

The TFHE Scheme

Lecture 15

SEC 595 — Encrypted Computing

- 1 The Idea
- 2 Ciphertexts
- 3 Bootstrapping
- 4 Programmable Bootstrapping
- 5 Comparison

Bootstrap after every gate

GSW bootstrapping (Lecture 14) is correct but rare and expensive. BFV, BGV and CKKS spend a noise budget between bootstraps.

TFHE (Chillotti, Gama, Georgieva, Izabachène, 2016)

Keep ciphertexts small, make bootstrapping fast, and bootstrap after **every** gate.

No levels, no depth limit, nothing to plan.

Notation: n is the LWE dimension, N the ring degree.

The message layout

The torus $\mathbb{T} = \mathbb{R}/\mathbb{Z}$ is stored as $\mathbb{Z}_q$, $q = 2^{32}$ or 2^{64} .

As in BFV (Lecture 10): $\Delta = q/p$, and a message $m \in \{0, \dots, p/2 - 1\}$ sits at Δm .

The top half is kept empty: the **padding bit**.

Decryption is correct while the noise is below $\Delta/2$.

Example: $q = 1024$, $p = 8$, $\Delta = 128$. Messages $0, \dots, 3$ at $0, 128, 256, 384$; noise in $(-64, 64)$.

LWE ciphertext

Binary key $s \in \{0, 1\}^n$. An encryption of m is (a, b) with

$$b = \langle a, s \rangle + \Delta m + e.$$

Phase $\varphi = b - \langle a, s \rangle = \Delta m + e$; decrypt $\lfloor \varphi / \Delta \rfloor$.

The phase is linear: adding ciphertexts adds messages and noises, for free.

Inside bootstrapping, two more types:

- **GLWE**: ring-LWE over $\mathbb{Z}_q[X]/(X^N + 1)$, carries a polynomial — the accumulator;
- **GGSW**: GSW with ring entries — the bootstrapping key.

External product and CMux

$$C \boxdot c = G^{-1}(c) \cdot C, \quad \text{CMux}(C, d_1, d_0) = C \boxdot (d_1 - d_0) + d_0.$$

If C is a GGSW encryption of a bit x , then CMux returns an encryption of the message of d_x .

The external product of Lecture 13 over the ring: the noise grows *additively*, with the accumulator in the decomposed slot.

Nobody learns which branch was taken.

$$\text{LWE}_s \xrightarrow{\text{mod switch}} \mathbb{Z}_{2N} \xrightarrow{\text{blind rotation}} \text{GLWE}_S \xrightarrow{\text{extract}} \text{LWE}_S \xrightarrow{\text{key switch}} \text{LWE}_s$$

- **Mod switch:** $\tilde{a}_i = \lfloor 2Na_i/q \rfloor$, $\tilde{b} = \lfloor 2Nb/q \rfloor$, so $\tilde{\varphi} = \tilde{b} - \sum \tilde{a}_i s_i \approx 2N\varphi/q$.
- Each message owns a **box** of $w = 2N/p$ values of $\tilde{\varphi}$.
- **Extract:** the constant coefficient of a GLWE ciphertext is an LWE ciphertext. Free.
- **Key switch:** back to s , with the gadget trick of relinearization.

Since $X^N = -1$, the constant coefficient of $X^{-k}v$ is

$$v_k \quad (k < N), \quad -v_{k-N} \quad (N \leq k < 2N).$$

So store a **table** in the coefficients of a test polynomial:

$$v_j = \Delta f(\lfloor j/w \rfloor), \quad 0 \leq j < N.$$

Rotating v by $X^{-\tilde{\varphi}}$ brings $\Delta f(m)$ to the front.

(Add $w/2$ to $\tilde{b}$ so that each box is centred on its message.)

BK_i = GGSW encryption of s_i . Start from $ACC = (0, X^{-\tilde{b}_v})$; for $i = 1, \dots, n$:

$$ACC \leftarrow \text{CMux}(BK_i, X^{\tilde{a}_i} ACC, ACC).$$

Step i multiplies by $X^{\tilde{a}_i s_i}$, so at the end ACC encrypts $X^{-\tilde{\varphi}_v}$: its constant coefficient is $\Delta f(m)$.

Blind: the evaluator rotates by $\tilde{\varphi}$ without learning it.

The output noise is fixed. The input noise only decides which box $\tilde{\varphi}$ lands in.

Example — one bootstrapping

$q = 1024$, $p = 8$, $\Delta = 128$, $s = (1, 0, 1)$, $N = 16$, $w = 4$, $f(m) = [m \geq 2]$.

- a. Encrypt $m = 2$ with $a = (333, 790, 150)$, $e = 5$: find b .
- b. Mod switch to $2N = 32$: find $\tilde{a}$, $\tilde{b}$, $\tilde{\varphi}$ and the box (remember $+w/2$).
- c. Track the exponent of $X^{(\cdot)}_v$ through the three CMux steps.
- d. Read the constant coefficient. What is the output?

Any function, one bootstrapping

The test polynomial is public and arbitrary.

One bootstrapping refreshes the noise *and* applies any table f to a few-bit message.

Comparisons, thresholds, ReLU: costly polynomial approximations in BFV, BGV and CKKS; one lookup here.

Padding bit: a phase in the upper half returns $-v_{\tilde{\varphi}-N}$, so messages stay below $p/2$.

Multiplication: $2^k c_x + c_y$ encrypts $2^k x + y$; the table $2^k x + y \mapsto xy$ returns the product.

Gates: a sum and a table

For bits, $c_1 + c_2$ encrypts $m_1 + m_2 \in \{0, 1, 2\}$.

$m_1 + m_2$	AND	OR	XOR	NAND
0	0	0	0	1
1	0	1	1	1
2	1	1	0	0

One bootstrapping per gate, each gate leaving a fresh ciphertext. NAND is universal: any circuit runs gate by gate.

Example — a NAND gate

Same parameters ($q = 1024$, $p = 8$, $N = 16$).

- a. Write the test polynomial v for NAND on $m_1 + m_2$.
- b. For $m_1 = m_2 = 1$, which box does the sum land in, and what comes out?
- c. Which box would $m_1 + m_2 = 3$ need, and why is it never used?

How TFHE differs

	BFV/BGV/CKKS	GSW	TFHE
Plaintext	many slots	a bit	a bit or few-bit integer
Ciphertext	ring pair	matrix	LWE vector
Multiply	tensor + relin	$G^{-1}(C_1)C_2$	table lookup
Noise	levels	asymmetric	reset every gate
Bootstrap	rare, costly	costly	every gate, fast
Best for	arithmetic on many values	theory	comparisons, branching

Original parameters: $n = 500$, $N = 1024$, $q = 2^{32}$. Key about 16 MB; one gate about 13 ms (FHEW: 1 GB, 690 ms).

- 1 Messages sit at Δm in $\mathbb{Z}_q$ with a padding bit. LWE carries data, GLWE the accumulator, GGSW the key.
- 2 CMux is an encrypted if-then-else with additive noise.
- 3 Bootstrapping: mod switch, blind rotation of a test polynomial, extraction, key switching. The output noise is fixed.
- 4 The test polynomial is a lookup table: any function in one bootstrapping. Gates are a sum plus a table.
- 5 Bootstrap after every gate: no depth limit, one small message at a time.