Carousel:

Fully Homomorphic Encryption with Bootstrapping over Automorphism Group

Intak Hwang, **Seonhong Min**, Yongsoo Song

Seoul National University, Seoul

FHE and Bootstrapping

- Lattice-based ciphertext contains noise, which increases when performing homomorphic operations (especially multiplication).
- Lattice-based FHE schemes require **bootstrapping** to evaluate circuits with arbitrary depth.
  - Roughly speaking, bootstrapping is a **homomorphic evaluation of the decryption circuit**.
  - Bootstrapping is the **main bottleneck** of homomorphic computation.

FHE and Bootstrapping

- **SIMD**
  - BGV, BFV, CKKS
  - Represents the decryption circuit with linear transformations and p-adic polynomials.

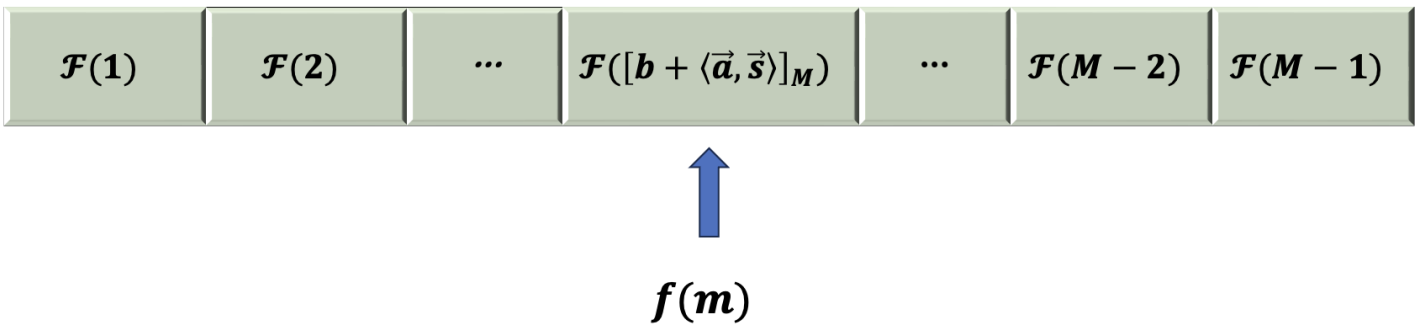
  \+ **High throughput**
  − **Large latency**

- **AP-like**
  - FHEW, TFHE, LMKCDEY
  - Represents the decryption circuit with a lookup table.

  \+ **Low latency, Programmable bootstrapping**
  − **Bad throughput, (and more...)**

AP-like bootstrapping

- **Regev encryption** (=BFV)
  - p, M denote the plaintext and ciphertext moduli, respectively.
  - Ciphertext: $(b, \vec{a}) \in \mathbb{Z}_M^{n+1}$
  - Encoding: $b + \langle \vec{a}, \vec{s} \rangle = \left\lfloor \frac{M}{p} \cdot m \right\rceil + e \pmod{M}$ for $\|e\|_\infty < \frac{M}{2p}$
  - Decryption: $m = \left\lfloor \frac{p}{M} (b + \langle \vec{a}, \vec{s} \rangle \bmod M) \right\rceil$
- (Functional) **Decryption Formula**
  - Given a function $f : \mathbb{Z}_p \to \mathbb{Z}_p$
  - Generate an LUT $\mathcal{F} : \mathbb{Z}_M \to \mathbb{Z}_p$ such that $\mathcal{F}(i) = f\left(\left\lfloor \frac{p}{M} i \right\rceil\right)$



AP-like bootstrapping

- **How can one decrypt homomorphically?**
  - The main difficulty is performing the **modulo reduction** with M.
    $\rightarrow$ Find a cyclic group of size M, and embed $\mathbb{Z}_M$ into it.
  - Simulate $\mathbb{Z}_M$ arithmetic using the **roots of unity** over the ring of integers $\mathbb{Z}[\zeta_M]$.
  - i.e., $\{1, \zeta_M, \ldots, \zeta_M^{M-1}\}$ forms a multiplicative cyclic group of size M.
- **Blind Rotation**
  - Recall the LUT $\mathcal{F}(i) = f\left(\left\lfloor \frac{p}{M} i \right\rceil\right)$.
  - Find $tv \in \mathbb{Z}_p[\zeta_M]$ such that the **constant term** of $tv \cdot \zeta_M^i$ is $\mathcal{F}(i)$.
  - Compute $tv \cdot \zeta_M^{b+\langle \vec{a}, \vec{s} \rangle}$ and **extract the constant term**.
  - Then the result is $\mathcal{F}(b + \langle \vec{a}, \vec{s} \rangle \bmod M) = f(m)$.

Limitation

- *tv* can have only $\phi(M) < M$ coefficients.
  - $\rightarrow$ Only a $\phi(M)/M$ fraction of LUT can be encoded!
- For example...
  - If M is a **power-of-two**...
  - Only **negacyclic** functions can be evaluated.
  - Or, we can only use half of the plaintext space.
  - Some works solve the issue with **more than two bootstrappings**.
- However, there is **no solution that natively solves this issue**.

$\rightarrow$ *Can a blind rotation algorithm naturally cover the entire input plaintext space?*

New Blind Rotation

- Is there any **other cyclic group** that we can use instead of the monomial group?
  - Yes, the **automorphism group**!
  - Find a ring where the automorphism group is isomorphic to $\mathbb{Z}_M$.
    - ★ i.e. $\{id, \Psi, \ldots, \Psi^{M-1}\}$.
    - ★ e.g., an odd prime power cyclotomic ring.
  - It has the same size as the coefficients.

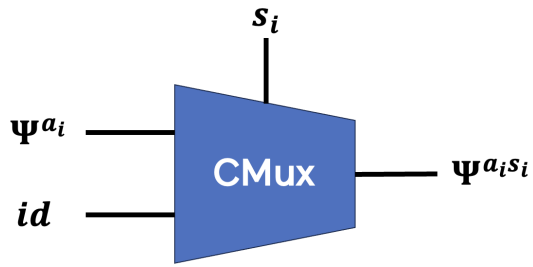
    $\rightarrow$ The whole LUT can be encoded!

- We homomorphically evaluate $\Psi^{b+\langle \vec{a}, \vec{s} \rangle} = \Psi^{b+\langle \vec{a}, \vec{s} \rangle \bmod M}$.
  - Given the LUT $\mathcal{F} \in \mathbb{Z}_p^M$,
  - If there exist Ecd and Dcd such that $\mathrm{Dcd}(\Psi^i(\mathrm{Ecd}(\mathcal{F}))) = \mathcal{F}(i)$ holds for all $0 \leq i < M$,
  - We set the **test vector** $tv = \mathrm{Ecd}(\mathcal{F})$.
  - Then $\mathrm{Dcd}(\Psi^{b+\langle \vec{a}, \vec{s} \rangle}(tv)) = \mathcal{F}(b + \langle \vec{a}, \vec{s} \rangle \bmod M)$.

New Blind Rotation

- We want to compute $\Psi^{b+\langle \vec{a}, \vec{s} \rangle}(tv)$, where $(b, \vec{a})$ is **public** and $\vec{s}$ is **private**.
- To compute $\Psi^{b+\langle \vec{a}, \vec{s} \rangle}(tv)$, we evaluate $\Psi^{a_i s_i}$ iteratively on $\Psi^{b}(tv)$.
- Use **CMux-gate** from [GINX16]!
  - Used in TFHE [CGGI16].
- We just need to assume that $\vec{s} = (s_0, s_1, \ldots, s_{n-1})$ is binary.

Cmux-gate



- CMux-gate
  - If s_i is binary, evaluating $\Psi^{a_i s_i}$ can be seen as choosing between Ψ^{a_i} and id w.r.t. s_i.
  - $\Psi^{a_i s_i} = (1 - s_i)id + s_i \Psi^{a_i} = id + s_i(\Psi^{a_i} - id)$.
  - This can be implemented with
    - ★ one automorphism
    - ★ one external product (RLWE-RGSW multiplication)
- n automorphisms and n external products in total.

Bootstrapping Algorithm

- Sample Extraction
  - Homomorphically **decode** the input ciphertext.
- Key-switching
  - Switch to a **smaller ciphertext dimension** homomorphically.
  - Make the decryption circuit as small as possible.
- Blind Rotation
  - Homomorphically evaluate the **decryption circuit**.
  - Run the blind rotation algorithm.

Carousel

- Instantiation of our blind rotation over a **subring** of $\mathbb{Z}[\zeta_N]$ with a cyclic automorphism group.
  - Let N be an odd **prime**.
  - Then the automorphism group is generated by some automorphism Ψ.
    $\rightarrow$ Automorphism group is $\{id, \Psi, \ldots, \Psi^{N-2}\}$.
  - Base ring R is the smallest subring of $\mathbb{Z}[\zeta_N]$ **invariant** under Ψ^M.
    $\rightarrow$ Automorphism group is $\{id, \Psi, \ldots, \Psi^{M-1}\}$.
- Choosing the subring has **advantages**.
  - Optimal **space and time** complexity
  - Subring elements can be written with M **coefficients**, instead of N.
  - There exists an **efficient FFT** algorithm with time complexity $O(M \log M)$ instead of $O(N \log N)$.

Carousel

- Carousel supports two types of **encoding/decoding** strategies.
- **Coefficient packing**
  - We call this Carousel-C
  - Corresponds to the **monomial basis**
  - \+ Lower noise growth
  - − Incompatible with multiplications
- **Slot packing**
  - We call this Carousel-S
  - Corresponds to the **FFT basis**
  - \+ Compatible with multiplications
  - − Higher noise growth

Implementation

- Implemented in Golang
- Existing **optimisation techniques** for TFHE were applied.
  - e.g. **block binary key** [LMSS23].
- https://github.com/SNUCP/carousel

Table 4. The key sizes and latency for a single functional bootstrapping using Carousel.

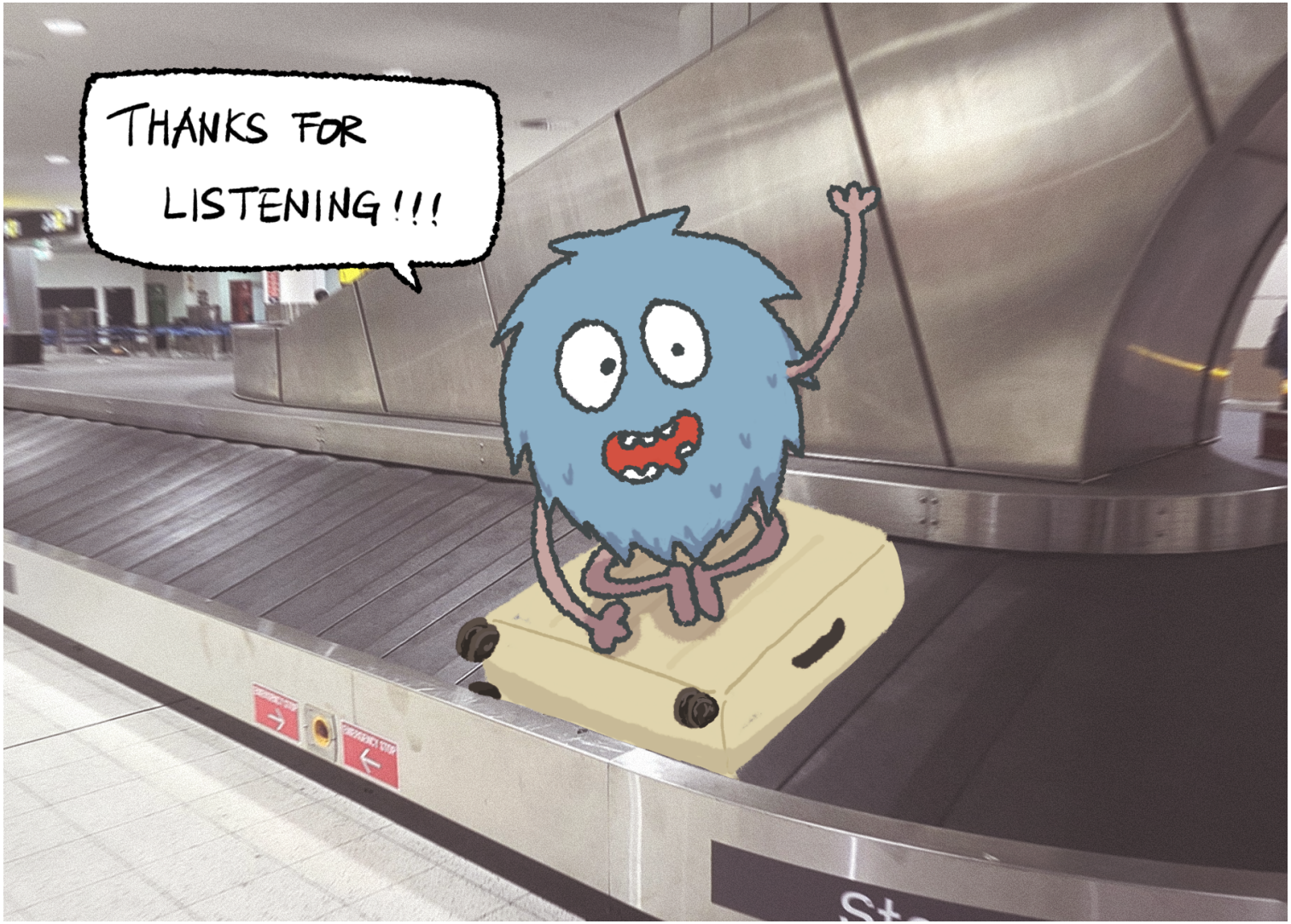
Mode	p	brk	atk	ksk	Elapsed Time
Carousel-C	2^2	43.17MB	64.31MB	44.00MB	19.0ms
	2^3	49.51MB	64.31MB	23.36MB	18.9ms
	2^4	52.03MB	64.31MB	39.46MB	19.6ms
Carousel-S	2^2	86.26MB	128.52MB	44.00MB	28.4ms
	2^3	98.94MB	128.52MB	23.36MB	28.8ms
	2^4	103.97MB	128.52MB	39.46MB	29.6ms

Comparison

Table 5. The latency and total key sizes for a single functional bootstrapping of FHEW [DM15], LMKCDEY [LMK+23], TFHE [Zam22] and Carousel-C.

Scheme	$p = 2^2$	$p = 2^3$	$p = 2^4$
FHEW	(114ms, 556.3MB)	-	-
LMKCDEY	(124ms, 55.4MB)	-	-
TFHE	(11.4ms, 118.4MB)	(12.5ms, 105.0MB)	(15.4ms, 122.4MB)
Carousel-C	(19.6ms, 151.5MB)	(19.4ms, 137.2MB)	(19.6ms, 155.8MB)

- Carousel-C is 1.27–1.72x slower than **negacyclic** functional bootstrapping for TFHE.
- For **full-domain bootstrapping**, Carousel is more than 1.16–1.57x faster than TFHE.



Any questions?