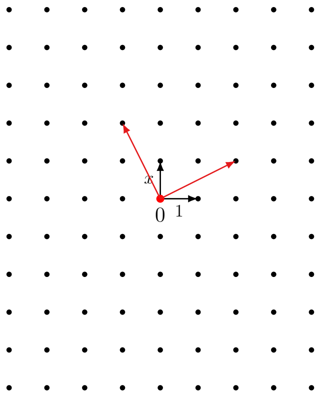
Definition:

- $I \subset R$ is called *ideal* iff for all $a, b \in I$, $r \in R$:
  - $0 \in I$.
  - $a + b \in I$.
  - $r \cdot a \in I$.



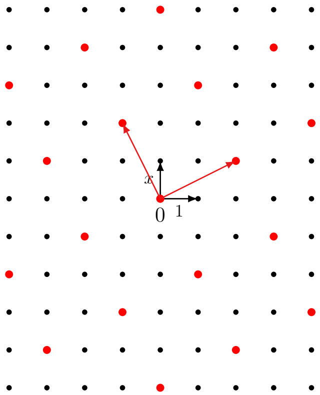
Lattices & Ideal Lattices

- Consider vector space $\mathbb{R}^d$ and some basis $\mathcal{B} \in \mathbb{Z}^{d \times d}$.
- Lattice $L = \mathcal{L}(\mathcal{B})$ is integer linear combination of columns in $\mathcal{B}$.
- Infinite number of lattice bases for $d \geq 2$.
- Ideal lattice: Ring elements corresponding to elements in L form ideal.
- Quotient ring $R/L \Leftrightarrow \mathbb{Z}^d \mod \mathcal{B}$



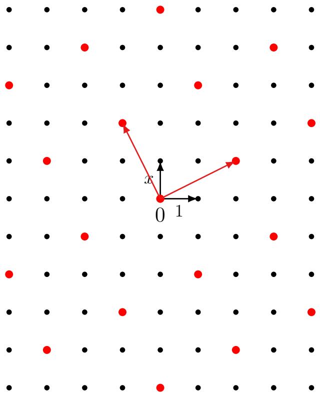
Lattices & Ideal Lattices

- Consider vector space $\mathbb{R}^d$ and some basis $\mathcal{B} \in \mathbb{Z}^{d \times d}$.
- Lattice $L = \mathcal{L}(\mathcal{B})$ is integer linear combination of columns in $\mathcal{B}$.
- Infinite number of lattice bases for $d \geq 2$.
- Ideal lattice: Ring elements corresponding to elements in L form ideal.
- Quotient ring $R/L \Leftrightarrow \mathbb{Z}^d \mod \mathcal{B}$



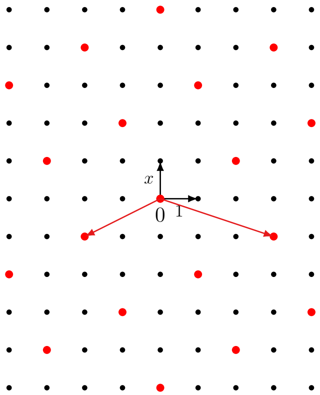
Lattices & Ideal Lattices

- Consider vector space $\mathbb{R}^d$ and some basis $\mathcal{B} \in \mathbb{Z}^{d \times d}$.
- Lattice $L = \mathcal{L}(\mathcal{B})$ is integer linear combination of columns in $\mathcal{B}$.
- Infinite number of lattice bases for $d \geq 2$.
- Ideal lattice: Ring elements corresponding to elements in L form ideal.
- Quotient ring $R/L \Leftrightarrow \mathbb{Z}^d \mod \mathcal{B}$



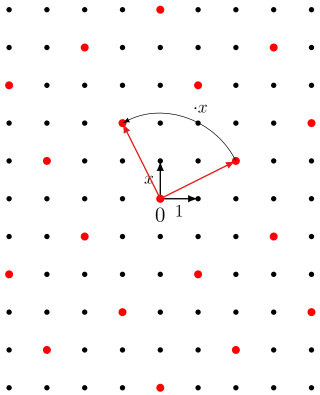
Lattices & Ideal Lattices

- Consider vector space $\mathbb{R}^d$ and some basis $\mathcal{B} \in \mathbb{Z}^{d \times d}$.
- Lattice $L = \mathcal{L}(\mathcal{B})$ is integer linear combination of columns in $\mathcal{B}$.
- Infinite number of lattice bases for $d \geq 2$.
- Ideal lattice: Ring elements corresponding to elements in L form ideal.
- Quotient ring $R/L \Leftrightarrow \mathbb{Z}^d \mod \mathcal{B}$



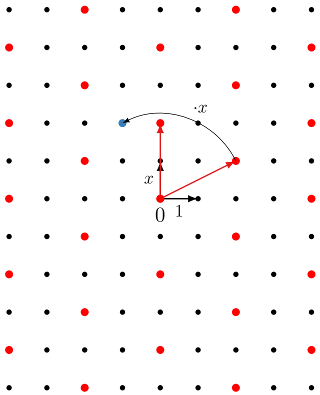
Lattices & Ideal Lattices

- Consider vector space $\mathbb{R}^d$ and some basis $\mathcal{B} \in \mathbb{Z}^{d \times d}$.
- Lattice $L = \mathcal{L}(\mathcal{B})$ is integer linear combination of columns in $\mathcal{B}$.
- Infinite number of lattice bases for $d \geq 2$.
- Ideal lattice: Ring elements corresponding to elements in L form ideal.
- Quotient ring $R/L \Leftrightarrow \mathbb{Z}^d \mod \mathcal{B}$



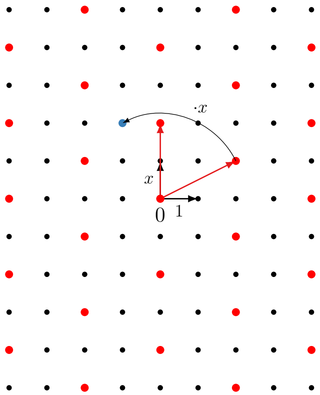
Lattices & Ideal Lattices

- Consider vector space $\mathbb{R}^d$ and some basis $\mathcal{B} \in \mathbb{Z}^{d \times d}$.
- Lattice $L = \mathcal{L}(\mathcal{B})$ is integer linear combination of columns in $\mathcal{B}$.
- Infinite number of lattice bases for $d \geq 2$.
- Ideal lattice: Ring elements corresponding to elements in L form ideal.
- Quotient ring $R/L \Leftrightarrow \mathbb{Z}^d \mod \mathcal{B}$

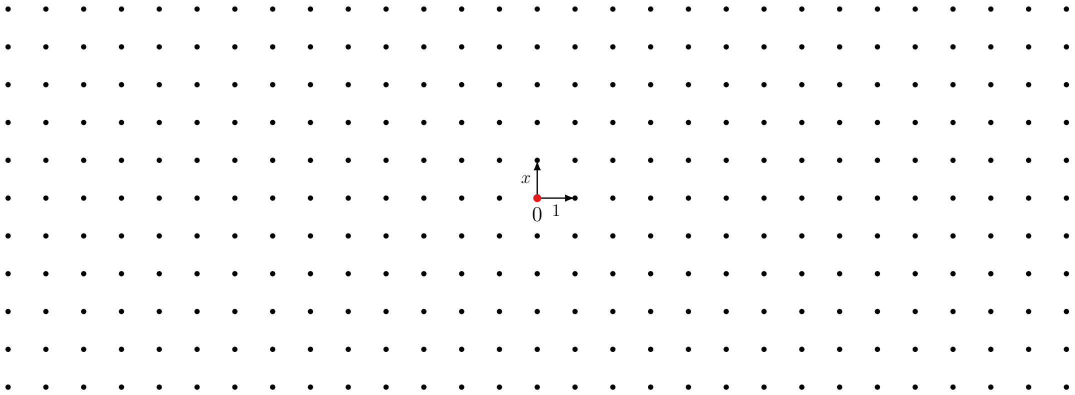


Lattices & Ideal Lattices

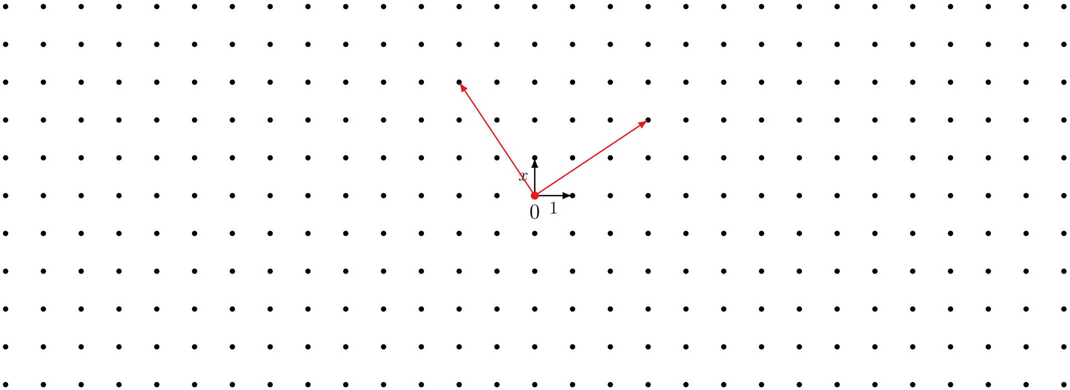
- Consider vector space $\mathbb{R}^d$ and some basis $\mathcal{B} \in \mathbb{Z}^{d \times d}$.
- Lattice $L = \mathcal{L}(\mathcal{B})$ is integer linear combination of columns in $\mathcal{B}$.
- Infinite number of lattice bases for $d \geq 2$.
- Ideal lattice: Ring elements corresponding to elements in L form ideal.
- Quotient ring $R/L \Leftrightarrow \mathbb{Z}^d \mod \mathcal{B}$



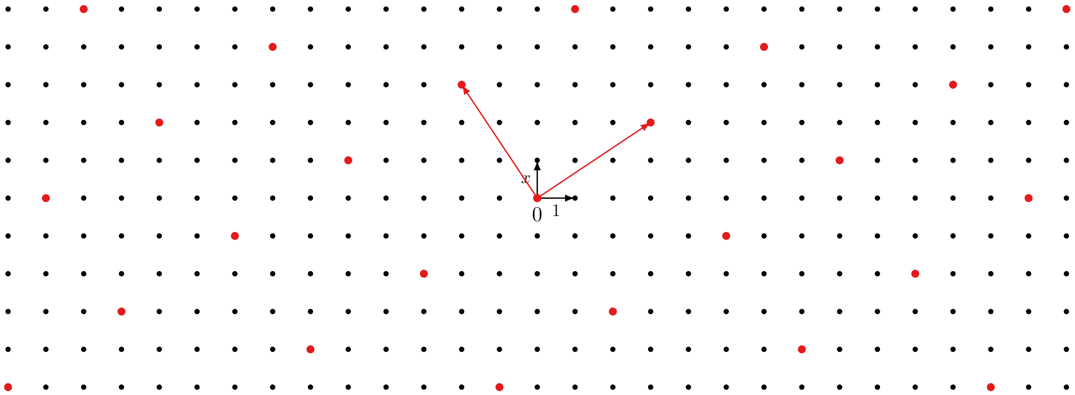
Lattices



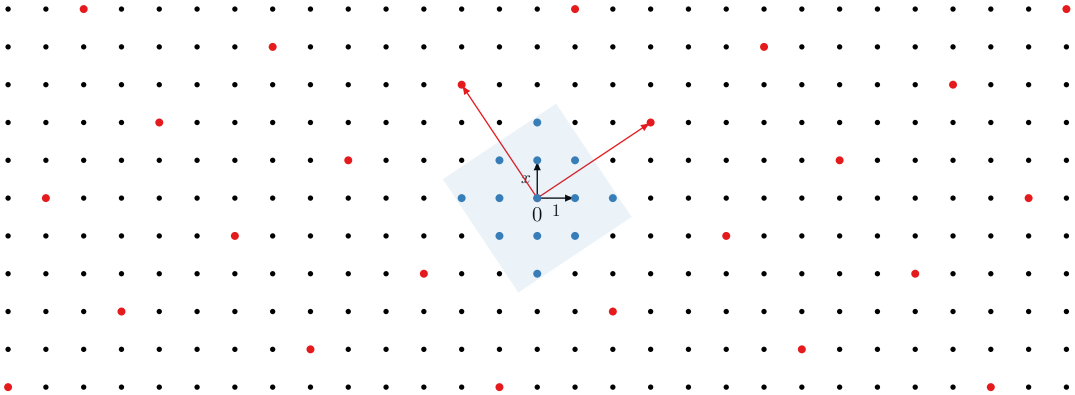
Lattices



Lattices



Lattices



Lattice Basis - Rotation Basis

- Generating element $v \in R$.
- $\mathcal{B}_{\mathsf{Rot}}(v) = \{b_i \in R \mid b_i = v \cdot x^i\}_{i \in \{0,\ldots,d-1\}}$.

Important Feature:

- Basis vectors are (almost) orthogonal.
- $R \mod \mathcal{B}_{\mathsf{Rot}}(v)$ contains a large ball around zero.

Lattice Basis - Rotation Basis

- Generating element $v \in R$.
- $\mathcal{B}_{\mathsf{Rot}}(v) = \{b_i \in R \mid b_i = v \cdot x^i\}_{i \in \{0,\ldots,d-1\}}$.

Important Feature:

- Basis vectors are (almost) orthogonal.
- $R \mod \mathcal{B}_{\mathsf{Rot}}(v)$ contains a large ball around zero.

Lattice Basis - Rotation Basis

- Generating element $v \in R$.
- $\mathcal{B}_{\mathsf{Rot}}(v) = \{b_i \in R \mid b_i = v \cdot x^i\}_{i \in \{0,\ldots,d-1\}}$.

Important Feature:

- Basis vectors are (almost) orthogonal.
- $R \mod \mathcal{B}_{\mathsf{Rot}}(v)$ contains a large ball around zero.

Lattice Basis - Rotation Basis

- Generating element $v \in R$.
- $\mathcal{B}_{\mathsf{Rot}}(v) = \{b_i \in R \mid b_i = v \cdot x^i\}_{i \in \{0,\ldots,d-1\}}$.

Important Feature:

- Basis vectors are (almost) orthogonal.
- $R \mod \mathcal{B}_{\mathsf{Rot}}(v)$ contains a large ball around zero.

Lattice Basis - Hermite Normal Form

Definition:

- A matrix $H \in \mathbb{Z}^{d \times d}$ is in HNF, if it is a non-singular non-negative lower-triangular matrix such that each row has a unique maximum entry, which is on the diagonal.

- HNF can be computed from any basis.
- Unique HNF per lattice.

$\Rightarrow$ HNF is least revealing basis.

Lattice Basis - Hermite Normal Form

Definition:

- A matrix $H \in \mathbb{Z}^{d \times d}$ is in HNF, if it is a non-singular non-negative lower-triangular matrix such that each row has a unique maximum entry, which is on the diagonal.

- HNF can be computed from any basis.
- Unique HNF per lattice.

$\Rightarrow$ HNF is least revealing basis.

Lattice Basis - Hermite Normal Form

Definition:

- A matrix $H \in \mathbb{Z}^{d \times d}$ is in HNF, if it is a non-singular non-negative lower-triangular matrix such that each row has a unique maximum entry, which is on the diagonal.

- HNF can be computed from any basis.
- Unique HNF per lattice.

$\Rightarrow$ HNF is least revealing basis.

Lattice Basis - Hermite Normal Form

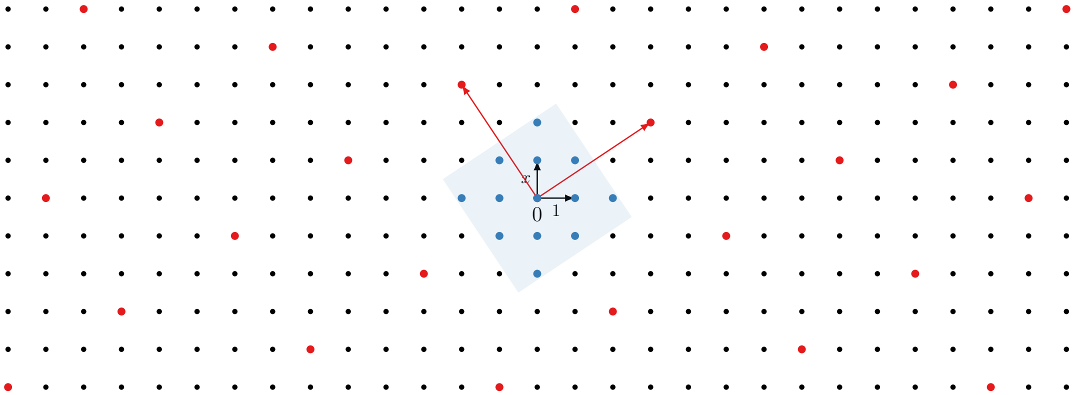
Definition:

- A matrix $H \in \mathbb{Z}^{d \times d}$ is in HNF, if it is a non-singular non-negative lower-triangular matrix such that each row has a unique maximum entry, which is on the diagonal.

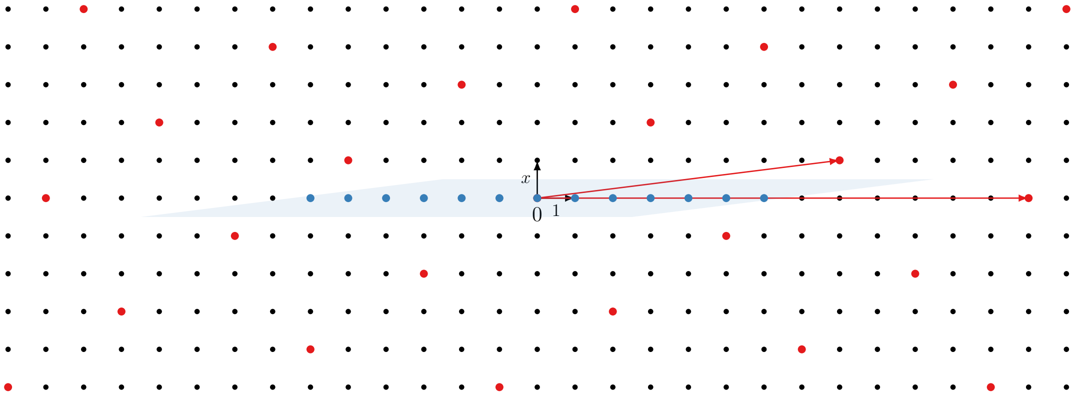
- HNF can be computed from any basis.
- Unique HNF per lattice.

$\Rightarrow$ HNF is least revealing basis.

Rotation Basis vs Hermite Normal Form



Rotation Basis vs Hermite Normal Form



Gentry's Encryption Scheme [Gen09; GH11]

- Two ideals I and J in ring R.
- Ideal $I = 2R$ defines the plaintext space R/I.
- Ideal J defines the ciphertext space R/J.
- A "powerful" basis is used as secret key (rotation basis).
- A "weak" basis is used as public key (HNF).

Gentry's Encryption Scheme [Gen09; GH11]

- Two ideals I and J in ring R.
- Ideal $I = 2R$ defines the plaintext space R/I.
- Ideal J defines the ciphertext space R/J.
- A "powerful" basis is used as secret key (rotation basis).
- A "weak" basis is used as public key (HNF).

Gentry's Encryption Scheme [Gen09; GH11]

- Two ideals I and J in ring R.
- Ideal $I = 2R$ defines the plaintext space R/I.
- Ideal J defines the ciphertext space R/J.
- A "powerful" basis is used as secret key (rotation basis).
- A "weak" basis is used as public key (HNF).

Gentry's Encryption Scheme [Gen09; GH11]

- Two ideals I and J in ring R.
- Ideal $I = 2R$ defines the plaintext space R/I.
- Ideal J defines the ciphertext space R/J.
- A "powerful" basis is used as secret key (rotation basis).
- A "weak" basis is used as public key (HNF).

Gentry's Encryption Scheme [Gen09; GH11]

- Two ideals I and J in ring R.
- Ideal $I = 2R$ defines the plaintext space R/I.
- Ideal J defines the ciphertext space R/J.
- A "powerful" basis is used as secret key (rotation basis).
- A "weak" basis is used as public key (HNF).

Gentry's Encryption Scheme [Gen09; GH11]

KeyGen(1^κ)

- Fix ring R, and basis B_I of ideal $I = 2R$.
- Generate ideal J co-prime to I and two bases (B_J^{sk}, B_J^{pk}).
- Return $pk \leftarrow (R, B_I, B_J^{pk})$ and $sk \leftarrow (R, B_I, B_J^{sk})$.

Encrypt(pk, m)

- Sample noise $rI \in I$ with $r_i \xleftarrow{\$} \{0, \pm 1\}$.
- Return $c \leftarrow m + rI \mod B_J^{pk} = m + 2r + b$ for $b \in J$.

Decrypt(sk, c)

- Return $m \leftarrow (c \mod B_J^{sk}) \mod B_I$.

Gentry's Encryption Scheme [Gen09; GH11]

KeyGen(1^κ)

- Fix ring R, and basis B_I of ideal $I = 2R$.
- Generate ideal J co-prime to I and two bases (B_J^{sk}, B_J^{pk}).
- Return $pk \leftarrow (R, B_I, B_J^{pk})$ and $sk \leftarrow (R, B_I, B_J^{sk})$.

Encrypt(pk, m)

- Sample noise $rI \in I$ with $r_i \xleftarrow{\$} \{0, \pm 1\}$.
- Return $c \leftarrow m + rI \mod B_J^{pk} = m + 2r + b$ for $b \in J$.

Decrypt(sk, c)

- Return $m \leftarrow (c \mod B_J^{sk}) \mod B_I$.

Gentry's Encryption Scheme [Gen09; GH11]

KeyGen(1^κ)

- Fix ring R, and basis B_I of ideal $I = 2R$.
- Generate ideal J co-prime to I and two bases (B_J^{sk}, B_J^{pk}).
- Return $pk \leftarrow (R, B_I, B_J^{pk})$ and $sk \leftarrow (R, B_I, B_J^{sk})$.

Encrypt(pk, m)

- Sample noise $rI \in I$ with $r_i \xleftarrow{\$} \{0, \pm 1\}$.
- Return $c \leftarrow m + rI \mod B_J^{pk} = m + 2r + b$ for $b \in J$.

Decrypt(sk, c)

- Return $m \leftarrow (c \mod B_J^{sk}) \mod B_I$.

Gentry's Encryption Scheme [Gen09; GH11]

KeyGen(1^κ)

- Fix ring R, and basis B_I of ideal $I = 2R$.
- Generate ideal J co-prime to I and two bases (B_J^{sk}, B_J^{pk}).
- Return $pk \leftarrow (R, B_I, B_J^{pk})$ and $sk \leftarrow (R, B_I, B_J^{sk})$.

Encrypt(pk, m)

- Sample noise $rI \in I$ with $r_i \leftarrow \{0, \pm 1\}$.
- Return $c \leftarrow m + rI \mod B_J^{pk} = m + 2r + b$ for $b \in J$.

Decrypt(sk, c)

- Return $m \leftarrow (c \mod B_J^{sk}) \mod B_I$.

Gentry's Encryption Scheme [Gen09; GH11]

KeyGen(1^κ)

- Fix ring R, and basis B_I of ideal $I = 2R$.
- Generate ideal J co-prime to I and two bases (B_J^{sk}, B_J^{pk}).
- Return $pk \leftarrow (R, B_I, B_J^{pk})$ and $sk \leftarrow (R, B_I, B_J^{sk})$.

Encrypt(pk, m)

- Sample noise $rI \in I$ with $r_i \leftarrow \{0, \pm 1\}$.
- Return $c \leftarrow m + rI \mod B_J^{pk} = m + 2r + b$ for $b \in J$.

Decrypt(sk, c)

- Return $m \leftarrow (c \mod B_J^{sk}) \mod B_I$.

Gentry's Encryption Scheme [Gen09; GH11]

KeyGen(1^κ)

- Fix ring R, and basis B_I of ideal $I = 2R$.
- Generate ideal J co-prime to I and two bases (B_J^{sk}, B_J^{pk}).
- Return $pk \leftarrow (R, B_I, B_J^{pk})$ and $sk \leftarrow (R, B_I, B_J^{sk})$.

Encrypt(pk, m)

- Sample noise $rI \in I$ with $r_i \xleftarrow{\$} \{0, \pm 1\}$.
- Return $c \leftarrow m + rI \mod B_J^{pk} = m + 2r + b$ for $b \in J$.

Decrypt(sk, c)

- Return $m \leftarrow (c \mod B_J^{sk}) \mod B_I$.

Gentry's Encryption Scheme - Homomorphic Operations

- Ring operations reflect operations on plaintext.
- $c_1 = m_2 + 2r_1 + b_1$.
- $c_2 = m_2 + 2r_2 + b_2$.
- $c_1 + c_2 = (m_1 + m_2) + 2(r_1 + r_2) + (b_1 + b_2)$.
- $c_1 \cdot c_2 = m_1 \cdot m_2 + 2(r_1 m_2 + r_1 r_2 + r_2 m_1) + b \cdot \ldots$

✓ Can decrypt as long as noise stays small.

Gentry's Encryption Scheme - Homomorphic Operations

- Ring operations reflect operations on plaintext.
- $c_1 = m_2 + 2r_1 + b_1$.
- $c_2 = m_2 + 2r_2 + b_2$.
- $c_1 + c_2 = (m_1 + m_2) + 2(r_1 + r_2) + (b_1 + b_2)$.
- $c_1 \cdot c_2 = m_1 \cdot m_2 + 2(r_1 m_2 + r_1 r_2 + r_2 m_1) + b \cdot \ldots$

✓ Can decrypt as long as noise stays small.

Gentry's Encryption Scheme - Homomorphic Operations

- Ring operations reflect operations on plaintext.
- $c_1 = m_2 + 2r_1 + b_1$.
- $c_2 = m_2 + 2r_2 + b_2$.
- $c_1 + c_2 = (m_1 + m_2) + 2(r_1 + r_2) + (b_1 + b_2)$.
- $c_1 \cdot c_2 = m_1 \cdot m_2 + 2(r_1 m_2 + r_1 r_2 + r_2 m_1) + b \cdot \ldots$

✓ Can decrypt as long as noise stays small.

Gentry's Encryption Scheme - Homomorphic Operations

- Ring operations reflect operations on plaintext.
- $c_1 = m_2 + 2r_1 + b_1$.
- $c_2 = m_2 + 2r_2 + b_2$.
- $c_1 + c_2 = (m_1 + m_2) + 2(r_1 + r_2) + (b_1 + b_2)$.
- $c_1 \cdot c_2 = m_1 \cdot m_2 + 2(r_1 m_2 + r_1 r_2 + r_2 m_1) + b \cdot \ldots$

✓ Can decrypt as long as noise stays small.

Gentry's Encryption Scheme - Homomorphic Operations

- Ring operations reflect operations on plaintext.
- $c_1 = m_2 + 2r_1 + b_1$.
- $c_2 = m_2 + 2r_2 + b_2$.
- $c_1 + c_2 = (m_1 + m_2) + 2(r_1 + r_2) + (b_1 + b_2)$.
- $c_1 \cdot c_2 = m_1 \cdot m_2 + 2(r_1 m_2 + r_1 r_2 + r_2 m_1) + b \cdot \ldots$

✓ Can decrypt as long as noise stays small.

BGV Encryption Scheme (simplified) [BGV14; Car+18]

KeyGen(1^κ)

- Fix ring $R = \mathbb{Z}[x]/\Phi(x)$ as before.
- Pick modulus q and let $R_q = R/qR$.
- Sample $s \xleftarrow{\$} R_2$.
- Sample $B \xleftarrow{\$} R_q$.
- Sample $e \xleftarrow{\$} R_2$.
- $b \leftarrow Bs + 2e$.
- Return $sk \leftarrow \mathbf{s} = (1, s)$, $pk \leftarrow \mathbf{A} = (b, -B)$.
- Note: $\langle \mathbf{A}, \mathbf{s} \rangle = 1 \cdot b - B \cdot s = 2e$.

BGV Encryption Scheme (simplified) [BGV14; Car+18]

KeyGen(1^κ)

- Fix ring $R = \mathbb{Z}[x]/\Phi(x)$ as before.
- Pick modulus q and let $R_q = R/qR$.
- Sample $s \overset{🎲}{\leftarrow} R_2$.
- Sample $B \overset{🎲}{\leftarrow} R_q$.
- Sample $e \overset{🎲}{\leftarrow} R_2$.
- $b \leftarrow Bs + 2e$.
- Return $sk \leftarrow \mathbf{s} = (1, s)$, $pk \leftarrow \mathbf{A} = (b, -B)$.
- Note: $\langle \mathbf{A}, \mathbf{s} \rangle = 1 \cdot b - B \cdot s = 2e$.

BGV Encryption Scheme (simplified) [BGV14; Car+18]

KeyGen(1^κ)

- Fix ring $R = \mathbb{Z}[x]/\Phi(x)$ as before.
- Pick modulus q and let $R_q = R/qR$.
- Sample $s \xleftarrow{\$} R_2$.
- Sample $B \xleftarrow{\$} R_q$.
- Sample $e \xleftarrow{\$} R_2$.
- $b \leftarrow Bs + 2e$.
- Return $sk \leftarrow \mathbf{s} = (1, s)$, $pk \leftarrow \mathbf{A} = (b, -B)$.
- Note: $\langle \mathbf{A}, \mathbf{s} \rangle = 1 \cdot b - B \cdot s = 2e$.

BGV Encryption Scheme (simplified) [BGV14; Car+18]

KeyGen(1^κ)

- Fix ring $R = \mathbb{Z}[x]/\Phi(x)$ as before.
- Pick modulus q and let $R_q = R/qR$.
- Sample $s \xleftarrow{\$} R_2$.
- Sample $B \xleftarrow{\$} R_q$.
- Sample $e \xleftarrow{\$} R_2$.
- $b \leftarrow Bs + 2e$.
- Return $sk \leftarrow \mathbf{s} = (1, s)$, $pk \leftarrow \mathbf{A} = (b, -B)$.
- Note: $\langle \mathbf{A}, \mathbf{s} \rangle = 1 \cdot b - B \cdot s = 2e$.

BGV Encryption Scheme (simplified) [BGV14; Car+18]

KeyGen(1^κ)

- Fix ring $R = \mathbb{Z}[x]/\Phi(x)$ as before.
- Pick modulus q and let $R_q = R/qR$.
- Sample $s \xleftarrow{\$} R_2$.
- Sample $B \xleftarrow{\$} R_q$.
- Sample $e \xleftarrow{\$} R_2$.
- $b \leftarrow Bs + 2e$.
- Return $sk \leftarrow \mathbf{s} = (1, s)$, $pk \leftarrow \mathbf{A} = (b, -B)$.
- Note: $\langle \mathbf{A}, \mathbf{s} \rangle = 1 \cdot b - B \cdot s = 2e$.

BGV Encryption Scheme (simplified) [BGV14; Car+18]

KeyGen(1^κ)

- Fix ring $R = \mathbb{Z}[x]/\Phi(x)$ as before.
- Pick modulus q and let $R_q = R/qR$.
- Sample $s \xleftarrow{\$} R_2$.
- Sample $B \xleftarrow{\$} R_q$.
- Sample $e \xleftarrow{\$} R_2$.
- $b \leftarrow Bs + 2e$.
- Return $sk \leftarrow \mathbf{s} = (1, s)$, $pk \leftarrow \mathbf{A} = (b, -B)$.
- Note: $\langle \mathbf{A}, \mathbf{s} \rangle = 1 \cdot b - B \cdot s = 2e$.

BGV Encryption Scheme (simplified) [BGV14; Car+18]

KeyGen(1^κ)

- Fix ring $R = \mathbb{Z}[x]/\Phi(x)$ as before.
- Pick modulus q and let $R_q = R/qR$.
- Sample $s \xleftarrow{\$} R_2$.
- Sample $B \xleftarrow{\$} R_q$.
- Sample $e \xleftarrow{\$} R_2$.
- $b \leftarrow Bs + 2e$.
- Return $sk \leftarrow \mathbf{s} = (1, s)$, $pk \leftarrow \mathbf{A} = (b, -B)$.
- Note: $\langle \mathbf{A}, \mathbf{s} \rangle = 1 \cdot b - B \cdot s = 2e$.

BGV Encryption Scheme (simplified) [BGV14; Car+18]

KeyGen(1^κ)

- Fix ring $R = \mathbb{Z}[x]/\Phi(x)$ as before.
- Pick modulus q and let $R_q = R/qR$.
- Sample $s \xleftarrow{\$} R_2$.
- Sample $B \xleftarrow{\$} R_q$.
- Sample $e \xleftarrow{\$} R_2$.
- $b \leftarrow Bs + 2e$.
- Return $sk \leftarrow \mathbf{s} = (1, s)$, $pk \leftarrow \mathbf{A} = (b, -B)$.
- Note: $\langle \mathbf{A}, \mathbf{s} \rangle = 1 \cdot b - B \cdot s = 2e$.

BGV Encryption Scheme (simplified) [BGV14; Car+18]

Encrypt(pk, m)

- $\mathbf{m} \leftarrow (m, 0)$.
- $r \xleftarrow{\$} R_2$.
- Return $\mathbf{c} \leftarrow \mathbf{m} + r \cdot \mathbf{A} \in R_q^2$.

Decrypt(sk, $\mathbf{c}$)

- Return $m \leftarrow \langle \mathbf{c}, \mathbf{s} \rangle \mod 2$.
- Note: $\langle \mathbf{c}, \mathbf{s} \rangle = m + r(Bs + 2e) - rBs = m + 2re$.

BGV Encryption Scheme (simplified) [BGV14; Car+18]

Encrypt(pk, m)

- $\mathbf{m} \leftarrow (m, 0)$.
- $r \xleftarrow{\text{dice}} R_2$.
- Return $\mathbf{c} \leftarrow \mathbf{m} + r \cdot \mathbf{A} \in R_q^2$.

Decrypt(sk, $\mathbf{c}$)

- Return $m \leftarrow \langle \mathbf{c}, \mathbf{s} \rangle \mod 2$.
- Note: $\langle \mathbf{c}, \mathbf{s} \rangle = m + r(Bs + 2e) - rBs = m + 2re$.

BGV Encryption Scheme (simplified) [BGV14; Car+18]

Encrypt(pk, m)

- $\mathbf{m} \leftarrow (m, 0)$.
- $r \xleftarrow{\$} R_2$.
- Return $\mathbf{c} \leftarrow \mathbf{m} + r \cdot \mathbf{A} \in R_q^2$.

Decrypt(sk, $\mathbf{c}$)

- Return $m \leftarrow \langle \mathbf{c}, \mathbf{s} \rangle \mod 2$.
- Note: $\langle \mathbf{c}, \mathbf{s} \rangle = m + r(Bs + 2e) - rBs = m + 2re$.

BGV Encryption Scheme (simplified) [BGV14; Car+18]

Encrypt(pk, m)

- $\mathbf{m} \leftarrow (m, 0)$.
- $r \xleftarrow{\$} R_2$.
- Return $\mathbf{c} \leftarrow \mathbf{m} + r \cdot \mathbf{A} \in R_q^2$.

Decrypt(sk, $\mathbf{c}$)

- Return $m \leftarrow \langle \mathbf{c}, \mathbf{s} \rangle \mod 2$.
- Note: $\langle \mathbf{c}, \mathbf{s} \rangle = m + r(Bs + 2e) - rBs = m + 2re$.

BGV Encryption Scheme (simplified) [BGV14; Car+18]

Encrypt(pk, m)

- $\mathbf{m} \leftarrow (m, 0)$.
- $r \overset{🎲}{\leftarrow} R_2$.
- Return $\mathbf{c} \leftarrow \mathbf{m} + r \cdot \mathbf{A} \in R_q^2$.

Decrypt(sk, $\mathbf{c}$)

- Return $m \leftarrow \langle \mathbf{c}, \mathbf{s} \rangle \mod 2$.
- Note: $\langle \mathbf{c}, \mathbf{s} \rangle = m + r(Bs + 2e) - rBs = m + 2re$.

BGV Encryption Scheme - Homomorphic Operations

- Adding two ciphertexts adds their plaintext:

$$\langle \mathbf{c}_1, \mathbf{s} \rangle + \langle \mathbf{c}_2, \mathbf{s} \rangle = \langle \mathbf{c}_1 + \mathbf{c}_2, \mathbf{s} \rangle$$

- Multiplication is more difficult:

$$\langle \mathbf{c}_1, \mathbf{s} \rangle \cdot \langle \mathbf{c}_2, \mathbf{s} \rangle = \mathbf{c}_1^t (\mathbf{s} \oplus \mathbf{s}) \mathbf{c}_2 = \langle \mathbf{c}_1 \oplus \mathbf{c}_2, \mathbf{s} \oplus \mathbf{s} \rangle$$

- “Key switching” (Out of scope)

BGV Encryption Scheme - Homomorphic Operations

- Adding two ciphertexts adds their plaintext:

$$\langle \mathbf{c}_1, \mathbf{s} \rangle + \langle \mathbf{c}_2, \mathbf{s} \rangle = \langle \mathbf{c}_1 + \mathbf{c}_2, \mathbf{s} \rangle$$

- Multiplication is more difficult:

$$\langle \mathbf{c}_1, \mathbf{s} \rangle \cdot \langle \mathbf{c}_2, \mathbf{s} \rangle = \mathbf{c}_1^t (\mathbf{s} \oplus \mathbf{s}) \mathbf{c}_2 = \langle \mathbf{c}_1 \oplus \mathbf{c}_2, \mathbf{s} \oplus \mathbf{s} \rangle$$

- "Key switching" (Out of scope)

BGV Encryption Scheme - Homomorphic Operations

- Adding two ciphertexts adds their plaintext:

$$\langle \mathbf{c}_1, \mathbf{s} \rangle + \langle \mathbf{c}_2, \mathbf{s} \rangle = \langle \mathbf{c}_1 + \mathbf{c}_2, \mathbf{s} \rangle$$

- Multiplication is more difficult:

$$\langle \mathbf{c}_1, \mathbf{s} \rangle \cdot \langle \mathbf{c}_2, \mathbf{s} \rangle = \mathbf{c}_1^t (\mathbf{s} \oplus \mathbf{s}) \mathbf{c}_2 = \langle \mathbf{c}_1 \oplus \mathbf{c}_2, \mathbf{s} \oplus \mathbf{s} \rangle$$

- "Key switching" (Out of scope)

Ciphertext Switching

Switching Ciphertexts [Car+18]

- FHE computation: BGV scheme.
- ZK proof: Gentry's scheme.

Goal:

- Switch ciphertext via bootstrapping-like approach:
  - Encrypt BGV secret key under Gentry.
  - Decrypt homomorphically.

Switching Ciphertexts [Car+18]

- FHE computation: BGV scheme.
- ZK proof: Gentry's scheme.

Goal:

- Switch ciphertext via bootstrapping-like approach:
  - Encrypt BGV secret key under Gentry.
  - Decrypt homomorphically.

Switching Ciphertexts [Car+18]

- FHE computation: BGV scheme.
- ZK proof: Gentry's scheme.

Goal:

- Switch ciphertext via bootstrapping-like approach:
  - Encrypt BGV secret key under Gentry.
  - Decrypt homomorphically.

Switching Ciphertexts [Car+18]

- FHE computation: BGV scheme.
- ZK proof: Gentry's scheme.

Goal:

- Switch ciphertext via bootstrapping-like approach:
  - Encrypt BGV secret key under Gentry.
  - Decrypt homomorphically.

Switching Ciphertexts [Car+18]

- FHE computation: BGV scheme.
- ZK proof: Gentry's scheme.

Goal:

- Switch ciphertext via bootstrapping-like approach:
  - Encrypt BGV secret key under Gentry.
  - Decrypt homomorphically.

Match Ciphertext Spaces

- Ciphertext Spaces:
  - BGV: R_q^2.
  - Gentry: $R \mod B_J^{pk}$.
- Require $qR \subset J$, i.e. $q = B_J^{pk} \cdot t$ with $t \in \mathbb{Z}^d$.
- For $x \in R$ we have: $(x \mod q) \mod B_J^{pk} = x \mod B_J^{pk}$.

Match Ciphertext Spaces

- Ciphertext Spaces:
  - BGV: R_q^2.
  - Gentry: $R \mod B_J^{pk}$.
- Require $qR \subset J$, i.e. $q = B_J^{pk} \cdot t$ with $t \in \mathbb{Z}^d$.
- For $x \in R$ we have: $(x \mod q) \mod B_J^{pk} = x \mod B_J^{pk}$.

Match Ciphertext Spaces

- Ciphertext Spaces:
  - BGV: R_q^2.
  - Gentry: $R \mod B_J^{pk}$.
- Require $qR \subset J$, i.e. $q = B_J^{pk} \cdot t$ with $t \in \mathbb{Z}^d$.
- For $x \in R$ we have: $(x \mod q) \mod B_J^{pk} = x \mod B_J^{pk}$.

Match Ciphertext Spaces

- Ciphertext Spaces:
  - BGV: R_q^2.
  - Gentry: $R \mod B_J^{pk}$.
- Require $qR \subset J$, i.e. $q = B_J^{pk} \cdot t$ with $t \in \mathbb{Z}^d$.
- For $x \in R$ we have: $(x \mod q) \mod B_J^{pk} = x \mod B_J^{pk}$.

Match Ciphertext Spaces

- Ciphertext Spaces:
  - BGV: R_q^2.
  - Gentry: $R \mod B_J^{pk}$.
- Require $qR \subset J$, i.e. $q = B_J^{pk} \cdot t$ with $t \in \mathbb{Z}^d$.
- For $x \in R$ we have: $(x \mod q) \mod B_J^{pk} = x \mod B_J^{pk}$.

Switch BGV Ciphertext to Gentry Ciphertext

Preparation:

- BGV secret key $\mathbf{s} = (1, s) \in R_q^2$.
- $s \in R_2 = R/I$.
- Encrypt secret key $\{s\}_{\mathsf{G}} = s + 2r + b \in (R \mod B_J^{pk})$.

Publicly Available:

- Encrypted BGV secret key $\{s\}_{\mathsf{G}}$.
- BGV ciphertext $\{m\}_{\mathsf{BGV}} = \mathbf{c} = (c_0, c_1) \in R_q^2$ with $\langle \mathbf{c}, \mathbf{s} \rangle = m + 2e$.

- Decrypt BGV ciphertext with encrypted private key!

Switch BGV Ciphertext to Gentry Ciphertext

Preparation:

- BGV secret key $\mathbf{s} = (1, s) \in R_q^2$.
- $s \in R_2 = R/I$.
- Encrypt secret key $\{s\}_{\mathsf{G}} = s + 2r + b \in (R \mod B_J^{pk})$.

Publicly Available:

- Encrypted BGV secret key $\{s\}_{\mathsf{G}}$.
- BGV ciphertext $\{m\}_{\mathsf{BGV}} = \mathbf{c} = (c_0, c_1) \in R_q^2$ with $\langle \mathbf{c}, \mathbf{s} \rangle = m + 2e$.

- Decrypt BGV ciphertext with encrypted private key!

Switch BGV Ciphertext to Gentry Ciphertext

Preparation:

- BGV secret key $\mathbf{s} = (1, s) \in R_q^2$.
- $s \in R_2 = R/I$.
- Encrypt secret key $\{s\}_{\mathsf{G}} = s + 2r + b \in (R \mod B_J^{pk})$.

Publicly Available:

- Encrypted BGV secret key $\{s\}_{\mathsf{G}}$.
- BGV ciphertext $\{m\}_{\mathsf{BGV}} = \mathbf{c} = (c_0, c_1) \in R_q^2$ with $\langle \mathbf{c}, \mathbf{s} \rangle = m + 2e$.

- Decrypt BGV ciphertext with encrypted private key!

Switch BGV Ciphertext to Gentry Ciphertext

Preparation:

- BGV secret key $\mathbf{s} = (1, s) \in R_q^2$.
- $s \in R_2 = R/I$.
- Encrypt secret key $\{s\}_{\mathsf{G}} = s + 2r + b \in (R \mod B_J^{pk})$.

Publicly Available:

- Encrypted BGV secret key $\{s\}_{\mathsf{G}}$.
- BGV ciphertext $\{m\}_{\mathsf{BGV}} = \mathbf{c} = (c_0, c_1) \in R_q^2$ with $\langle \mathbf{c}, \mathbf{s} \rangle = m + 2e$.
- Decrypt BGV ciphertext with encrypted private key!

Switch BGV Ciphertext to Gentry Ciphertext

Preparation:

- BGV secret key $\mathbf{s} = (1, s) \in R_q^2$.
- $s \in R_2 = R/I$.
- Encrypt secret key $\{s\}_{\mathsf{G}} = s + 2r + b \in (R \mod B_J^{pk})$.

Publicly Available:

- Encrypted BGV secret key $\{s\}_{\mathsf{G}}$.
- BGV ciphertext $\{m\}_{\mathsf{BGV}} = \mathbf{c} = (c_0, c_1) \in R_q^2$ with $\langle \mathbf{c}, \mathbf{s} \rangle = m + 2e$.

- Decrypt BGV ciphertext with encrypted private key!

Switch BGV Ciphertext to Gentry Ciphertext

Preparation:

- BGV secret key $\mathbf{s} = (1, s) \in R_q^2$.
- $s \in R_2 = R/I$.
- Encrypt secret key $\{s\}_\mathsf{G} = s + 2r + b \in (R \mod B_J^{pk})$.

Publicly Available:

- Encrypted BGV secret key $\{s\}_\mathsf{G}$.
- BGV ciphertext $\{m\}_\mathsf{BGV} = \mathbf{c} = (c_0, c_1) \in R_q^2$ with $\langle \mathbf{c}, \mathbf{s} \rangle = m + 2e$.

- Decrypt BGV ciphertext with encrypted private key!

Homomorphically Decrypt BGV Ciphertext

$$\langle \{m\}_{\mathsf{BGV}}, \{\mathbf{s}\}_{\mathsf{G}} \rangle \mod B_J^{pk}$$
$$= c_0 + c_1 \cdot \{s\}_{\mathsf{G}}$$
$$= c_0 + c_1 \cdot (s + 2r + b)$$
$$= c_0 + c_1 \cdot s + c_1 \cdot (2r + b)$$
$$= m + 2e + kq + 2c_1 r + c_1 b$$
$$= \underbrace{m + 2(e + c_1 r)}_{\text{Noise}} + \underbrace{(kq + c_1 b)}_{\text{Lattice point } \in J}$$

- $\{m\}_{\mathsf{BGV}} = (c_0, c_1)$.
- $\{s\}_{\mathsf{G}} = s + 2r + b$.
- $c_0 + c_1 \cdot s = m + 2e \mod q$.
- $q \in J$.

✓ Valid ciphertext if $m + 2(e + c_1 r)$ is small enough.

Homomorphically Decrypt BGV Ciphertext

$$\langle \{m\}_{\mathsf{BGV}}, \{\mathbf{s}\}_{\mathsf{G}} \rangle \mod B_J^{pk}$$
$$= c_0 + c_1 \cdot \{s\}_{\mathsf{G}}$$
$$= c_0 + c_1 \cdot (s + 2r + b)$$
$$= c_0 + c_1 \cdot s + c_1 \cdot (2r + b)$$
$$= m + 2e + kq + 2c_1 r + c_1 b$$
$$= \underbrace{m + 2(e + c_1 r)}_{\text{Noise}} + \underbrace{(kq + c_1 b)}_{\text{Lattice point } \in J}$$

- $\{m\}_{\mathsf{BGV}} = (c_0, c_1)$.
- $\{s\}_{\mathsf{G}} = s + 2r + b$.
- $c_0 + c_1 \cdot s = m + 2e \mod q$.
- $q \in J$.

✓ Valid ciphertext if $m + 2(e + c_1 r)$ is small enough.

Homomorphically Decrypt BGV Ciphertext

$$\begin{aligned}
&\langle \{m\}_{\mathsf{BGV}}, \{\mathbf{s}\}_{\mathsf{G}} \rangle \mod B_J^{pk} \\
&= c_0 + c_1 \cdot \{s\}_{\mathsf{G}} \\
&= c_0 + c_1 \cdot (s + 2r + b) \\
&= c_0 + c_1 \cdot s + c_1 \cdot (2r + b) \\
&= m + 2e + kq + 2c_1 r + c_1 b \\
&= \underbrace{m + 2(e + c_1 r)}_{\text{Noise}} + \underbrace{(kq + c_1 b)}_{\text{Lattice point } \in J}
\end{aligned}$$

- $\{m\}_{\mathsf{BGV}} = (c_0, c_1)$.
- $\{s\}_{\mathsf{G}} = s + 2r + b$.
- $c_0 + c_1 \cdot s = m + 2e \mod q$.
- $q \in J$.

✓ Valid ciphertext if $m + 2(e + c_1 r)$ is small enough.

Homomorphically Decrypt BGV Ciphertext

$$\begin{aligned}
&\langle \{m\}_{\mathsf{BGV}}, \{\mathbf{s}\}_{\mathsf{G}} \rangle \mod B_J^{pk} \\
&= c_0 + c_1 \cdot \{s\}_{\mathsf{G}} \\
&= c_0 + c_1 \cdot (s + 2r + b) \\
&= c_0 + c_1 \cdot s + c_1 \cdot (2r + b) \\
&= m + 2e + kq + 2c_1 r + c_1 b \\
&= \underbrace{m + 2(e + c_1 r)}_{\text{Noise}} + \underbrace{(kq + c_1 b)}_{\text{Lattice point } \in J}
\end{aligned}$$

- $\{m\}_{\mathsf{BGV}} = (c_0, c_1)$.
- $\{s\}_{\mathsf{G}} = s + 2r + b$.
- $c_0 + c_1 \cdot s = m + 2e \mod q$.
- $q \in J$.

✓ Valid ciphertext if $m + 2(e + c_1 r)$ is small enough.

Homomorphically Decrypt BGV Ciphertext

$$
\begin{aligned}
&\langle \{m\}_{\mathsf{BGV}}, \{\mathbf{s}\}_{\mathsf{G}} \rangle \mod B_J^{pk} \\
&= c_0 + c_1 \cdot \{s\}_{\mathsf{G}} \\
&= c_0 + c_1 \cdot (s + 2r + b) \\
&= c_0 + c_1 \cdot s + c_1 \cdot (2r + b) \\
&= m + 2e + kq + 2c_1 r + c_1 b \\
&= \underbrace{m + 2(e + c_1 r)}_{\text{Noise}} + \underbrace{(kq + c_1 b)}_{\text{Lattice point } \in J}
\end{aligned}
$$

- $\{m\}_{\mathsf{BGV}} = (c_0, c_1)$.
- $\{s\}_{\mathsf{G}} = s + 2r + b$.
- $c_0 + c_1 \cdot s = m + 2e \mod q$.
- $q \in J$.

✓ Valid ciphertext if $m + 2(e + c_1 r)$ is small enough.

Homomorphically Decrypt BGV Ciphertext

$$
\begin{aligned}
&\langle \{m\}_{\mathsf{BGV}}, \{\mathbf{s}\}_{\mathsf{G}} \rangle \mod B_J^{pk} \\
&= c_0 + c_1 \cdot \{s\}_{\mathsf{G}} \\
&= c_0 + c_1 \cdot (s + 2r + b) \\
&= c_0 + c_1 \cdot s + c_1 \cdot (2r + b) \\
&= m + 2e + kq + 2c_1 r + c_1 b \\
&= \underbrace{m + 2(e + c_1 r)}_{\text{Noise}} + \underbrace{(kq + c_1 b)}_{\text{Lattice point } \in J}
\end{aligned}
$$

- $\{m\}_{\mathsf{BGV}} = (c_0, c_1)$.
- $\{s\}_{\mathsf{G}} = s + 2r + b$.
- $c_0 + c_1 \cdot s = m + 2e \mod q$.
- $q \in J$.

✓ Valid ciphertext if $m + 2(e + c_1 r)$ is small enough.

Homomorphically Decrypt BGV Ciphertext

$$
\begin{aligned}
&\langle \{m\}_{\mathsf{BGV}}, \{\mathbf{s}\}_{\mathsf{G}} \rangle \mod B_J^{pk} \\
&= c_0 + c_1 \cdot \{s\}_{\mathsf{G}} \\
&= c_0 + c_1 \cdot (s + 2r + b) \\
&= c_0 + c_1 \cdot s + c_1 \cdot (2r + b) \\
&= m + 2e + kq + 2c_1 r + c_1 b \\
&= \underbrace{m + 2(e + c_1 r)}_{\text{Noise}} + \underbrace{(kq + c_1 b)}_{\text{Lattice point } \in J}
\end{aligned}
$$

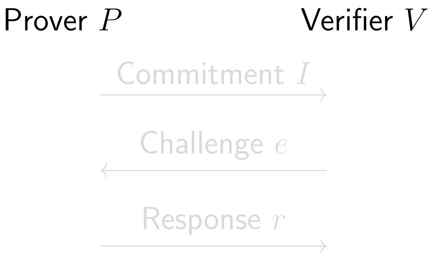
- $\{m\}_{\mathsf{BGV}} = (c_0, c_1)$.
- $\{s\}_{\mathsf{G}} = s + 2r + b$.
- $c_0 + c_1 \cdot s = m + 2e \mod q$.
- $q \in J$.

✓ Valid ciphertext if $m + 2(e + c_1 r)$ is small enough.

Zero-Knowledge Proof of Decryption

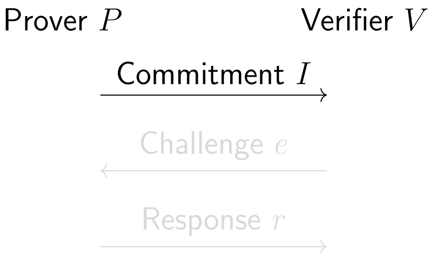
Sigma Protocol

- Prover P and Verifier V.
- P sends commitment I.
- V sends challenge e.
- P sends response r.
- V verifies.



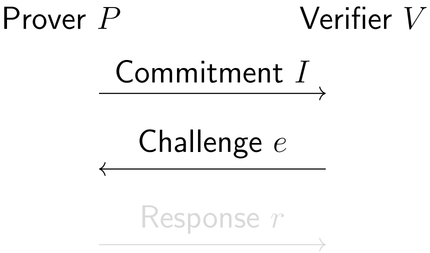
Sigma Protocol

- Prover P and Verifier V.
- P sends commitment I.
- V sends challenge e.
- P sends response r.
- V verifies.



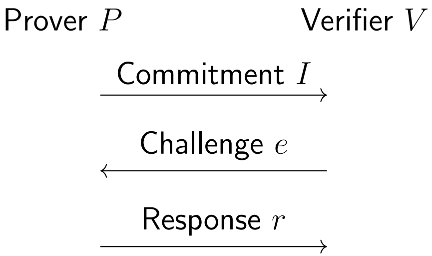
Sigma Protocol

- Prover P and Verifier V.
- P sends commitment I.
- V sends challenge e.
- P sends response r.
- V verifies.



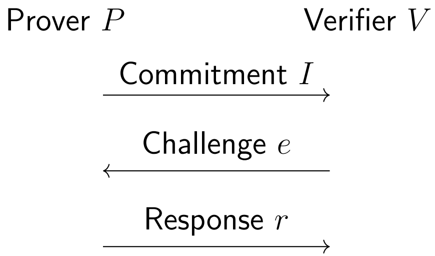
Sigma Protocol

- Prover P and Verifier V.
- P sends commitment I.
- V sends challenge e.
- P sends response r.
- V verifies.



Sigma Protocol

- Prover P and Verifier V.
- P sends commitment I.
- V sends challenge e.
- P sends response r.
- V verifies.



(Wanted) Properties

Correctness:

- Can a true statement be proven?

Special Soundness:

- Given two transcripts (I, e_0, r_0) and (I, e_1, r_1).
- Can we compute the secret?

Special Honest Verifier Zero-Knowledge:

- Given challenge e.
- Can transcripts be generated without knowledge of secret?

(Wanted) Properties

Correctness:

- Can a true statement be proven?

Special Soundness:

- Given two transcripts (I, e_0, r_0) and (I, e_1, r_1).
- Can we compute the secret?

Special Honest Verifier Zero-Knowledge:

- Given challenge e.
- Can transcripts be generated without knowledge of secret?

(Wanted) Properties

Correctness:

- Can a true statement be proven?

Special Soundness:

- Given two transcripts (I, e_0, r_0) and (I, e_1, r_1).
- Can we compute the secret?

Special Honest Verifier Zero-Knowledge:

- Given challenge e.
- Can transcripts be generated without knowledge of secret?

The ZK Protocol [Car+18]

Statement: A given ciphertext $c = m + 2r + b$ is an encryption of 0.

- P Choose encryption $c' = 2r' + b'$ of 0. Send c' to the verifier.
- V Choose challenge $e \xleftarrow{\$} \{0, 1\}$ uniformly at random. Send e to the prover.
- P Compute response $d \leftarrow e \cdot b + b'$. Send d to the verifier.
- V Verify that d is a valid lattice point, and check that $e \cdot c + c' - d$ is well formed and sufficiently small.

Transcript: (c', e, d).

The ZK Protocol [Car+18]

Statement: A given ciphertext $c = m + 2r + b$ is an encryption of 0.

- P Choose encryption $c' = 2r' + b'$ of 0. Send c' to the verifier.
- V Choose challenge $e \xleftarrow{\$} \{0, 1\}$ uniformly at random. Send e to the prover.
- P Compute response $d \leftarrow e \cdot b + b'$. Send d to the verifier.
- V Verify that d is a valid lattice point, and check that $e \cdot c + c' - d$ is well formed and sufficiently small.

Transcript: (c', e, d).

The ZK Protocol [Car+18]

Statement: A given ciphertext $c = m + 2r + b$ is an encryption of 0.

- P Choose encryption $c' = 2r' + b'$ of 0. Send c' to the verifier.
- V Choose challenge $e \xleftarrow{\$} \{0, 1\}$ uniformly at random. Send e to the prover.
- P Compute response $d \leftarrow e \cdot b + b'$. Send d to the verifier.
- V Verify that d is a valid lattice point, and check that $e \cdot c + c' - d$ is well formed and sufficiently small.

Transcript: (c', e, d).

The ZK Protocol [Car+18]

Statement: A given ciphertext $c = m + 2r + b$ is an encryption of 0.

- P Choose encryption $c' = 2r' + b'$ of 0. Send c' to the verifier.
- V Choose challenge $e \xleftarrow{\$} \{0, 1\}$ uniformly at random. Send e to the prover.
- P Compute response $d \leftarrow e \cdot b + b'$. Send d to the verifier.
- V Verify that d is a valid lattice point, and check that $e \cdot c + c' - d$ is well formed and sufficiently small.

Transcript: (c', e, d).

The ZK Protocol [Car+18]

Statement: A given ciphertext $c = m + 2r + b$ is an encryption of 0.

- P Choose encryption $c' = 2r' + b'$ of 0. Send c' to the verifier.
- V Choose challenge $e \xleftarrow{\$} \{0, 1\}$ uniformly at random. Send e to the prover.
- P Compute response $d \leftarrow e \cdot b + b'$. Send d to the verifier.
- V Verify that d is a valid lattice point, and check that $e \cdot c + c' - d$ is well formed and sufficiently small.

Transcript: (c', e, d).

The ZK Protocol [Car+18]

Statement: A given ciphertext $c = m + 2r + b$ is an encryption of 0.

- **P** Choose encryption $c' = 2r' + b'$ of 0. Send c' to the verifier.
- **V** Choose challenge $e \xleftarrow{\$} \{0, 1\}$ uniformly at random. Send e to the prover.
- **P** Compute response $d \leftarrow e \cdot b + b'$. Send d to the verifier.
- **V** Verify that d is a valid lattice point, and check that $e \cdot c + c' - d$ is well formed and sufficiently small.

Transcript: (c', e, d).

Correctness

If the statement is correct, then V verifies:

- ✓ $d \in J$ is a valid lattice point. By definition, b and b' are lattice points.
- ✓ $e \cdot c + c' - d$ is well formed and sufficiently small. This is $2(e \cdot r + r')$, which is the noise vector of $e \cdot c + c'$.

Correctness

If the statement is correct, then V verifies:

- ✓ $d \in J$ is a valid lattice point. By definition, b and b' are lattice points.
- ✓ $e \cdot c + c' - d$ is well formed and sufficiently small. This is $2(e \cdot r + r')$, which is the noise vector of $e \cdot c + c'$.

Correctness

If the statement is correct, then V verifies:

- ✓ $d \in J$ is a valid lattice point. By definition, b and b' are lattice points.
- ✓ $e \cdot c + c' - d$ is well formed and sufficiently small. This is $2(e \cdot r + r')$, which is the noise vector of $e \cdot c + c'$.

Special Soundness

- If we know $b \in J$ from $c = m + 2r + b$, we can get m.

$\Rightarrow$ b is a *witness* for the statement we want to prove.

Given two transcripts with same commitment: (c', e_0, d_0) and (c', e_1, d_1).

$$
\begin{aligned}
&(e_1 - e_0)^{-1} \cdot (d_1 - d_0) \\
&= (e_1 - e_0)^{-1} \cdot (e_1 b + b' - e_0 b - b') \\
&= (e_1 - e_0)^{-1} \cdot (e_1 - e_0) \cdot b \\
&= b
\end{aligned}
$$

✓ Witness b can be computed.

Special Soundness

- If we know $b \in J$ from $c = m + 2r + b$, we can get m.

$\Rightarrow$ b is a *witness* for the statement we want to prove.

Given two transcripts with same commitment: (c', e_0, d_0) and (c', e_1, d_1).

$$
\begin{aligned}
&(e_1 - e_0)^{-1} \cdot (d_1 - d_0) \\
&= (e_1 - e_0)^{-1} \cdot (e_1 b + b' - e_0 b - b') \\
&= (e_1 - e_0)^{-1} \cdot (e_1 - e_0) \cdot b \\
&= b
\end{aligned}
$$

✓ Witness b can be computed.

Special Soundness

- If we know $b \in J$ from $c = m + 2r + b$, we can get m.

$\Rightarrow$ b is a *witness* for the statement we want to prove.

Given two transcripts with same commitment: (c', e_0, d_0) and (c', e_1, d_1).

$$
\begin{aligned}
&(e_1 - e_0)^{-1} \cdot (d_1 - d_0) \\
&= (e_1 - e_0)^{-1} \cdot (e_1 b + b' - e_0 b - b') \\
&= (e_1 - e_0)^{-1} \cdot (e_1 - e_0) \cdot b \\
&= b
\end{aligned}
$$

✓ Witness b can be computed.

Special Soundness

- If we know $b \in J$ from $c = m + 2r + b$, we can get m.

$\Rightarrow$ b is a *witness* for the statement we want to prove.

Given two transcripts with same commitment: (c', e_0, d_0) and (c', e_1, d_1).

$$
\begin{aligned}
&(e_1 - e_0)^{-1} \cdot (d_1 - d_0) \\
&= (e_1 - e_0)^{-1} \cdot (e_1 b + b' - e_0 b - b') \\
&= (e_1 - e_0)^{-1} \cdot (e_1 - e_0) \cdot b \\
&= b
\end{aligned}
$$

✓ Witness b can be computed.

Special Soundness

- If we know $b \in J$ from $c = m + 2r + b$, we can get m.

$\Rightarrow$ b is a *witness* for the statement we want to prove.

Given two transcripts with same commitment: (c', e_0, d_0) and (c', e_1, d_1).

$$
\begin{aligned}
&(e_1 - e_0)^{-1} \cdot (d_1 - d_0) \\
&= (e_1 - e_0)^{-1} \cdot (e_1 b + b' - e_0 b - b') \\
&= (e_1 - e_0)^{-1} \cdot (e_1 - e_0) \cdot b \\
&= b
\end{aligned}
$$

✓ Witness b can be computed.

Special Soundness

- If we know $b \in J$ from $c = m + 2r + b$, we can get m.

$\Rightarrow$ b is a *witness* for the statement we want to prove.

Given two transcripts with same commitment: (c', e_0, d_0) and (c', e_1, d_1).

$$\begin{aligned}
&(e_1 - e_0)^{-1} \cdot (d_1 - d_0) \\
&= (e_1 - e_0)^{-1} \cdot (e_1 b + b' - e_0 b - b') \\
&= (e_1 - e_0)^{-1} \cdot (e_1 - e_0) \cdot b \\
&= b
\end{aligned}$$

✓ Witness b can be computed.

Special Soundness

- If we know $b \in J$ from $c = m + 2r + b$, we can get m.

$\Rightarrow$ b is a *witness* for the statement we want to prove.

Given two transcripts with same commitment: (c', e_0, d_0) and (c', e_1, d_1).

$$\begin{aligned} & (e_1 - e_0)^{-1} \cdot (d_1 - d_0) \\ = & (e_1 - e_0)^{-1} \cdot (e_1 b + b' - e_0 b - b') \\ = & (e_1 - e_0)^{-1} \cdot (e_1 - e_0) \cdot b \\ = & b \end{aligned}$$

✓ Witness b can be computed.

Special Soundness

- If we know $b \in J$ from $c = m + 2r + b$, we can get m.

⇒ b is a *witness* for the statement we want to prove.

Given two transcripts with same commitment: (c', e_0, d_0) and (c', e_1, d_1).

$$\begin{aligned}
&(e_1 - e_0)^{-1} \cdot (d_1 - d_0) \\
&= (e_1 - e_0)^{-1} \cdot (e_1 b + b' - e_0 b - b') \\
&= (e_1 - e_0)^{-1} \cdot (e_1 - e_0) \cdot b \\
&= b
\end{aligned}$$

✓ Witness b can be computed.

Special Honest-Verifier Zero-Knowledge

- Honest verifier should not learn anything from an execution of the protocol.
- I.e. Simulator exists, that generates transcripts for arbitrary challenges e.

Simulator(c, e):

- Sample* random noise vector $\hat{r}$.
- Compute lattice point $d \in J$ corresponding to $2\hat{r}$. I.e. $\hat{c} = 2\hat{r} + d$.
- $c' \leftarrow \hat{c} - e \cdot c$.
- Output transcript (c', e, d).

✓ Transcript is valid. In particular $e \cdot c + c' - d = 2\hat{r}$ is well-formed noise.

✓ Honest verifier does not learn anything about b.

Special Honest-Verifier Zero-Knowledge

- Honest verifier should not learn anything from an execution of the protocol.
- I.e. Simulator exists, that generates transcripts for arbitrary challenges e.

Simulator(c, e):

- Sample* random noise vector $\hat{r}$.
- Compute lattice point $d \in J$ corresponding to $2\hat{r}$. I.e. $\hat{c} = 2\hat{r} + d$.
- $c' \leftarrow \hat{c} - e \cdot c$.
- Output transcript (c', e, d).

✓ Transcript is valid. In particular $e \cdot c + c' - d = 2\hat{r}$ is well-formed noise.

✓ Honest verifier does not learn anything about b.

Special Honest-Verifier Zero-Knowledge

- Honest verifier should not learn anything from an execution of the protocol.
- I.e. Simulator exists, that generates transcripts for arbitrary challenges e.

Simulator(c, e):

- Sample* random noise vector $\hat{r}$.
- Compute lattice point $d \in J$ corresponding to $2\hat{r}$. I.e. $\hat{c} = 2\hat{r} + d$.
- $c' \leftarrow \hat{c} - e \cdot c$.
- Output transcript (c', e, d).

✓ Transcript is valid. In particular $e \cdot c + c' - d = 2\hat{r}$ is well-formed noise.

✓ Honest verifier does not learn anything about b.

Special Honest-Verifier Zero-Knowledge

- Honest verifier should not learn anything from an execution of the protocol.
- I.e. Simulator exists, that generates transcripts for arbitrary challenges e.

Simulator(c, e):

- Sample* random noise vector $\hat{r}$.
- Compute lattice point $d \in J$ corresponding to $2\hat{r}$. I.e. $\hat{c} = 2\hat{r} + d$.
- $c' \leftarrow \hat{c} - e \cdot c$.
- Output transcript (c', e, d).

✓ Transcript is valid. In particular $e \cdot c + c' - d = 2\hat{r}$ is well-formed noise.

✓ Honest verifier does not learn anything about b.

Special Honest-Verifier Zero-Knowledge

- Honest verifier should not learn anything from an execution of the protocol.
- I.e. Simulator exists, that generates transcripts for arbitrary challenges e.

Simulator(c, e):

- Sample* random noise vector $\hat{r}$.
- Compute lattice point $d \in J$ corresponding to $2\hat{r}$. I.e. $\hat{c} = 2\hat{r} + d$.
- $c' \leftarrow \hat{c} - e \cdot c$.
- Output transcript (c', e, d).

✓ Transcript is valid. In particular $e \cdot c + c' - d = 2\hat{r}$ is well-formed noise.

✓ Honest verifier does not learn anything about b.

Special Honest-Verifier Zero-Knowledge

- Honest verifier should not learn anything from an execution of the protocol.
- I.e. Simulator exists, that generates transcripts for arbitrary challenges e.

Simulator(c, e):

- Sample* random noise vector $\hat{r}$.
- Compute lattice point $d \in J$ corresponding to $2\hat{r}$. I.e. $\hat{c} = 2\hat{r} + d$.
- $c' \leftarrow \hat{c} - e \cdot c$.
- Output transcript (c', e, d).

✓ Transcript is valid. In particular $e \cdot c + c' - d = 2\hat{r}$ is well-formed noise.

✓ Honest verifier does not learn anything about b.

Special Honest-Verifier Zero-Knowledge

- Honest verifier should not learn anything from an execution of the protocol.
- I.e. Simulator exists, that generates transcripts for arbitrary challenges e.

Simulator(c, e):

- Sample* random noise vector $\hat{r}$.
- Compute lattice point $d \in J$ corresponding to $2\hat{r}$. I.e. $\hat{c} = 2\hat{r} + d$.
- $c' \leftarrow \hat{c} - e \cdot c$.
- Output transcript (c', e, d).

✓ Transcript is valid. In particular $e \cdot c + c' - d = 2\hat{r}$ is well-formed noise.

✓ Honest verifier does not learn anything about b.

Special Honest-Verifier Zero-Knowledge

- Honest verifier should not learn anything from an execution of the protocol.
- I.e. Simulator exists, that generates transcripts for arbitrary challenges e.

Simulator(c, e):

- Sample* random noise vector $\hat{r}$.
- Compute lattice point $d \in J$ corresponding to $2\hat{r}$. I.e. $\hat{c} = 2\hat{r} + d$.
- $c' \leftarrow \hat{c} - e \cdot c$.
- Output transcript (c', e, d).

✓ Transcript is valid. In particular $e \cdot c + c' - d = 2\hat{r}$ is well-formed noise.

✓ Honest verifier does not learn anything about b.

Remarks on Fully Homomorphic Encryption Schemes

- Single bit plaintexts with current construction.
  - Parameters can be chosen to support larger plaintext spaces.
- Bootstrapping during FHE computation: Encryption uses randomness.
  - Everyone should be able to retrace computation on ciphertexts.
- Integrity during ciphertext switching?
  - Ensure that encrypted secret key during key switching is later used in ZK proof.
  - Addressed in [Car+18]: verify integrity of single message.
  - Doubt that this is enough!

Remarks on Fully Homomorphic Encryption Schemes

- Single bit plaintexts with current construction.
  - Parameters can be chosen to support larger plaintext spaces.
- Bootstrapping during FHE computation: Encryption uses randomness.
  - Everyone should be able to retrace computation on ciphertexts.
- Integrity during ciphertext switching?
  - Ensure that encrypted secret key during key switching is later used in ZK proof.
  - Addressed in [Car+18]: verify integrity of single message.
  - Doubt that this is enough!

Remarks on Fully Homomorphic Encryption Schemes

- Single bit plaintexts with current construction.
  - Parameters can be chosen to support larger plaintext spaces.
- Bootstrapping during FHE computation: Encryption uses randomness.
  - Everyone should be able to retrace computation on ciphertexts.
- Integrity during ciphertext switching?
  - Ensure that encrypted secret key during key switching is later used in ZK proof.
  - Addressed in [Car+18]: verify integrity of single message.
  - Doubt that this is enough!

Remarks on Fully Homomorphic Encryption Schemes

- Single bit plaintexts with current construction.
  - Parameters can be chosen to support larger plaintext spaces.
- Bootstrapping during FHE computation: Encryption uses randomness.
  - Everyone should be able to retrace computation on ciphertexts.
- Integrity during ciphertext switching?
  - Ensure that encrypted secret key during key switching is later used in ZK proof.
  - Addressed in [Car+18]: verify integrity of single message.
  - Doubt that this is enough!

Remarks on Fully Homomorphic Encryption Schemes

- Single bit plaintexts with current construction.
  - Parameters can be chosen to support larger plaintext spaces.
- Bootstrapping during FHE computation: Encryption uses randomness.
  - Everyone should be able to retrace computation on ciphertexts.
- Integrity during ciphertext switching?
  - Ensure that encrypted secret key during key switching is later used in ZK proof.
  - Addressed in [Car+18]: verify integrity of single message.
  - Doubt that this is enough!

Remarks on Zero-Knowledge Proof of Decryption

- Challenge $e \in \{0, 1\}$ too simple.
  - With larger e, $e \cdot c + c'$ might be undecryptable $\rightarrow$ choose parameters wisely.
  - Maybe use $e \in R_2$?
- Do we really need a ZK protocol in the end?
  - Only want to protect secret inputs + secret key.
  - Isn't it enough to simply publish b's from $c = m + 2r + b$?

Remarks on Zero-Knowledge Proof of Decryption

- Challenge $e \in \{0, 1\}$ too simple.
  - With larger e, $e \cdot c + c'$ might be undecryptable
    $\rightarrow$ choose parameters wisely.
  - Maybe use $e \in R_2$?
- Do we really need a ZK protocol in the end?
  - Only want to protect secret inputs + secret key.
  - Isn't it enough to simply publish b's from $c = m + 2r + b$?

Remarks on Zero-Knowledge Proof of Decryption

- Challenge $e \in \{0, 1\}$ too simple.
  - With larger e, $e \cdot c + c'$ might be undecryptable
    $\rightarrow$ choose parameters wisely.
  - Maybe use $e \in R_2$?
- Do we really need a ZK protocol in the end?
  - Only want to protect secret inputs + secret key.
  - Isn't it enough to simply publish b's from $c = m + 2r + b$?

Remarks on Zero-Knowledge Proof of Decryption

- Challenge $e \in \{0, 1\}$ too simple.
  - With larger e, $e \cdot c + c'$ might be undecryptable
    $\rightarrow$ choose parameters wisely.
  - Maybe use $e \in R_2$?
- Do we really need a ZK protocol in the end?
  - Only want to protect secret inputs + secret key.
  - Isn't it enough to simply publish b's from $c = m + 2r + b$?

Remarks on Zero-Knowledge Proof of Decryption

- Challenge $e \in \{0, 1\}$ too simple.
  - With larger e, $e \cdot c + c'$ might be undecryptable
    $\rightarrow$ choose parameters wisely.
  - Maybe use $e \in R_2$?
- Do we really need a ZK protocol in the end?
  - Only want to protect secret inputs + secret key.
  - Isn't it enough to simply publish b's from $c = m + 2r + b$?

Remarks on Zero-Knowledge Proof of Decryption

- Challenge $e \in \{0,1\}$ too simple.
  - With larger e, $e \cdot c + c'$ might be undecryptable
    $\rightarrow$ choose parameters wisely.
  - Maybe use $e \in R_2$?
- Do we really need a ZK protocol in the end?
  - Only want to protect secret inputs + secret key.
  - Isn't it enough to simply publish b's from $c = m + 2r + b$?

Zero-Knowledge Proof of Decryption for FHE Ciphertexts

Thank you for your attention!

Questions?

References I

[BGV14] Zvika Brakerski, Craig Gentry, and Vinod Vaikuntanathan. “(Leveled) fully homomorphic encryption without bootstrapping”. In: *ACM Transactions on Computation Theory (TOCT)* 6.3 (2014), p. 13.

[Car+18] Christopher Carr et al. *Zero-Knowledge Proof of Decryption for FHE Ciphertexts*. Tech. rep. Cryptology ePrint Archive, Report 2018/026, 2018. https://eprint.iacr.org/2018/026, 2018.

[Gen09] Craig Gentry. “Fully homomorphic encryption using ideal lattices”. In: *Proceedings of the Forty-First Annual ACM Symposium on Theory of Computing*. ACM. 2009, pp. 169–178.

[GH11] Craig Gentry and Shai Halevi. “Implementing gentry's fully-homomorphic encryption scheme”. In: *Annual International Conference on the Theory and Applications of Cryptographic Techniques*. Springer. 2011, pp. 129–148.