Introduction to FHE
and
the TFHE Scheme

Ilaria Chillotti
KU Leuven & Zama

Lattices: From Theory to Practice
Simons Institute
30 April 2020
Table of Contents

1 Introduction to FHE

2 The TFHE scheme
   - Gate bootstrapping
   - Vertical packing and LUT evaluation
   - TFHE implementation

3 Conclusion
1. Introduction to FHE

2. The TFHE scheme
   - Gate bootstrapping
   - Vertical packing and LUT evaluation
   - TFHE implementation

3. Conclusion
Homomorphic Encryption

Allows to perform computations on encrypted messages, without decrypting.

\[ m_1 \rightarrow m_2 \]

- Possibly any function
- Different message spaces
- Secret and public key solutions

Many applications

- Computations over sensitive data (medical, biological, financial, etc.)
- Outsourced computations
- Electronic voting
- Multiparty Computations
- And more...
Once upon a time...

- **1978** - Rivest, Adleman, Dertouzos: *privacy homomorphisms*
- ...
- **2009** - Gentry: first **fully** homomorphic encryption construction

What happened in the meantime?

Many schemes are homomorphic...

- RSA
- ElGamal
- ...
- Paillier
- Goldwasser-Micali

...but only **partially**.

Some schemes can support both addition and multiplication, but "with limits":

- **somewhat**: example the scheme by Boneh, Goh and Nissim 2005
- **leveled**........
A world full of noise...

Example: [DGHV10]

Scheme based on the Approximate GCD problem [HG01], proposed by Van Dijk, Gentry, Halevi, Vaikuntanathan in 2010.

\[ c = m + 2r + pq \]

- \( m \in \{0, 1\} \) message
- \( p \in \mathbb{Z} \) secret key
- \( q \in \mathbb{Z} \) large \( (p \ll q) \)
- \( r \in \mathbb{Z} \) small noise \( (r \ll p) \)

To decrypt: ciphertext modulo \( p \) and then modulo 2.
A world full of noise...

\[ c_1 = m_1 + 2r_1 + pq_1 \quad \quad c_2 = m_2 + 2r_2 + pq_2 \]

Addition (XOR):
\[ c_1 + c_2 = (m_1 + m_2) + 2(r_1 + r_2) + p(q_1 + q_2) \]

*Noise amount : double...*

Multiplication (AND):
\[ c_1 \cdot c_2 = (m_1 \cdot m_2) + 2(2r_1 \cdot r_2 + \ldots) + p(q_1 \cdot q_2 + \ldots) \]

*Noise amount : square...*

If noise grows too much, a correct decryption cannot be guaranteed!
Bootstrapping [Gen09]

\[ m \xrightarrow{\varphi} \varphi(m) \xrightarrow{\varphi} \varphi(m) \xrightarrow{Dec} \varphi(m) \]

\[ m = Enc_{pk_1} \]

\[ \varphi(m) = Enc_{pk_2} \]
Bootstrapping is very costly!

"To bootstrap, or not to bootstrap, that is the question" (semi cit.)

Leveled homomorphic

Set the function, there exist parameters to homomorphically evaluate it.

- Fast evaluations
- The depth has to be known in advance

Fully homomorphic

Set the parameters, it is possible to homomorphically evaluate any function.

- Slow evaluations (Bootstrapping)
- No depth limitations
Existing schemes

Lattice problems

**Approximate-GCD** [HG01], **NTRU** [HPS98], **(Ring-)LWE** [Reg05],[SSTX09],[LPR10]

- In this workshop we will mainly concentrate on (Ring-)LWE-based solutions

Some (Ring-)LWE-based schemes

"BGV-like"
- B(G)V: [BV11], [BGV12]
- B/FV: [Bra12], [FV12]
- HEAAN: [CKKS17]

"GSW-like"
- GSW: [GSW13]
- FHEW: [DM15]
- TFHE: [CGGI16-17]

In practice, they are less different than expected: Chimera [BGGJ19]

Some implementations

- cuFHE
- FHEW
- HEAAN
- HElib
- Lattigo
- Microsoft SEAL
- NFLlib
- nuFHE
- Palisade
- TFHE
- ...


Table of Contents

1 Introduction to FHE

2 The TFHE scheme
   - Gate bootstrapping
   - Vertical packing and LUT evaluation
   - TFHE implementation

3 Conclusion
FHEW

[DM15]

- GSW-based construction
- They build a FHE brick: a bootstrapped NAND gate
- Slow (but significantly improved): \( \sim 0.69 \) seconds per bootstrapped NAND gate
- Large bootstrapping keys: \( \sim 1 \) GByte

**Bootstrapped versions [CGGI16]**

- Slow (but significantly improved):
  \[\sim 0.69 \sim 0.05 \text{ seconds per bootstrapped NAND gate}\]
- Slow (but significantly improved) [CGGI17]:
  \[\sim 0.69 \sim 0.05 \sim 0.013 \text{ seconds per bootstrapped NAND gate}\]
- Large bootstrapping keys: \[\sim 1 \text{ GByte} \sim 23.4 \text{ MBytes}\]

**Leveled versions [CGGI17]**

- Fast(er) for small depth circuits
- New techniques to improve leveled evaluations
- New Bootstrapping for larger circuits

The real torus $\mathbb{T} = \mathbb{R}/\mathbb{Z} = \mathbb{R} \mod 1$

**Torus**

$(\mathbb{T}, +, \cdot)$ is a $\mathbb{Z}$-module

( the external product $\cdot : \mathbb{Z} \times \mathbb{T} \to \mathbb{T}$ is well defined)

- ✓ It is an abelian group: $x + y \mod 1$, $-x \mod 1$, ...
- ✓ It is a $\mathbb{Z}$-module: $0 \cdot \frac{1}{2} = 0$ is defined!
- ✗ It is **not** a Ring: $0 \times \frac{1}{2}$ is **not** defined!

**Torus polynomials**

$(\mathbb{T}_N[X], +, \cdot)$ is a $\mathcal{R}$-module

- Here, $\mathcal{R} = \mathbb{Z}[X]/(X^N + 1)$
- And $\mathbb{T}_N[X] = \mathbb{T}[X] \mod (X^N + 1)$
TFHE ciphertexts

**LWE**

Message $\mu \in \mathbb{T}$, secret key $s \in \mathbb{B}^n$

\[
\mathbf{c} = (a, b) \in \mathbb{T}^{n+1}
\]

- a random mask, $b = s \cdot a + \varphi$
- $\varphi = e + \mu$, $e \in \mathbb{T}$ Gaussian

$\mathbb{T} = \mathbb{R} \mod 1$, $\mathbb{B} = \{0, 1\}$

**RLWE**

Message $\mu \in \mathbb{T}_N[X]$, secret key $s \in \mathbb{B}_N[X]$

\[
\mathbf{c} = (a, b) \in \mathbb{T}_N[X]^2
\]

- a random mask, $b = s \cdot a + e + \mu$, $e \in \mathbb{T}_N[X]$ Gaussian

\[
\mathbb{T}_N[X] = \mathbb{R}[X]/(X^N + 1) \mod 1, \mathbb{B}_N[X] = \mathbb{Z}[X]/(X^N + 1)\text{ with binary coefs}
\]
TFHE operations

**RGSW**

Message $m \in \mathbb{Z}_N[X]$, secret key $s \in \mathbb{B}_N[X]$ as in RLWE

$C = Z + m \cdot G_2 \in \mathbb{T}_N[X]^{2\ell \times 2}$

- with $Z$ is a list of $2\ell$ RLWE encryptions of 0
- with $G_2$ the **gadget** matrix

$$G_2 = \begin{pmatrix} g & 0 \\ 0 & g \end{pmatrix}, \text{ with } g^T = (2^{-1}, \ldots, 2^{-\ell})$$

$G_2^{-1}$: easy to decompose $\mathbb{T}_N[X]$ elements w.r.t. $G_2$

$$\mathbb{Z}_N[X] = \mathbb{Z}[X]/(X^N + 1)$$
### TFHE Ciphertexts

<table>
<thead>
<tr>
<th></th>
<th>Plaintext</th>
<th>Ciphertext</th>
<th>Linear Combinations</th>
<th>Product</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>LWE</strong></td>
<td>$T$</td>
<td>$T^{n+1}$</td>
<td>✓</td>
<td>X</td>
</tr>
<tr>
<td><strong>RLWE</strong></td>
<td>$T_N[X]$</td>
<td>$T_N[X]^2$</td>
<td>✓</td>
<td>X</td>
</tr>
<tr>
<td><strong>RGSW</strong></td>
<td>$\mathbb{Z}_N[X]$</td>
<td>$T_N[X]^{2\ell \times 2}$</td>
<td>✓</td>
<td>✓</td>
</tr>
</tbody>
</table>

**Diagram:**

- **Plaintext**
  - **LWE**: Empty
  - **RLWE**: Empty
  - **RGSW**: Empty

- **Ciphertext**
  - **LWE**: Empty
  - **RLWE**: Empty
  - **RGSW**: Empty
TFHE products

Internal RGSW product

\[
C \boxtimes D = G_2^{-1}(D) \cdot C = \begin{bmatrix}
G_2^{-1}(d_1) \cdot C \\
\vdots \\
G_2^{-1}(d_{2\ell}) \cdot C
\end{bmatrix} = \begin{bmatrix}
C \boxdot d_1 \\
\vdots \\
C \boxdot d_{2\ell}
\end{bmatrix}
\]

External RGSW – RLWE product \cite{CGGI16,BP16}

\[
C \boxdot d = G_2^{-1}(d) \cdot C
\]
MUX($C, d_1, d_0$) = $C \boxplus (d_1 - d_0) + d_0$

Largely used in TFHE leveled and bootstrapped constructions.
How often shall we bootstrap?

**Gate bootstrapping:** bootstrap after every gate (like [DM15])

**Circuit bootstrapping:** bootstrap after a larger circuit
Input LWE ciphertext

\[ c = (a, b) \]

Depending on

\[ \varphi = b - a \cdot s \]

we compute an output LWE ciphertext encrypting \( v_\varphi \in \mathbb{T} \)

---

1. Start from (a trivial) RLWE ciphertext of message\(^a\)

\[ ACC = v_0 + v_1 X + \cdots + v_{N-1} X^{N-1} \]

2. Do a blind rotation of ACC by \(-\varphi\) positions (i.e. \( ACC \cdot X^{-\varphi}\))

3. **Extract** the constant term of ACC (which encrypts \( v_\varphi \))

\(^a\)N coefficients modulo \( X^N + 1 \) can be viewed as 2N coefficients modulo \( X^{2N} - 1 \) s.t. \( v_{N+i} = -v_i \)
Gate Bootstrapping

**Gate Bootstrapping**

**Linear Combination**

**Blind Rotation**

**Accumulator**

**Bootstrapping Key**

**MUX**

**i=1**

**Extract**
Leveled constructions

Look-Up Table evaluation

The RLWE slots can be used in an optimal way

- **LWE**: messages \( m \in \mathbb{T} \)
- **RLWE**: messages \( \mathbf{m} \in \mathbb{T}_N[X] \)

\[
\mathbf{m} = \sum_{i=0}^{N-1} m_i \cdot X^i \quad \sim \quad \mathbf{m} = (m_0, m_1, \ldots, m_{N-1})
\]

<table>
<thead>
<tr>
<th>( m_0 )</th>
<th>( m_1 )</th>
<th>( m_2 )</th>
<th>( \ldots )</th>
<th>( m_{N-2} )</th>
<th>( m_{N-1} )</th>
</tr>
</thead>
</table>

Generally \( N = 2^{10} \)
LookUp Tables (LUT)

\[ f : \mathbb{B}^d \rightarrow \mathbb{T}^s \]

\[ x = (x_0, \ldots, x_{d-1}) \mapsto f(x) = (f_0(x), \ldots, f_{s-1}(x)) \]

Example with \( d = 3 \) and \( s = 2 \)

<table>
<thead>
<tr>
<th>( x_0 )</th>
<th>( x_1 )</th>
<th>( x_2 )</th>
<th>( f_0 )</th>
<th>( f_1 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0.5</td>
<td>0.3</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0.25</td>
<td>0.7</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0.1</td>
<td>0.61</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0.83</td>
<td>0.9</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0.23</td>
<td>0.47</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0.67</td>
<td>0.42</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0.78</td>
<td>0.12</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0.35</td>
<td>0.95</td>
</tr>
</tbody>
</table>

LUT largely used in cryptology (ex. evaluation of arbitrary functions, SBoxes, ...)
How to evaluate it?

<table>
<thead>
<tr>
<th>$x_0 \cdots x_{d-1}$</th>
<th>$f_0 \cdots f_{s-1}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 \cdots 0</td>
<td>$\sigma_{0,0} \cdots \sigma_{s-1,0}$</td>
</tr>
<tr>
<td>1 \cdots 0</td>
<td>$\sigma_{0,1} \cdots \sigma_{s-1,1}$</td>
</tr>
<tr>
<td>0 \cdots 0</td>
<td>$\sigma_{0,2} \cdots \sigma_{s-1,2}$</td>
</tr>
<tr>
<td>1 \cdots 0</td>
<td>$\sigma_{0,3} \cdots \sigma_{s-1,3}$</td>
</tr>
<tr>
<td>\vdots</td>
<td>\vdots \vdots \vdots \vdots</td>
</tr>
<tr>
<td>0 \cdots 1</td>
<td>$\sigma_{0,2^{d-4}} \cdots \sigma_{s-1,2^{d-4}}$</td>
</tr>
<tr>
<td>1 \cdots 1</td>
<td>$\sigma_{0,2^{d-3}} \cdots \sigma_{s-1,2^{d-3}}$</td>
</tr>
<tr>
<td>0 \cdots 1</td>
<td>$\sigma_{0,2^{d-2}} \cdots \sigma_{s-1,2^{d-2}}$</td>
</tr>
<tr>
<td>1 \cdots 1</td>
<td>$\sigma_{0,2^{d-1}} \cdots \sigma_{s-1,2^{d-1}}$</td>
</tr>
</tbody>
</table>

$\sigma_{j,i} \rightarrow \begin{cases} 0 & \text{if } x_i = 0 \\ 1 & \text{if } x_i = 1 \end{cases}$
Batching (Horizontal Packing)

- Pack the outputs in a RLWE ciphertext (green box)

<table>
<thead>
<tr>
<th>$x_0 \cdots x_{d-1}$</th>
<th>$f_0 \cdots f_{s-1}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 ... 0</td>
<td>$\sigma_{0,0} \cdots \sigma_{s-1,0}$</td>
</tr>
<tr>
<td>1 ... 0</td>
<td>$\sigma_{0,1} \cdots \sigma_{s-1,1}$</td>
</tr>
<tr>
<td>0 ... 0</td>
<td>$\sigma_{0,2} \cdots \sigma_{s-1,2}$</td>
</tr>
<tr>
<td>1 ... 0</td>
<td>$\sigma_{0,3} \cdots \sigma_{s-1,3}$</td>
</tr>
<tr>
<td>... ... ...</td>
<td>... ... ... ...</td>
</tr>
<tr>
<td>0 ... 1</td>
<td>$\sigma_{0,2^d-4} \cdots \sigma_{s-1,2^d-4}$</td>
</tr>
<tr>
<td>1 ... 1</td>
<td>$\sigma_{0,2^d-3} \cdots \sigma_{s-1,2^d-3}$</td>
</tr>
<tr>
<td>0 ... 1</td>
<td>$\sigma_{0,2^d-2} \cdots \sigma_{s-1,2^d-2}$</td>
</tr>
<tr>
<td>1 ... 1</td>
<td>$\sigma_{0,2^d-1} \cdots \sigma_{s-1,2^d-1}$</td>
</tr>
</tbody>
</table>
## Vertical packing

<table>
<thead>
<tr>
<th></th>
<th>$x_0$</th>
<th>$x_1$</th>
<th>$x_2$</th>
<th>$x_{d-1}$</th>
<th>$f_0$</th>
<th>$f_1$</th>
<th>$f_2$</th>
<th>$f_{s-1}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>···</td>
<td>0</td>
<td>···</td>
<td>···</td>
<td>···</td>
<td>···</td>
<td>···</td>
<td>···</td>
</tr>
<tr>
<td>1</td>
<td>···</td>
<td>0</td>
<td>···</td>
<td>···</td>
<td>···</td>
<td>···</td>
<td>···</td>
<td>···</td>
</tr>
<tr>
<td>0</td>
<td>···</td>
<td>0</td>
<td>···</td>
<td>···</td>
<td>···</td>
<td>···</td>
<td>···</td>
<td>···</td>
</tr>
<tr>
<td>1</td>
<td>···</td>
<td>0</td>
<td>···</td>
<td>···</td>
<td>···</td>
<td>···</td>
<td>···</td>
<td>···</td>
</tr>
<tr>
<td>···</td>
<td>···</td>
<td>···</td>
<td>···</td>
<td>···</td>
<td>···</td>
<td>···</td>
<td>···</td>
<td>···</td>
</tr>
<tr>
<td>0</td>
<td>···</td>
<td>1</td>
<td>···</td>
<td>···</td>
<td>···</td>
<td>···</td>
<td>···</td>
<td>···</td>
</tr>
<tr>
<td>1</td>
<td>···</td>
<td>1</td>
<td>···</td>
<td>···</td>
<td>···</td>
<td>···</td>
<td>···</td>
<td>···</td>
</tr>
<tr>
<td>0</td>
<td>···</td>
<td>1</td>
<td>···</td>
<td>···</td>
<td>···</td>
<td>···</td>
<td>···</td>
<td>···</td>
</tr>
<tr>
<td>1</td>
<td>···</td>
<td>1</td>
<td>···</td>
<td>···</td>
<td>···</td>
<td>···</td>
<td>···</td>
<td>···</td>
</tr>
</tbody>
</table>
LUT evaluation: Batching and Vertical Packing

Vertical Packing

<table>
<thead>
<tr>
<th>$x_0$</th>
<th>$\cdots$</th>
<th>$x_{d-1}$</th>
<th>$f_j$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>$\cdots$</td>
<td>0</td>
<td>$\sigma_j,0$</td>
</tr>
<tr>
<td>1</td>
<td>$\cdots$</td>
<td>0</td>
<td>$\sigma_j,1$</td>
</tr>
<tr>
<td>0</td>
<td>$\cdots$</td>
<td>0</td>
<td>$\sigma_j,2$</td>
</tr>
<tr>
<td>1</td>
<td>$\cdots$</td>
<td>0</td>
<td>$\sigma_j,3$</td>
</tr>
<tr>
<td>\vdots</td>
<td>$\cdots$</td>
<td>\vdots</td>
<td>\vdots</td>
</tr>
<tr>
<td>0</td>
<td>$\cdots$</td>
<td>1</td>
<td>$\sigma_j,2^{d-4}$</td>
</tr>
<tr>
<td>1</td>
<td>$\cdots$</td>
<td>1</td>
<td>$\sigma_j,2^{d-3}$</td>
</tr>
<tr>
<td>0</td>
<td>$\cdots$</td>
<td>1</td>
<td>$\sigma_j,2^{d-2}$</td>
</tr>
<tr>
<td>1</td>
<td>$\cdots$</td>
<td>1</td>
<td>$\sigma_j,2^{d-1}$</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>$x_{d-2}$</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>$x_{d-1}$</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>$(x_0, \ldots, x_{d-3})$</td>
</tr>
</tbody>
</table>
LUT evaluation: Batching and Vertical Packing

Mix them all...

- Depending on the use case, choose which type of packing is the best
- You can mix them: they are compatible

<table>
<thead>
<tr>
<th>$x_0$</th>
<th>$\cdots$</th>
<th>$x_{d-1}$</th>
<th>$f_0$</th>
<th>$\cdots$</th>
<th>$f_{s-1}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>$\cdots$</td>
<td>0</td>
<td>$\sigma_{0,0}$</td>
<td>$\cdots$</td>
<td>$\sigma_{s-1,0}$</td>
</tr>
<tr>
<td>1</td>
<td>$\cdots$</td>
<td>0</td>
<td>$\sigma_{0,1}$</td>
<td>$\cdots$</td>
<td>$\sigma_{s-1,1}$</td>
</tr>
<tr>
<td>0</td>
<td>$\cdots$</td>
<td>0</td>
<td>$\sigma_{0,2}$</td>
<td>$\cdots$</td>
<td>$\sigma_{s-1,2}$</td>
</tr>
<tr>
<td>1</td>
<td>$\cdots$</td>
<td>0</td>
<td>$\sigma_{0,3}$</td>
<td>$\cdots$</td>
<td>$\sigma_{s-1,3}$</td>
</tr>
<tr>
<td>$\vdots$</td>
<td>$\cdots$</td>
<td>$\vdots$</td>
<td>$\vdots$</td>
<td>$\cdots$</td>
<td>$\vdots$</td>
</tr>
<tr>
<td>0</td>
<td>$\cdots$</td>
<td>1</td>
<td>$\sigma_{0,2d-4}$</td>
<td>$\cdots$</td>
<td>$\sigma_{s-1,2d-4}$</td>
</tr>
<tr>
<td>1</td>
<td>$\cdots$</td>
<td>1</td>
<td>$\sigma_{0,2d-3}$</td>
<td>$\cdots$</td>
<td>$\sigma_{s-1,2d-3}$</td>
</tr>
<tr>
<td>0</td>
<td>$\cdots$</td>
<td>1</td>
<td>$\sigma_{0,2d-2}$</td>
<td>$\cdots$</td>
<td>$\sigma_{s-1,2d-2}$</td>
</tr>
<tr>
<td>1</td>
<td>$\cdots$</td>
<td>1</td>
<td>$\sigma_{0,2d-1}$</td>
<td>$\cdots$</td>
<td>$\sigma_{s-1,2d-1}$</td>
</tr>
</tbody>
</table>
More TFHE

Seen in this presentation
- Basic construction
- Gate bootstrapping
- Evaluation of LUT (leveled)

More...
- Evaluate deterministic (weighted) finite automata
- The homomorphic counter TBSR
- Circuit bootstrapping
- ...
TFHE: Fast Fully Homomorphic Encryption over the Torus

- Open source C/C++ library
  https://tfhe.github.io/tfhe/
- Distributed under Apache 2.0 license

**Gate bootstrapping**
- All gates implemented in the official release

**Circuit bootstrapping and leveled operations**
- Implemented in the experimental repository
  https://github.com/tfhe/experimental-tfhe
TFHE in Gate Bootstrap mode versus Circuit Bootstrap mode

**TFHE Gate Bootstrap mode**
- Input/Output: LWE → LWE
- Gate bootstrapping in 10-20 ms
- All binary gates have the same cost

Evaluate about 70 bootstrapped binary gates per second.

**TFHE Circuit Bootstrap mode**
- Input/Output: LWE → RGSW
- Circuit bootstrapping in 137 ms
- After many transitions 34 μs

Evaluate a LUT from 16-bit input to 8-bit output in 1 second.

**Bit Overhead**
- LWE: 2.46 KB (encrypts 1 message)
- RLWE: 8 KB (encrypts up to 1024 messages)
- RGSW: 48 KB (encrypts up to 1024 messages)

- Implementation tested on (single core) Intel i7 and Intel i9 processor laptops
- Parameters have 128-bits of security according to the LWE estimator
# Table of Contents

1. Introduction to FHE
2. The TFHE scheme
   - Gate bootstrapping
   - Vertical packing and LUT evaluation
   - TFHE implementation
3. Conclusion
Conclusion

Some TFHE related works

- GPU implementations: cuFHE, nuFHE
- Neural network applications: [BMMP18], TFHE-Chimera solution at iDASH 2019
- Multi-key: MK-TFHE [CCS19]
- Use in MPC: Onion Ring ORAM [CCR19]

Thank you!

Questions?


