F1: A FAST AND PROGRAMMABLE ACCELERATOR FOR FULLY HOMOMORPHIC ENCRYPTION

1/12/2022

NICK GENISE

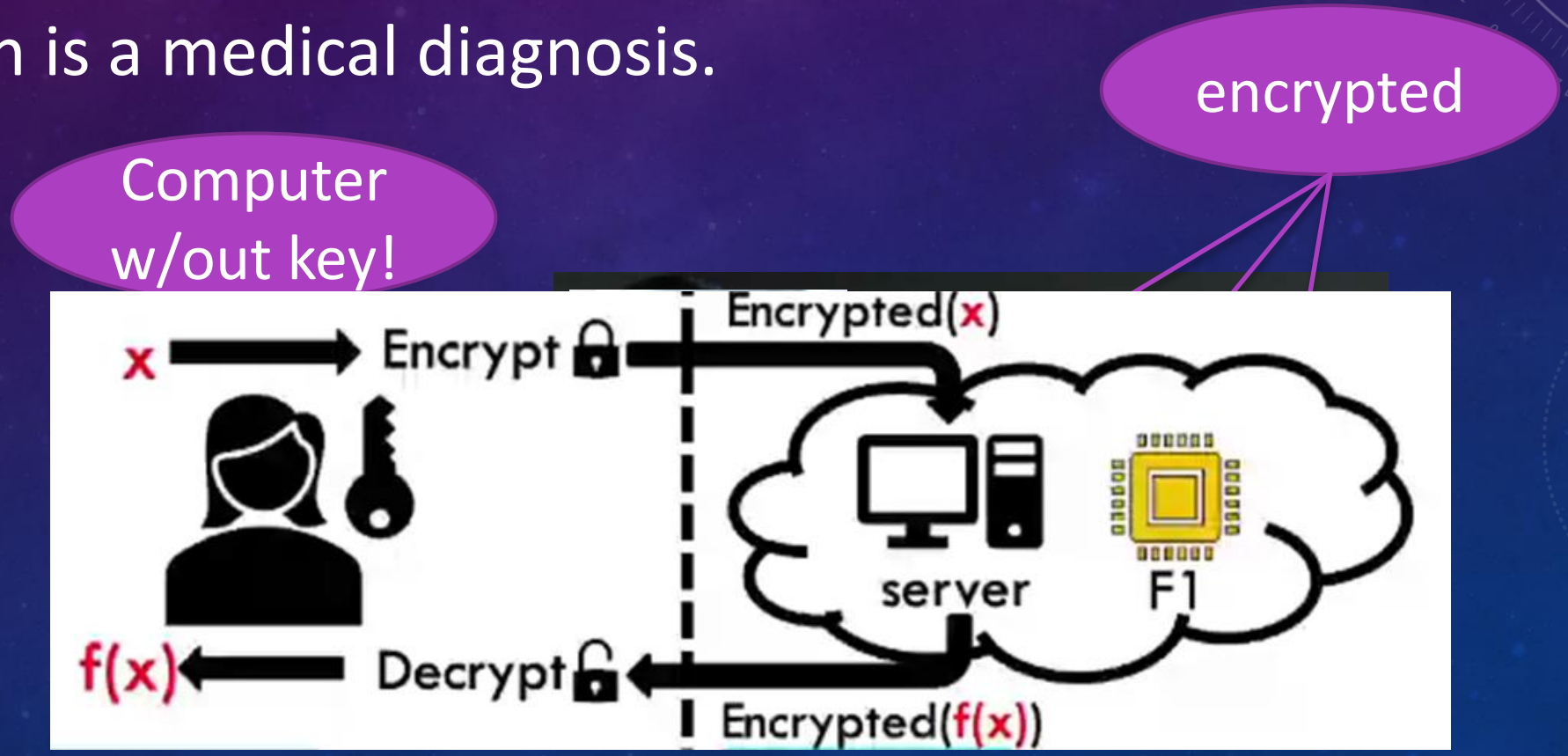
WORK DONE AT SRI INTERNATIONAL

JOINT WITH AXEL FELDMANN, NIKOLA SAMARDZIC, ALEKSANDAR KRASTEV, SRINI DEVADAS, RON DRESLINSKI, KARIM ELDEFRAWY, CHRISTOPHER PEIKERT, DANIEL SANCHEZ

FUNDED BY DARPA DPRIVE

FULLY HOMOMORPHIC ENCRYPTION (FHE)

Example, a patient encrypts their symptoms, and the function is a medical diagnosis.



A SHORT HISTORY OF FHE

1978: Rivest, Adleman, and Dertouzos propose “privacy homomorphism”. $Enc(m) \mapsto Enc(f(m))$.

2009: Gentry finds the first *fully homomorphic encryption* scheme. Bases it on a lattice-based hardness assumption.

2011: Brakerski, Gentry, and Vaikuntanathan propose an FHE scheme based on the popular (Ring) Learning with Errors problem.

Today: FHE was 100,000x slower than unencrypted computation, but recent efforts are speeding this up using application specific integrated circuit (ASICs).

(R)LWE

LEARNING WITH ERRORS (LWE)

Let $n \in \mathbb{N}$ be in the hundreds, q a function of n, and χ be a "small" distribution over $\mathbb{Z}_q$.

In 2005, Regev showed the following distribution is pseudorandom assuming worst-case lattice problems are hard for quantum computers:

$$(\boldsymbol{a}_i, \langle \boldsymbol{a}_i, \boldsymbol{s} \rangle + e_i).$$

where $\boldsymbol{a}_i \leftarrow \mathcal{U}(\mathbb{Z}_q^n)$ are uniformly random vectors, $e_i \leftarrow \chi$, $\boldsymbol{s} \leftarrow \chi^n$

Search version: can you recover $\boldsymbol{s}$ given noisy inner-products?

LEARNING WITH ERRORS (LWE), ENCRYPTION SCHEME

Key Gen. : $\boldsymbol{A} \leftarrow \mathbb{Z}_q^{m \times n}$ for $m \sim n\ log(q)$, $\boldsymbol{s}$ as before and $\boldsymbol{e} \leftarrow \chi^m$

$$\text{pk} = (\boldsymbol{A}, \boldsymbol{As} + \boldsymbol{e}) = (\boldsymbol{A}, \boldsymbol{b}),\ \text{sk} = \boldsymbol{s}.$$

Enc(m): Sample a random binary vector $\boldsymbol{u}$ and return

$$\text{ct} := (\boldsymbol{u^t A}, \boldsymbol{u^t b}\ + \beta m) = (\boldsymbol{c_0}, c_1)$$

where β is a scaling, decoding factor.

Dec(ct, sk): $c_1 - \boldsymbol{s^t c_0} \approx \beta m$. Can recover message if $\beta >$ noise.

Note, linearly-homomorphic! (with growing noise)

RING LEARNING WITH ERRORS (RLWE)

Problem with LWE: The public keys are huge! Roughly $\theta(\lambda^2 log^2(\lambda))$ bits.

In 2009, partially motivated by Gentry's FHE breakthrough, Lyubashevsky, Peikert, and Regev proved a more compact version of LWE is secure assuming the hardness of worst-case problems on ideal lattices.

Eventual result: efficient lattice-based encryption and a base scheme for FHE!

RING LEARNING WITH ERRORS (RLWE), ENCRYPTION SCHEME

Polynomials: $R_q \coloneqq \mathbb{Z}_q[x]/(x^N + 1)$ where $N = 2^{\text{some power}} \geq 1024$.

χ is a distribution over R_q by the sampling the same distribution over polynomial coefficients.

Key Gen: $a \leftarrow R_q$ and $s, e \leftarrow \chi$

$$\text{pk} = (a, as + e) = (a, b), \text{ sk} = s.$$

Enc(m): Sample a random binary polynomial u and return

$$\text{ct} := (ua, ub + \beta m) = (c_0, c_1) \in R_q \times R_q$$

where β is a scaling, decoding factor.

Note, linearly-homomorphic! (with growing noise)

BGV

ENCRYPTION AND DECRYPTION

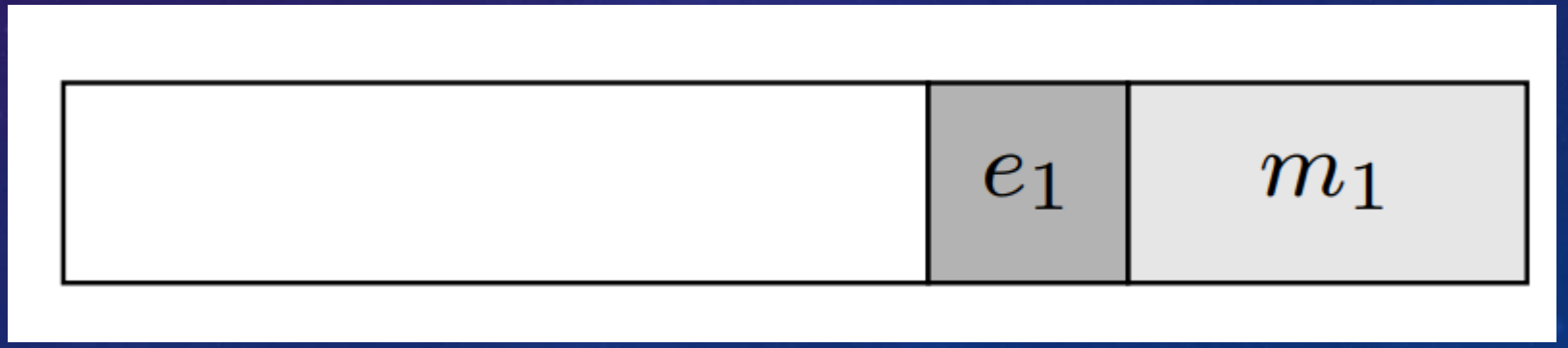
A BGV encryption is an RLWE encryption with the message in the “least sig. bits:”

Plaintext space = $R_p = \mathbb{Z}_p[x]/(x^N + 1)$

$E_s(m) = Enc(m)$: ct $= (c_0, c_1) = (as + pe + m, -a) \in R_q \times R_q$

$p \ll q$ with the former prime in $\mathbb{Z}$

Note: $c_0 + c_1 s \; mod \; q = pe + m$



HOMOMORPHIC ADDITION

$$E_s(m) + E_s(m') = E_s(m + m')$$

$$(c_0, c_1) + (c_0', c_1') = ([a + a']s + p[e + e'] + m, -a - a')$$

Noise magnitude grows by ~ 1 bit.

HOMOMORPHIC MULTIPLICATION

Idea: view through the decryption equation, $pe + m = c_0 + c_1 s \; mod \; q$. Then,

$$mm' + p(\text{small}) = c_0 c_0' + s[c_0 c_1' + c_0' c_1] + s^2 c_1 c_1' \; mod \; q.$$

Or, $(c_0 c_0', c_0 c_1' + c_0' c_1, c_1 c_1')$ encrypts mm' under $(1, s, s^2)$.

We use an encryption of s^2 to "Relinear-ize" back to an encryption under s!

Noise size squares (new noise $\approx ee'$ aka double the noise bits)!

POLYNOMIALS/DATA TYPES: CRT ON COEFFICIENTS

So far, we've ignored the structure of the modulus q.

In practice, we take $q = q_1 q_2 \cdots q_L$ where each q_i is a distinct prime of machine-size (32 or 64 bits).

Then, we can represent a coefficient of a polynomial in R_q via the Chinese remainder theorem: $\mathbb{Z}_q \cong \mathbb{Z}_{q_1} \times \mathbb{Z}_{q_2} \times \cdots \times \mathbb{Z}_{q_L}$ via

$$x \in \mathbb{Z}_q, \qquad CRT(x) = (x \bmod q_1, x \bmod q_2, \ldots, x \bmod q_L)$$

Addition and multiplication in-parallel!

POLYNOMIALS/DATA TYPES: CRT (NTT) ON POLYNOMIALS

Using the CRT on coefficients allows us to represent R_q as

$$R_q \cong R_{q_1} \times R_{q_2} \times \cdots \times R_{q_L}$$

What about each $R_{q_i} = \mathbb{Z}_{q_i}[x]/(x^N + 1)$?

Do the FFT!

The FFT modulo a prime number is called the NTT (number-theoretic transform).

POLYNOMIALS/DATA TYPES: CRT (NTT) ON POLYNOMIALS CONT.

If each $q_i \equiv 1 \; mod \; 2N$, then $\mathbb{Z}_{q_i}$ contains a multiplicative subgroup of size $2N$ which is the roots of $x^N + 1$ ($2N$-th roots of unity).

The NTT is the evaluation at all N of these points:

$$f \in R_{q_i}, NTT(f) = (f(\omega), f(\omega^3), \dots, f(\omega^{2N-1})).$$

Addition and multiplication over R_{q_i} in in-parallel!

Just like the FFT, the roots of unity allow us to compute this in time $O(N \cdot \log(N))$.

CRT AND NTT: MAIN POINTS

CRT allows us to represent $f \in R_q$ as L polynomials in

$$R_{q_1} \times R_{q_2} \times \cdots \times R_{q_L}.$$

NTT allows to represent R_{q_1} as $\mathbb{Z}_{q_i}^N$.

This is the "double-CRT" form. It allows polynomial addition and multiplication to be done in parallel over finite fields.

Much of the overhead in (software) FHE is computing forward and inverse NTT's. This is because most operations require us to switch between NTT form and coefficient form.

PLAINTEXT SLOTS

The same idea applies to p. That is, if $p \equiv 1 \; mod \; 2N$, then the NTT can be applied to the plaintext space $R_p \cong \mathbb{Z}_p^N$.

We call this a "packed" ciphertext since each ct can encrypt N elements in $\mathbb{Z}_p$. Each plaintext element is in a plaintext "slot."

For large computations, we need cross-talk between slots. This is done by signed permutations (automorphisms) on the ct polynomials. Note, these automorphisms are transitive on the slots.

$\sigma(as + pe + m) = \sigma(a)\sigma(s) + p\sigma(e) + \sigma(m)$.

Encryption under the wrong key! Need to key-switch again :/.

GENERAL CASE FOR PLAINTEXT SLOTS, POWER OF TWO DIMENSION

Let p be a prime not equal to 2. Then $x^n + 1$ factors into irreducible polynomials of the *same degree, d,* over the finite field $\mathbb{Z}_p$!

$$x^n + 1 = \prod_{i \in \{1,\dots,l\}} F_i(x)$$

The degree d is the smallest number s.t. $p \equiv 1 \; mod \; 2n$. Then $2n = ld$.

WHAT ARE THE GENERAL SLOTS?

Our old friend, the Chinese Remainder Theorem, gives us the answer:

$$R_p = \frac{\mathbb{Z}_p[x]}{(x^n+1)} \cong \frac{\mathbb{Z}_p[x]}{F_1(x)} \otimes \frac{\mathbb{Z}_p[x]}{F_2(x)} \otimes \cdots \otimes \frac{\mathbb{Z}_p[x]}{F_l(x)} \cong GF(p^d)^l$$

This map is given by evaluating a polynomial in R_p by a root of each polynomial. Let η_1 be a root to $F_1(x)$ and κ_i be distinct coset representatives of $\mathbb{Z}_m^*/\langle p \rangle$.

$$f(x) \mapsto \left(f\left(\eta_1^{\kappa_1}\right), f(\eta_2^2), \ldots, f\left(\eta_l^{\kappa_l}\right)\right)$$

This can be done with a DFT over a finite field!

COMPARISON: NTT V.S. GENERAL

NTT: When $q \equiv 1 \; mod \; 2n$, $x^n + 1 = \prod_{i \in \{1,3,\ldots,2n-1\}} (x - \omega^i)$ splits completely into degree-1 factors. Then,

$$R_q = \frac{\mathbb{Z}_q[x]}{(x^n+1)} \cong \frac{\mathbb{Z}_q[x]}{(x-\omega^1)} \otimes \frac{\mathbb{Z}_q[x]}{(x-\omega^3)} \otimes \cdots \otimes \frac{\mathbb{Z}_q[x]}{(x-\omega^{2n-1})} \cong \mathbb{Z}_q^n$$

General: When p is a prime not equal to 2, $x^n + 1 = \prod_{i \in \{1,\ldots,l\}} F_i(x)$ factors into same-degree irr. polynomials and

$$R_p = \frac{\mathbb{Z}_p[x]}{(x^n+1)} \cong \frac{\mathbb{Z}_p[x]}{F_1(x)} \otimes \frac{\mathbb{Z}_p[x]}{F_2(x)} \otimes \cdots \otimes \frac{\mathbb{Z}_p[x]}{F_l(x)} \cong GF(p^d)^l$$

PLAINTEXT SLOTS: MAIN POINTS

We apply signed permutations to enable plaintext “cross-talk.” Usually we pick κ_i so these are **rotations on the slots**.

After each permutation, we need to do a key-switching operation (usually represented as a matrix-vector multiply).

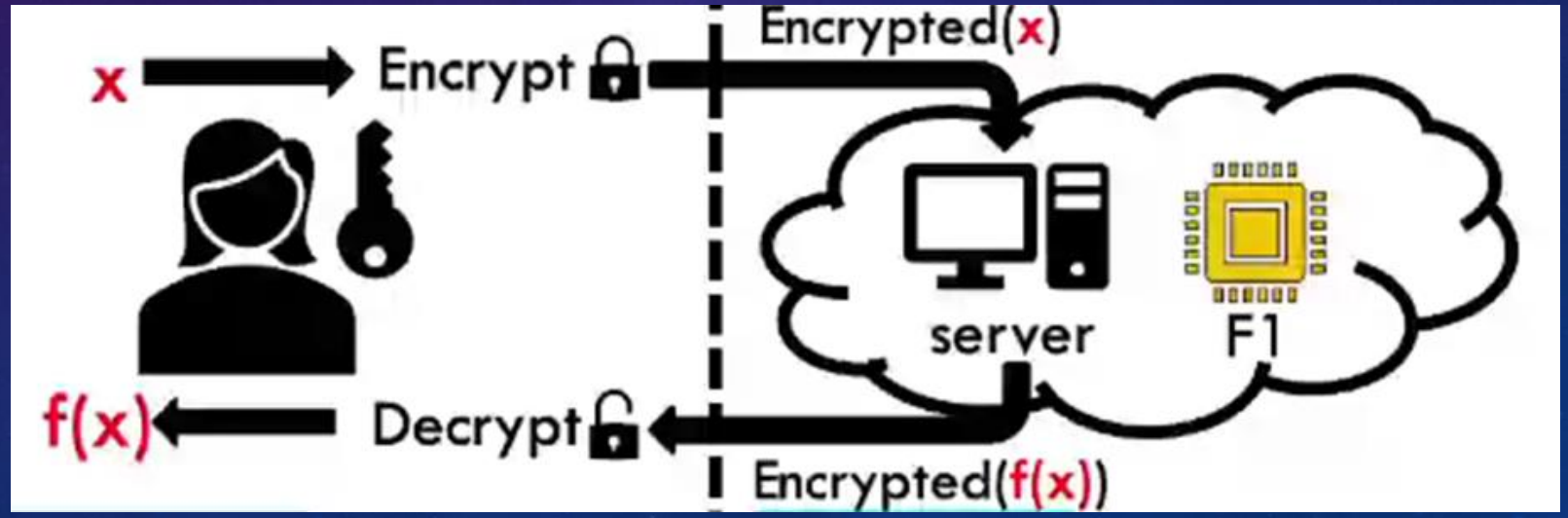
Key-switching cannot by done in NTT form (evaluation rep.).

THE FHE INTERFACE (SERVER SIDE)

Ciphertext **Add**: Add the polynomials $mod\ q$, $(c_0, c_1) + (c_0{}', c_1{}')$.

Ciphertext **Multiply**: Compute $(c_0 c_0', c_0 c_1' + c_0' c_1, c_1 c_1')$ then relinearize to $(c_0{}'', c_1{}'')$.

Ciphertext **Rotate**: Compute $(\sigma(c_0), \sigma(c_1))$ then key-switch.

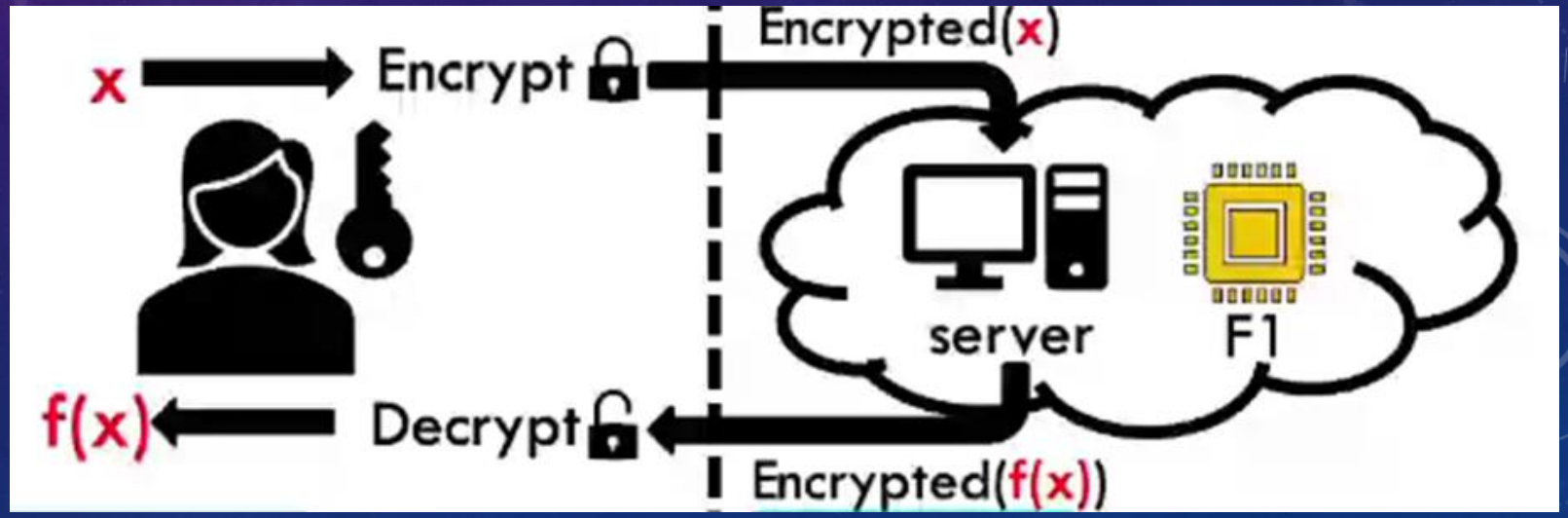


ACCELERATING FHE WITH ASICS (F1)

CHALLENGES TO ACCELERATING

You want a chip that:

1. stores multiple ciphertexts and hints,
2. accelerates FHE operations like ADD, MULT, and ROTATE,
3. reduces bottlenecks to data-movement or high re-use,
4. with few functional units with high throughput.



OVERVIEW OF F1

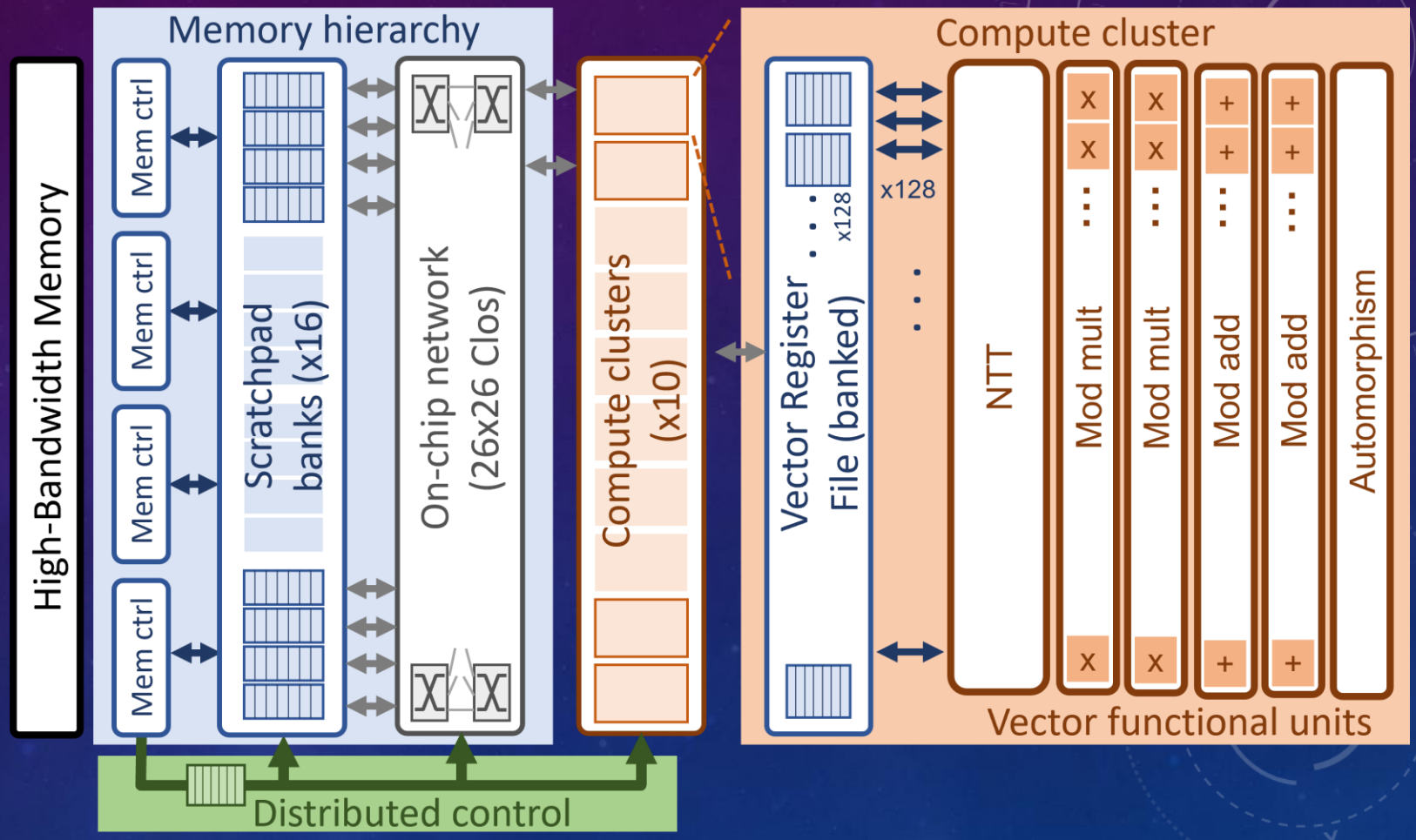
Small Design: 151 mm^2

Large scratch pad: 64 MB

Compute clusters with FUs for NTT, mod mult, mod add, and automorphism.

Each FU operates on an RNS polynomial, or a (1K-16K) vector of 24-bit values.

Note, for $N = 16K$, $L = 16$, then a ct is $2MB$. (A KSH is 32 MB.)



NTT-FRIENDLY MONTGOMERY MULTIPLIERS

Recall that the RNS moduli, q_i, are of the form $q_i \equiv 1 \; mod \; 2N$.

We use a Word-Level Montgomery multiplier, word size ~11 bits, for our modular multipliers.

This algorithm was initially used by Can Mert, Ozturk, and Savas in accelerating NTTs using FPGAs. (eprint)

We modify it to q_i s.t. $q_i \equiv -1 \; mod \; 2^{16}$. This allows us to remove one multiplier and save area and power.

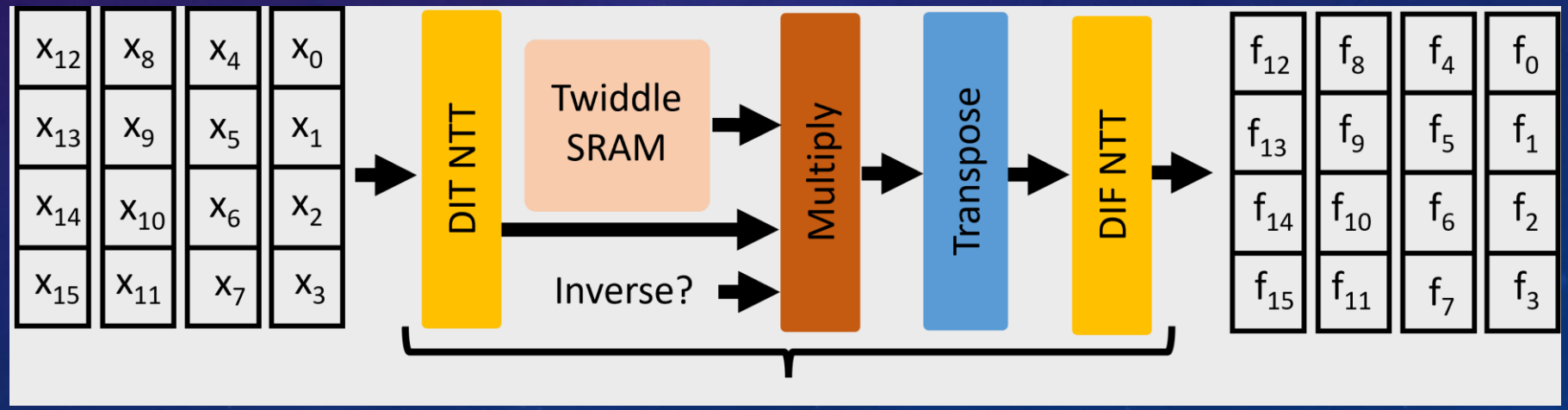
4 STEP NTT

Recall, the NTT is an FFT for modular arithmetic. Given $f \in R/q_i R$, compute

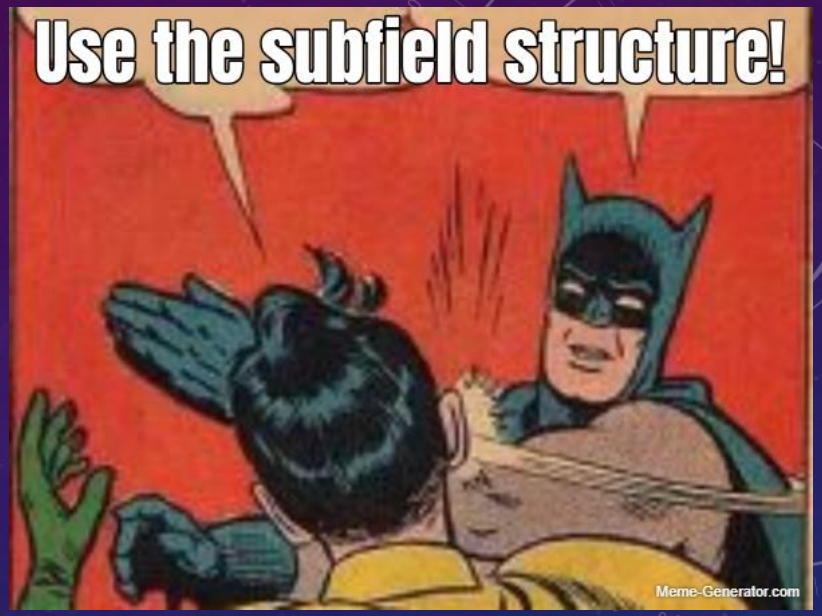
$$NTT(f) = \left(f(\omega), f(\omega^3), \ldots, f(\omega^{2N-1})\right)$$

where ω is a primitive $2N$th root of unity mod q_i.

We do both the forward and inverse NTT with the **same hardware.** This is by using a 4 Step NTT and mixing DIT and DIF butterflies. Our NTT is only for $E = 128$ elements. So, we reduce all NTT to 128-point NTTs.



AUTOMORPHISM FUNCTIONAL UNITS



Each automorphism is $\sigma\big(f(x)\big) \mapsto f(x^i)$ for some $i \in \mathbb{Z}_{2N}^*$ (odd).

The automorphisms form a group and are generated by $i = -1$ and $i = 3$.

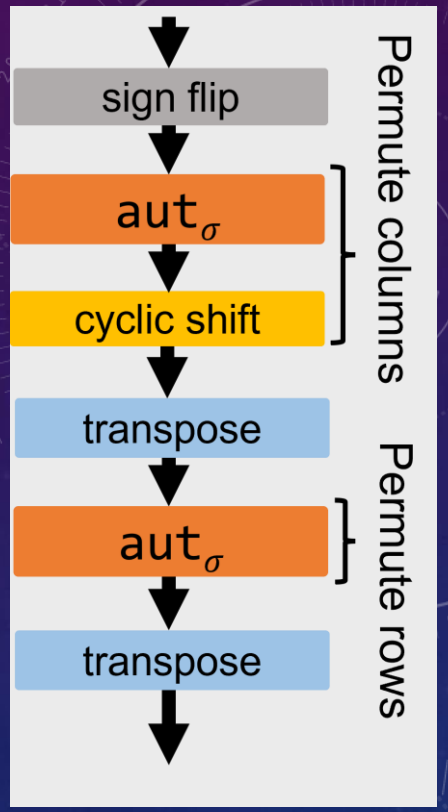
Standard hardware approaches fail! A general $16K$ permutation is too complex.

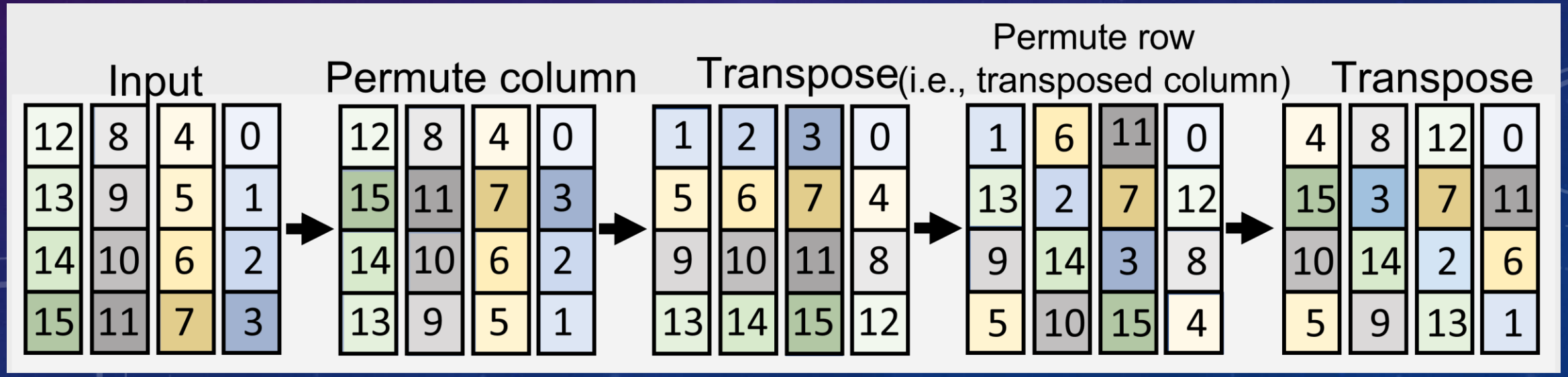
Our functional unit reduces to how an automorphism acts on a degree $E = 128$ subfield and the basis. For example, in $N = 256$, we have $f(x) = f_0(x^2) + xf_1(x^2)$. (Basis is $\{1, x\}$ here.)

$$\sigma\big(f(x)\big) = \sigma\big(f_0(x^2)\big) + \sigma(x)\sigma\big(f_1(x^2)\big)$$

AUTOMORPHISM FUNCTIONAL UNIT

$$\sigma\big(f(x)\big) = \sigma\big(f_0(x^2)\big) + \sigma(x)\sigma\big(f_1(x^2)\big)$$





PERFORMANCE AND FUTURE WORK

F1 Accelerates FHE programs by $\mathbf{5400} \times -\mathbf{17000} \times$ compared to software solutions run on a standard CPU.

Future work: accelerate more complex FHE operations (e.g., key-switching), focus on bootstrapping, and accelerate FHEW/TFHE schemes. (F1 is best used on BGV, BFV, and CKKS.)

THANK YOU!