Fully Homomorphic Encryptions and Blind SNARKs

Cauchy Huang

The National University of Singapore

kexi.huang@u.nus.edu

2025/05

Overview

Different Types of HEs

- **Partially HE (PHE)**. Supports only one type of operations (additions **or** multiplications), with no limitation on the depth of the operations. Eg. RSA, Paillier, ElGamal etc.
- **SomeWhat HE (SWHE)**. Supports additions and multiplications, but supports only **limited depth** of operations.
- **Leveled HE (LHE)**. Supports additions and multiplications with **pre-fixed** depth of operations.
- **Fully HE (FHE)**. Supports additions and multiplications with no limitation on the depth of operations.

Scheme Definition of HEs

Definition (Homomorphic Encryption Schemes.)

- $\mathsf{KeyGen}(1^\lambda) \rightarrow s$
- $\mathsf{Enc}(1^\lambda, s, m) \rightarrow c$
- $\mathsf{Dec}(s, c) \rightarrow m$
- $\mathsf{EvalKeyGen}(s, s') \rightarrow \mathsf{evk}$
- $\mathsf{Eval}(\mathsf{op}, c_1, c_2, \mathsf{evk}) \rightarrow c$

Hard Problems - (Ring) Learning With Errors

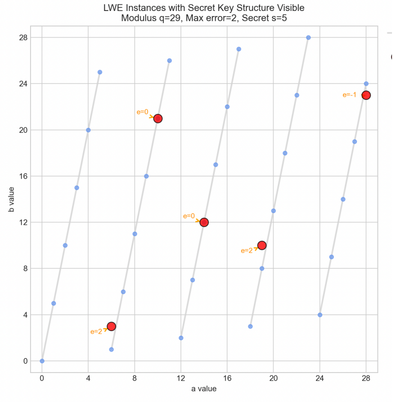
Definition (Ring Learning With Errors)

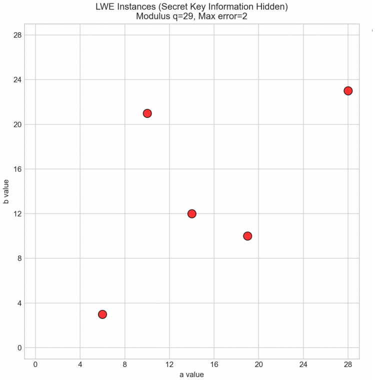
Fix a secret $s \in \mathcal{R}_q$. The RLWE distribution A_s^q is defined by first sampling $a \leftarrow \mathcal{U}_q, e \leftarrow \chi_{err}$ and then returning $(a, [a \cdot s + e]_q)$.

- (Decision version.) Given access to polynomially many samples from $\mathcal{R}_q^2$, distinguish the distributions A_s^q and $\mathcal{U}_q^2$.
- (Search version.) Given access to polynomially many samples from A_s^q, find the underlying s.

Assumption: Both variants of the RLWE problem are conjectured to be hard for appropriately chosen parameters.

Hard Problems - (Ring) Learning With Errors





Construction of BFV

Definition (BFV)

Given plaintext space $\mathcal{R}_t$ and ciphertext space $\mathcal{R}_q$, define the scaling factor $\Delta = q/t$.

- $\mathsf{KeyGen}(1^\lambda) \to s$ Sample $s \leftarrow \mathcal{R}_q$ and return s.
- $\mathsf{Enc}(1^\lambda, s, m) \to (c_0, c_1)$: Sample a RLWE instance (i.e. sample $a \leftarrow \mathcal{R}_q$ and $e \leftarrow \chi_{err}$). Compute $c = [a \cdot s + \lceil \Delta \cdot m \rfloor + e]_q$ and return $(c, -a)$.
- $\mathsf{Dec}(s, (c_0, c_1)) \to m$: Compute $m = \lceil (c_0 + c_1 \cdot s)/\Delta \rfloor$.

Construction of BFV

Definition (BFV (Addition))

- $\text{Eval}(\text{Add}, (c_0, c_1), (c_0', c_1')) \to (c_0'', c_1'')$: Return $(c_0 + c_0', c_1 + c_1')$.

Sketch of Proof:
$c_0 + c_0' = a \cdot s + \lceil \Delta \cdot m \rfloor + e + a' \cdot \lceil \Delta \cdot m' \rfloor + e' =$
$(a + a') \cdot s + \lceil \Delta \cdot (m + m') \rfloor + (e + e' + e_{round})$.
$c_1 + c_1' = -(a + a')$.
Which is the ciphertext of $m + m'$.

Construction of BFV

Definition (BFV (Multiplication))

- $\mathsf{Eval}(\mathsf{Mult}, (c_0, c_1), (c_0', c_1'), \mathsf{evk}) \to (c_0'', c_1'')$:

Underlying induction behind this multiplication:

- Given $c_0 = a \cdot s + \Delta \cdot m + e$ and $c_0' = a' \cdot s + \Delta \cdot m' + e'$, we are now willing to construct something like $aa' \cdot s + \Delta \cdot mm' + e''$.
- With quick observation, one may want to multiply c_0 and c_0' and divide Δ to obtain the term $\Delta \cdot mm'$:

$$c_0'' = c_0 c_0'/\Delta = (aa'/\Delta) \cdot s^2 + (am' + a'm) \cdot s + \Delta \cdot mm' + O(e)$$

- A straight attempt is to compute $c_0 c_1'$ (and for symmetry we also compute $c_0' c_1$), and obtain

$$c_1'' = (c_0 c_1' + c_0' c_1)/\Delta = 2 \cdot (aa' \cdot \Delta) \cdot s + (am' + a'm) + O(e)$$

- Try $c_0'' + c_1'' \cdot s = -(aa'/\Delta) \cdot s^2 + \Delta \cdot mm' + O(e)$

Central Technical Contribution of BFV - Key Switching

Lemma (Gadget Decomposition)

Define the gadget decomposition:

- $\mathcal{D}_{w,q}(a) = (a, \lceil \frac{a}{w} \rfloor, \lceil \frac{a}{w^2} \rfloor, \ldots, \lceil \frac{a}{w^{\log_w(q)-1}} \rfloor)^T$
- $\mathcal{P}_{w,q}(a) = (a, a \cdot w, \ldots, a \cdot w^{\log_w(q)-1})^T$

Then we have $< \mathcal{D}_{w,q}(a), \mathcal{P}_{w,q}(a) >= a \cdot b \mod q\mathcal{R}$

Definition (Key Switching)

- $\mathsf{EvalKeyGen}(s, s') \rightarrow \mathsf{evk}$: Sample $\vec{a} \leftarrow \mathcal{R}_q^{\log_w(q)}$ and $\vec{e} \leftarrow \chi_{err}^{\log_w(q)}$. Return $\mathsf{evk} = (\mathcal{P}_{w,q}(s') + \vec{a} \cdot s + \vec{e}, -\vec{a})$.
- $\mathsf{KeySwitch}(c_0, \mathsf{evk} = (\vec{r_0}, \vec{r_1})) \rightarrow (c_0', c_1')$: Return $(< \vec{c}, \vec{r_0} >, < \vec{c}, \vec{r_1} >)$.

Central Technical Contribution of BFV - Key Switching

Definition (Key Switching)

- $\mathsf{EvalKeyGen}(s, s') \to \mathsf{evk}$: Sample $\vec{a} \leftarrow \mathcal{R}_q^{\log_w(q)}$ and $\vec{e} \leftarrow \chi_{err}^{\log_w(q)}$. Return $\mathsf{evk} = (\mathcal{P}_{w,q}(s') + \vec{a} \cdot s + \vec{e}, -\vec{a})$.
- $\mathsf{KeySwitch}(c_0, \mathsf{evk} = (\vec{r_0}, \vec{r_1})) \to (c_0', c_1')$: Return $(< \mathcal{D}_{w,q}(c), \vec{r_0} >, < \mathcal{D}_{w,q}(c), \vec{r_1} >)$.

Some verification on this key switching process:

$$
\begin{aligned}
&(< \mathcal{D}_{w,q}(c_0), \vec{r_0} >, < \mathcal{D}_{w,q}(c_0), \vec{r_1} >) \\
&= (< \mathcal{D}(c_0), \mathcal{P}(s') > + < \mathcal{D}(c_0), \vec{a} \cdot s + \vec{e} >, < \mathcal{D}(c_0), -\vec{a} >) \\
&= (c_0 \cdot s' + < \mathcal{D}(c_0), \vec{a} \cdot s + \vec{e} >, < \mathcal{D}(c_0), -\vec{a} >)
\end{aligned}
$$

Note that, if we assign $s' = s^2$ and $c_0 = aa'/\Delta$, we can obtain the ciphertext of $aa'/\Delta \cdot s^2$ under the key s.

All Together

Definition (BFV (Multiplication))

- $\mathsf{Eval}(\mathsf{Mult}, (c_0, c_1), (c_0', c_1'), \mathsf{evk}) \rightarrow (c_0'', c_1'')$:
  - Compute $c_0''' = c_0 c_0' / \Delta$ and $c_1''' = (c_0 c_1' + c_0' c_1) / \Delta$
  - Compute $c_{swicth} = \mathsf{KeySwitch}(c_1 c_1' / \Delta, \mathsf{evk})$
  - Return $\mathsf{Eval}(\mathsf{Add}, (c_0''', c_1'''), c_{switch})$

Verifiable Computation On Encrypted Data (vCOED)

The problem setting is when you want to delegate a complicated computation to a powerful third-party, while you still need to keep the input secret.

Current techniques that implement vCOED:

- Trusted Executive Environment (TEE):
  - Pros: Efficient, Low barrier for the users, Low invasion to codes
  - Cons: Rely on the trustness of Processor Manufacturers
- Multi-Party Computation
  - Pros: Support multi-party setting, Faster than FHE
  - Cons: Large computation/communication cost, Every party needs to be online
- Fully Homomorphic Encryption
  - Pros: Non-interactive, support all arithmetic circuits
  - Cons: Huge computation cost

Challenges & Contributions

Challenges when applying FHE to Sumcheck:

- Adaptation of fields: SNARKs typically uses large extension fields (> 120 bits) to ensure soundness, while FHE typically occurs on small fields (< 64 bits).
- Growing depth of multiplications: For natural BFV scheme, multiplications of pt-ct need to encrypt pt and perform mulitplication on ciphertext space.
- Instantiation of Encrypted oracles: In Spartan, the final verification in the sumcheck needs the help of an oracle, which is typically instantiated by Polynomial Commitment Schemes.

Challenges & Contributions

Our Contributions:

- Relying on the security property of FHEs, we prove that it is enough to sample random challenges on base fields (< 64 bits) to achieve the security requirement
- Transform the multiplication of pt-ct into a collection of ct sums, leading to a significant reduce in depth of ct-ct multiplications (With smaller error)
- Instantiation of Encrypted oracles (Ongoing)

Theorem

Given a plaintext $a \in F_p$ and a ciphertext $(c_0, c_1) \in F_q^2$, $\mathrm{Eval}(\mathrm{Mult}, \mathrm{Enc}(a), (c_0, c_1), \mathrm{evk})$ *and* $(a_{\mathbb{Z}} \cdot c_0, a_{\mathbb{Z}} \cdot c_1)$ *share the common underlying plaintext, where $a_{\mathbb{Z}}$ is the element in $\mathbb{Z}$. In the other words, F_q is viewed as the $\mathbb{Z}-$module, and $\cdot$ is the scalar multiplication on the module.*

Blind Sumcheck Protocol

- **Input**:
  - **Public**: A plaintext $y \in \mathbb{F}_p$, claimed to be equal to $\sum_{\vec{x} \in B_n} f(\vec{x})$
  - **Prover $\mathcal{P}$**: The coefficients of an encrypted multilinear polynomial $\{[\![f_\alpha]\!]\}_{\alpha \in B_n}$, where $[\![f_i]\!] \in \mathbb{F}_q^2$ contains two elements from the ciphertext space.
  - **Verifier $\mathcal{V}$**: A multilinear polynomial $f(X_1, \ldots, X_n) = \sum_{\alpha \in B_n} f_\alpha \prod_{i=1}^{n} X_i^{\alpha_i}$.
- **Output**: Verifier outputs accept or reject.

- **Protocol**:
  - In the first round:
    - $\mathcal{P}$ sends the oracle of the encrypted multilinear polynomial $\boxed{[\![f]\!]}$ to $\mathcal{V}$.
    - $\mathcal{P}$ computes $\sum_{b_{[2:]} \in B_{n-1}} [\![f]\!](X, b_{[2:]})$ which is of the form $[\![a]\!] \cdot X + [\![b]\!]$.
    - $\mathcal{P}$ sends the full ciphertexts of $(c_0^a, c_1^a), (c_0^b, c_1^b)$ to $\mathcal{V}$.
    - $\mathcal{V}$ decrypts the plaintexts and obtains the line $L_1(X) = a \cdot X + b$. $\mathcal{V}$ checks if $L_1(0) + L_1(1) = y$.
    - $\mathcal{V}$ samples a random challenge $r_1 \leftarrow$, and sends it to $\mathcal{P}$.
  - In the i-th round, $i \in [2, n]$:
    - $\mathcal{P}$ computes $\sum_{b_{[i+1:]} \in B_{n-i}} [\![f]\!](r_{[:i]}, X, b_{[i+1:]})$ which is of the form $[\![a]\!] \cdot X + [\![b]\!]$.
    - $\mathcal{P}$ sends the full ciphertexts of $(c_0^a, c_1^a), (c_0^b, c_1^b)$ to $\mathcal{V}$.
    - $\mathcal{V}$ decrypts the plaintexts and obtains the line $L_i(X) = a \cdot X + b$. $\mathcal{V}$ checks if $L_i(0) + L_i(1) = L_{i-1}(r_{i-1})$.
    - $\mathcal{V}$ samples a random challenge $r_i \leftarrow \mathbb{F}_p$. $\mathcal{V}$ sends r_i to $\mathcal{P}$ if this is not the last round.
  - Final verification:
    - $\mathcal{V}$ computes $L_i(r_i)$ and queries the point $\vec{r} = (r_1, \ldots, r_n)$ from the oracle $\boxed{[\![f]\!]}$ and decrypts the response to get the evaluation z, which is expected to equal $f(\vec{r})$.

The End