HEBridge: Connecting Arithmetic and Logic Operations in FV-style HE Schemes

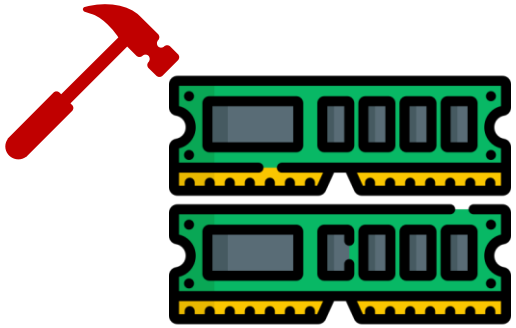
Yancheng Zhang[1], Xun Chen[2], **Qian Lou[1]**

[1]University of Central Florida

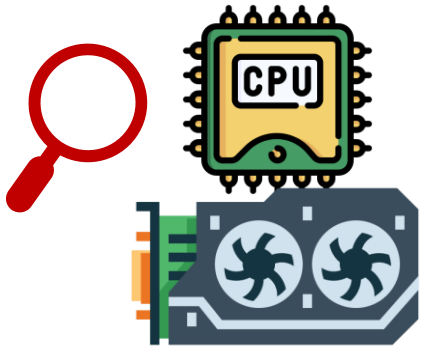
[2]Samsung Research America

Code: https://github.com/UCF-Lou-Lab-PET/HE-Bridge

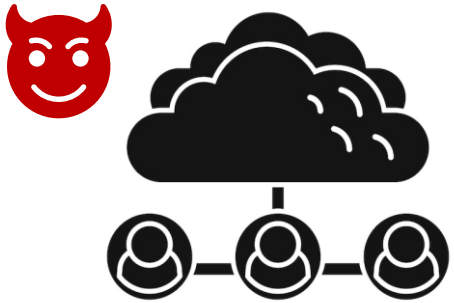
Privacy Attacks on "Data in Use"



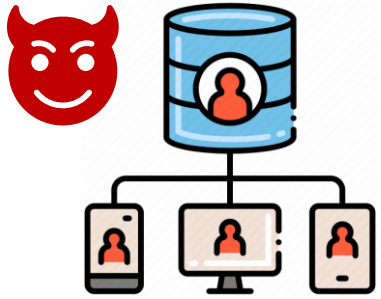
Run-time Memory Scraping



Side Channel on Processors



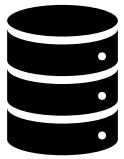
Untrusted Cloud Computing



Database Breach in Use

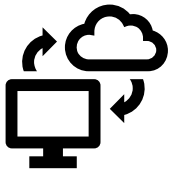
Secure "Data in Use" is Still an Open Problem

Data at Rest



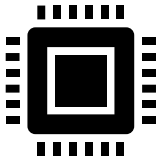
Secure

Data in Transit



Secure

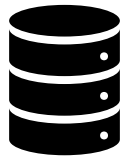
Data in Use



Risk

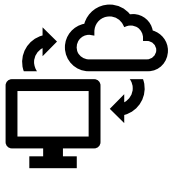
Fully Homomorphic Encryption: Encryption in Use

Data at Rest



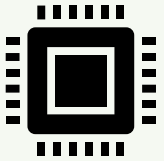
Secure

Data in Transit



Secure

Data in Use



Secure

// *Encryption* by client
1. $[x] = FHE.encrypt(x)$
// Encryption in use on server
2. $[y] = FHE.eval(f, [x])$
// Decryption by client
3. $y = FHE.decrypt([y], s)$

Encryption in use for $y = f(x)$

Fully Homomorphic Encryption

FHE Has Seen a Surge of Interests

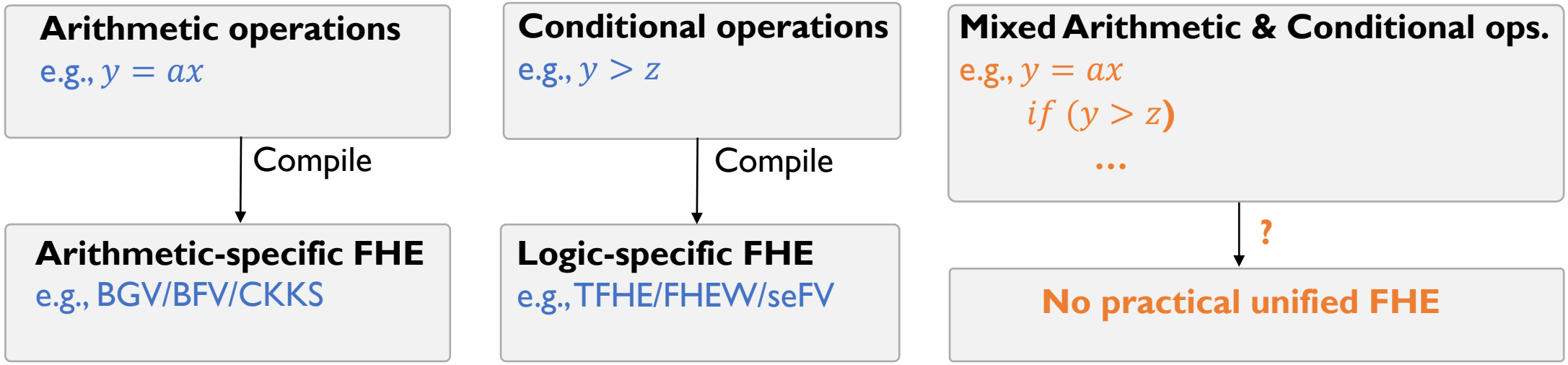
Edge Password Monitor

Live Caller ID Search

Satellite Collision Detection

Key Challenge for FHE's Broader Deployment: **Universality**

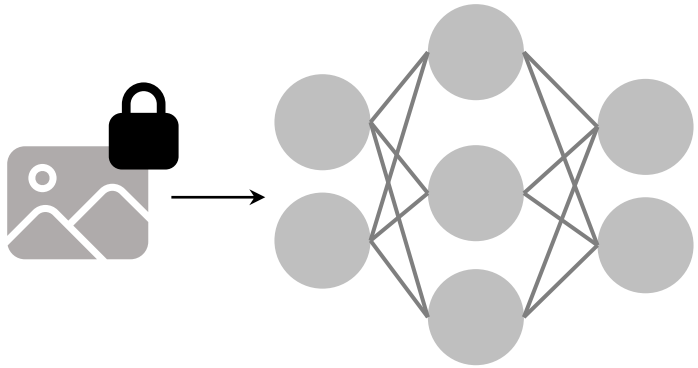
- Universality of a FHE scheme: A unified FHE scheme can execute **mixed arithmetic and logic operations** without decryption.



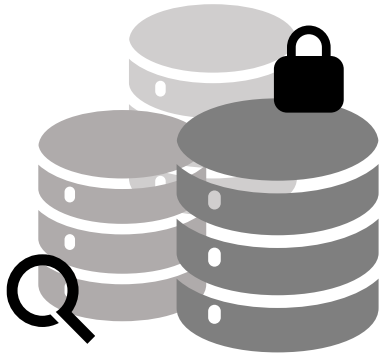
There is no universal FHE scheme that operates on a word-wise basis

Our Aim: Designing Universal and Efficient FHE Scheme

- Enabling important applications based on mixed arithmetic and logic operations.



Private machine learning

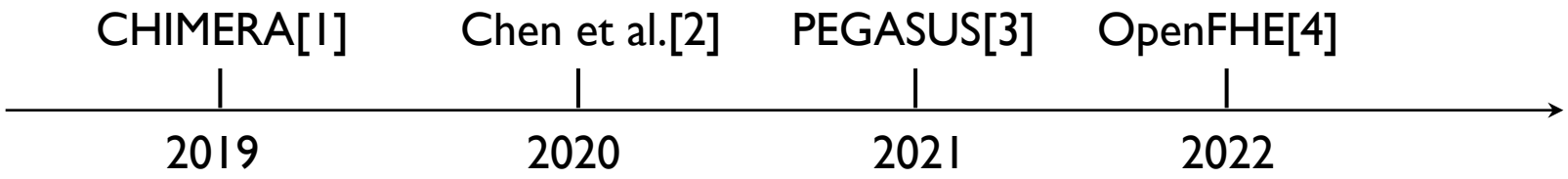


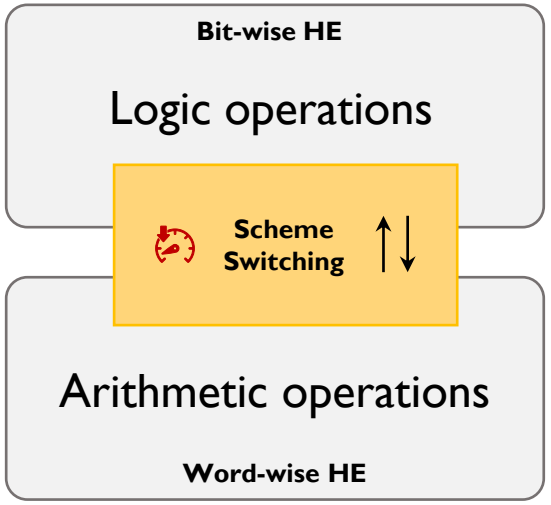
Private database query

Private genomics analysis

Previous: (1) Scheme Switching for Universality?

- Bit-wise FHE schemes for logic operations while the word-wise FHE scheme for arithmetic operations.





Issues:

- The switching is expensive, e.g., more than 10GB key size[1]
- Low bit-width input
  - Small domains like $[-8,8]$
  - The complexity grows exponentially to the input bit width

[1] CHIMERA: Combining Ring-LWE-based Fully Homomorphic Encryption Schemes

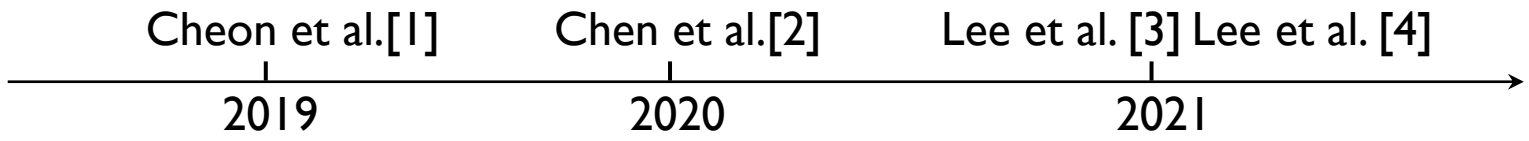
[2] Efficient Homomorphic Conversion Between (Ring) LWE Ciphertexts

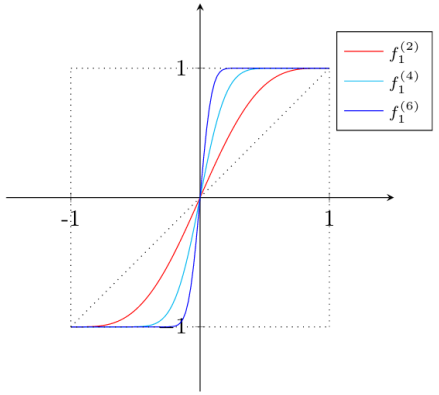
[3] PEGASUS: Bridging Polynomial and Non-polynomial Evaluations in Homomorphic Encryption

[4] OpenFHE: Open-Source Fully Homomorphic Encryption Library

Previous: (2) Polynomial Approximation for Universality?

- Approximate logic operations with polynomials





Example: use a polynomial to approximate the sign function

- $f(x) = 2.25x - 2.8125x^3 + 1.6875x^5 - 0.5625x^7 + 0.0625x^9$
- $f(-0.6) = -0.9395 \approx -1 = sign(-0.6)$

Issues:

- Approximation is always not exactly accurate
- It only works well on small intervals such as $[-1,1]$

[1] Numerical method for comparison on homomorphically encrypted numbers

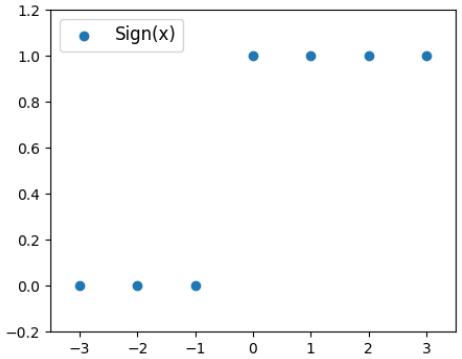
[2] Efficient homomorphic comparison methods with optimal complexity

[3] Minimax Approximation of Sign Function by Composite Polynomial for Homomorphic Comparison

[4] Optimization of Homomorphic Comparison Algorithm on RNS-CKKS Scheme

Previous: (3.1) Polynomial Interpolation on All Space?

- Interpolate Polynomial on All the Points of $\mathbb{Z}_{p^r}$



Example: use a polynomial to interpolate the sign function (accurate)

- $f(x) = -4x - 6x^3 - x^5 + 4x^6 \ (mod\ 7)$
- $f(-2) = -1 = sign(-2)$

Issues:

- It only works well for small plaintext space $\mathbb{Z}_{p^r}$
  - Degree-p^r interpolation polynomial

[1] On the Efficiency of FHE-based Private Queries

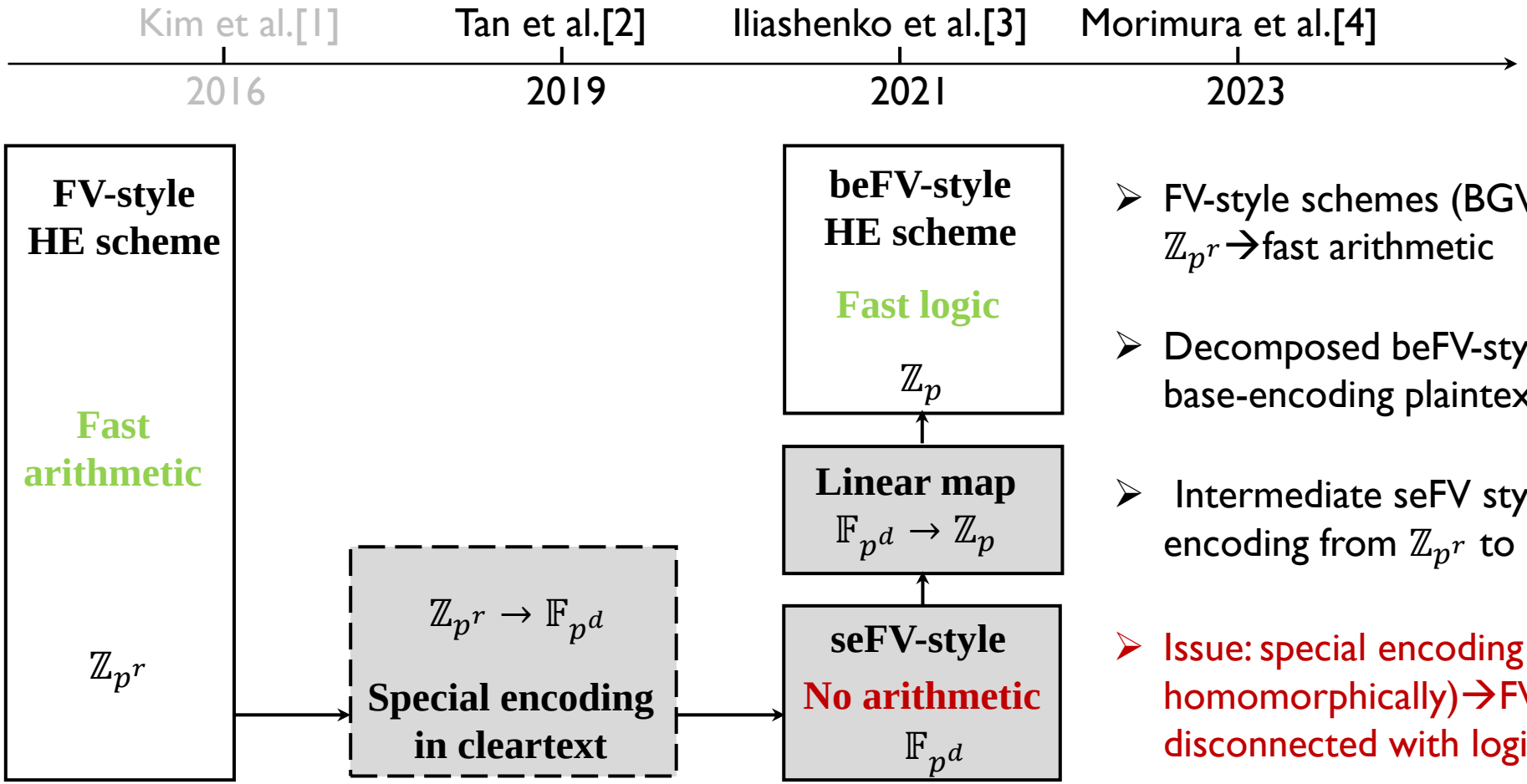
[2] Efficient Private Comparison Queries Over Encrypted Databases

[3] Faster homomorphic comparison operations for BGV and BFV

[4] Accelerating Polynomial Evaluation for Integer-wise Homomorphic Comparison and Division

Previous: (3.2) Polynomial Interpolation on Decomposed Space?

- Interpolate Polynomial on the decomposed Points of $\mathbb{Z}_p$ (base encoding)

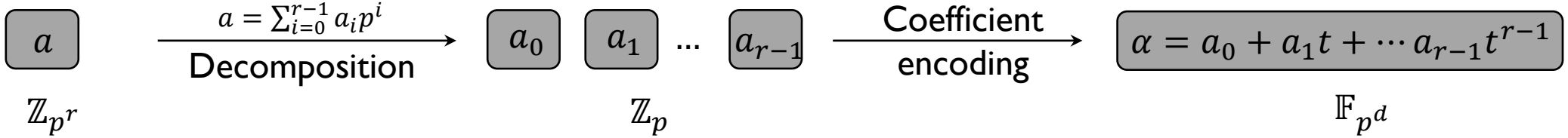


- FV-style schemes (BGV/BFV) have plaintext space $\mathbb{Z}_{p^r}$ → fast arithmetic
- Decomposed beFV-style schemes [2][3][4] have base-encoding plaintext space $\mathbb{Z}_p$ → fast logic
- Intermediate seFV style scheme requires a special encoding from $\mathbb{Z}_{p^r}$ to $\mathbb{F}_{p^d}$ in cleartext
- Issue: special encoding is done in plaintext (not homomorphically) → FV-style arithmetic FHE is disconnected with logic beFV

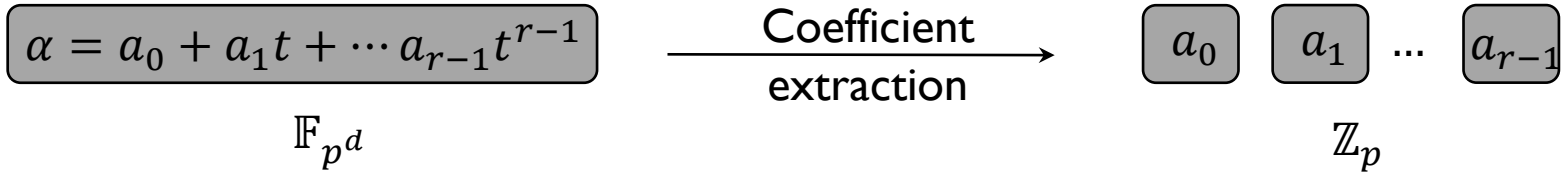
Previous: (3.2) Polynomial Interpolation on Decomposed Space?

➢ Workflow Illustration

Pre-encryption decomposition in seFV: Before encryption, a big integer in $\mathbb{Z}_{p^r}$ is decomposed into r base-p digits and then mapped to an element of the extension field $\mathbb{F}_{p^d}$.

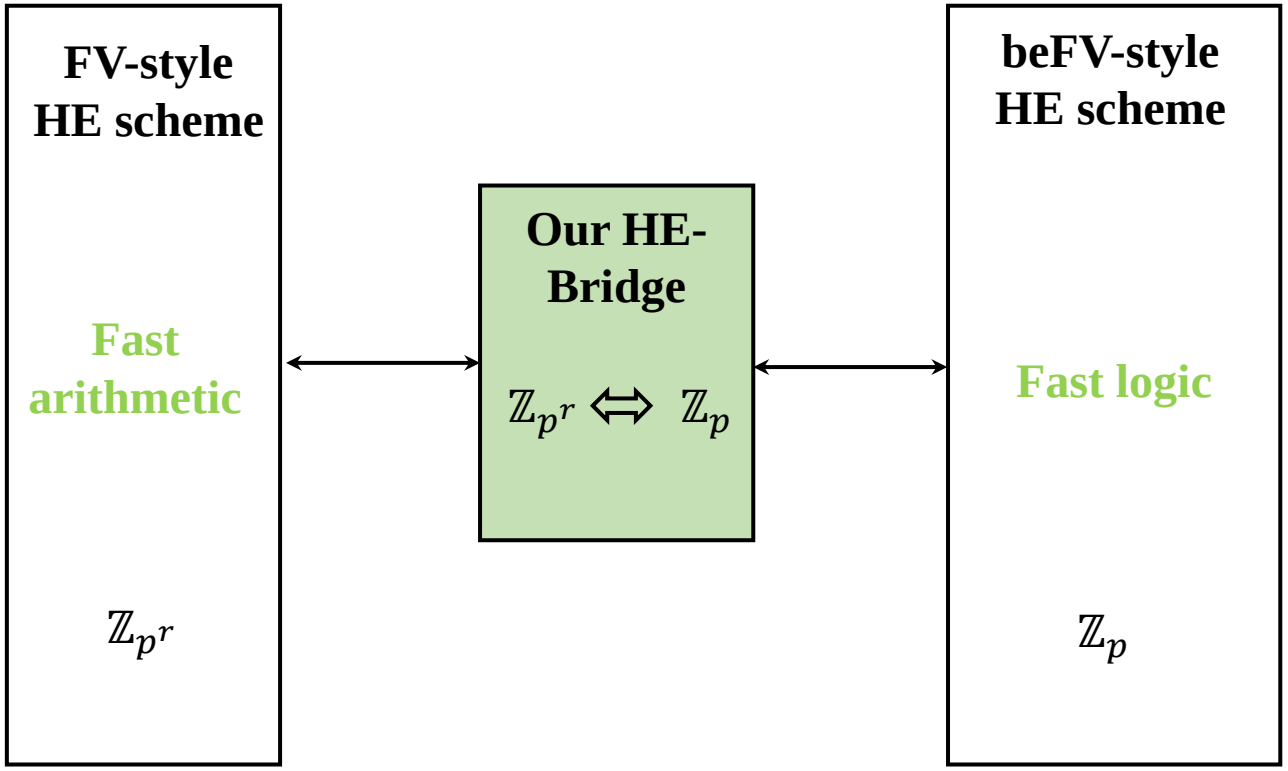


Coefficient extraction via homomorphic linear map: The base-p components of α is extracted homomorphically via a $\mathbb{F}_p$-linear map.

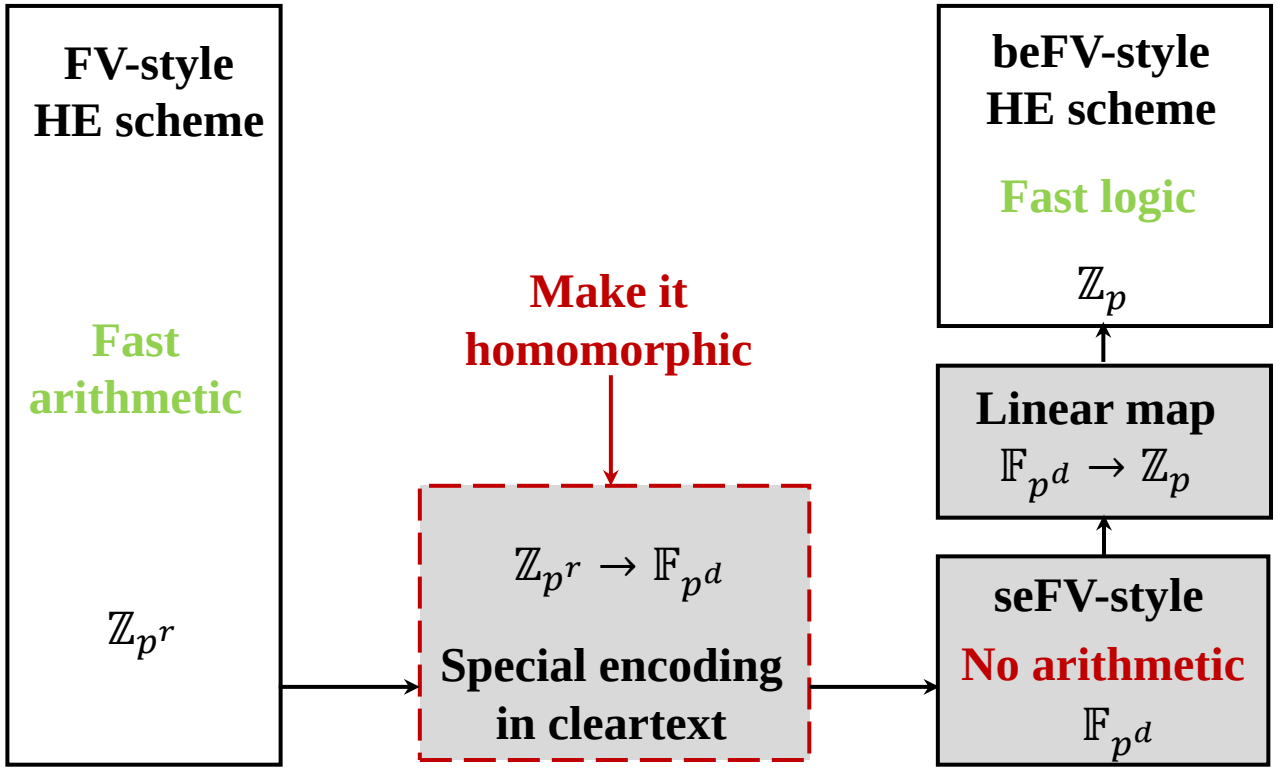


Digit-wise comparison: in beFV: Once the base-p digits are obtained, a can be compared with another integer b digit by digit via interpolating over $\mathbb{Z}_p$

Research Question: Can We Design an Efficient Bridge between FV-style HE and beFV-style HE?



Naïve method: Achieving Homomorphic Special Encoding

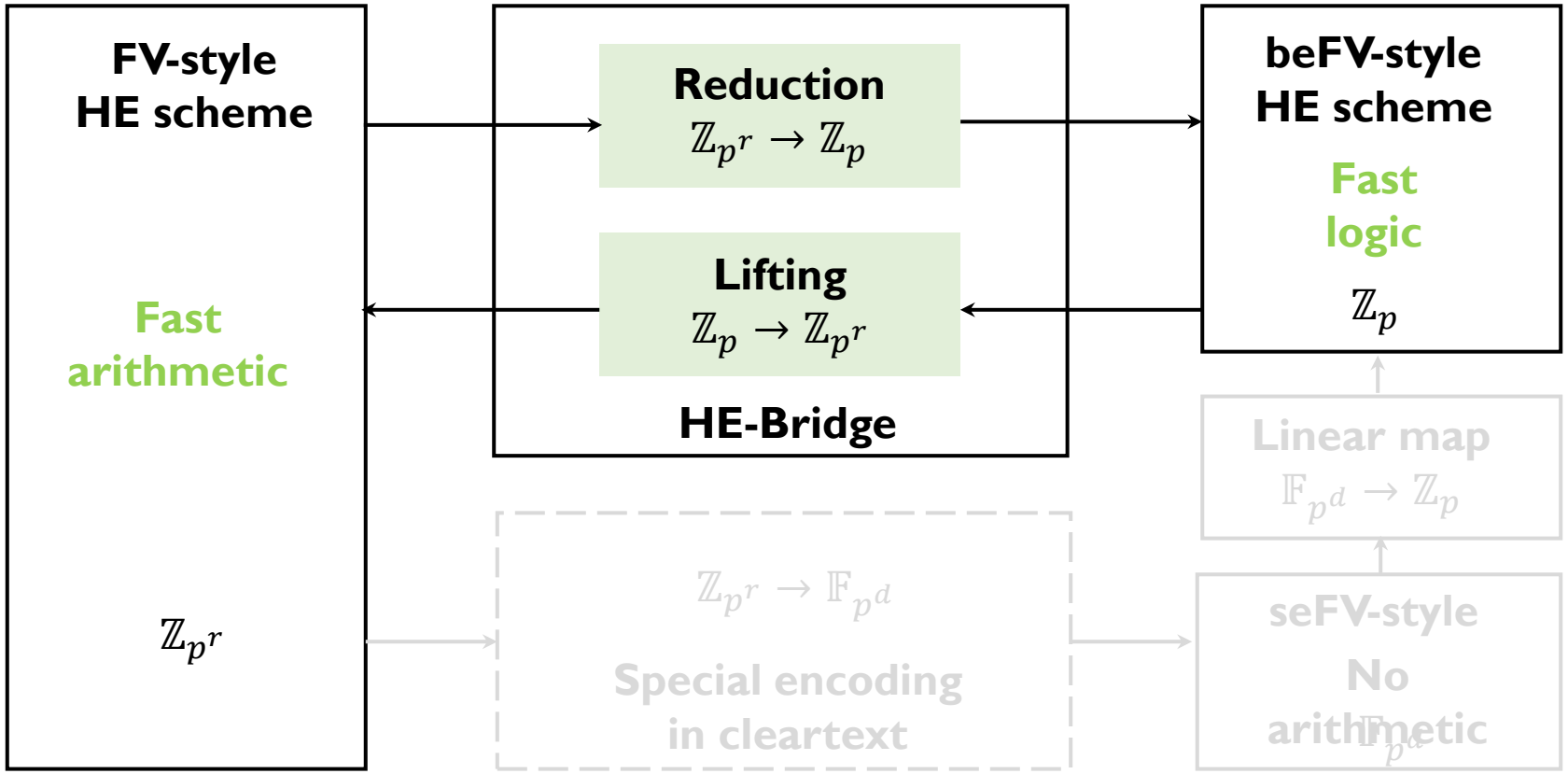


Issues:

- Circuitous bridge: why don't directly choose short-cut bridge between $\mathbb{Z}_{p^r}$ and $\mathbb{Z}_p$?
- One-direction bridge: it does not support well for the reverse direction

Our HE-Bridge: Short-cut and Bi-directional Bridge

- Short-cut: HEBridge connects FV and beFV directly without special encoding
- Bi-directional: our HE-Bridge is designed with reduction and lifting functions



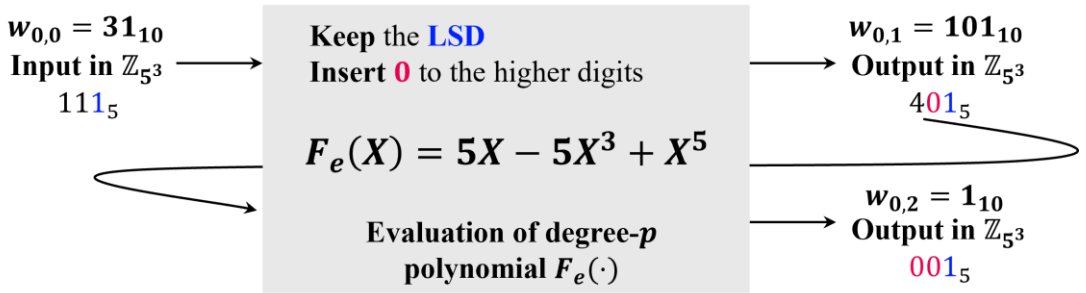
HE-Bridge: Reduction

- The reduction from FV to beFV can be achieved by homomorphic digit decomposition.
- Our reduction function is inspired by several observations from prior works.

1. Squaring an integer keeps its LSB unchanged.

$$w_{0,0} = 1010 \xrightarrow{\text{Square}} w_{0,1} = 0100 \xrightarrow{\text{Square}} w_{0,2} = 1000 \xrightarrow{\text{Square}} w_{0,3} = 0000$$
$$w_{1,1} = 011 \xrightarrow{\text{Square}} w_{1,2} = 001 \xrightarrow{\text{Square}} w_{1,3} = 001$$
$$w_{2,2} = 00 \xrightarrow{\text{Square}} w_{2,3} = 00$$
$$w_{3,3} = 1$$

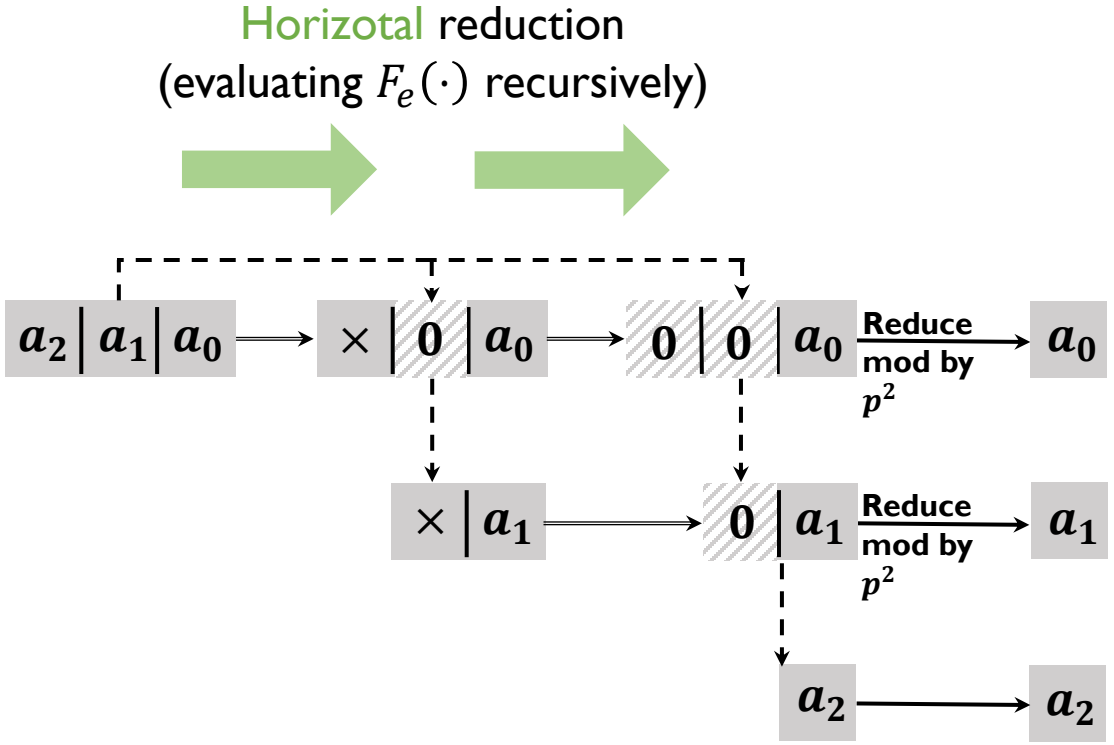
2. The square function in $\mathbb{Z}_2$ can be generalized to a degree-p polynomial, $F_e(\cdot)$, in $\mathbb{Z}_p$



HE-Bridge: Reduction

Based on the polynomial $F_e(\cdot)$, we can extract the degree-p digits from an integer $\boldsymbol{a} \in \mathbb{Z}_{p^r}$.

- HEBridge's reduction consists of horizontal reduction, vertical reduction and modulus reduction.



Algorithm 2: Reduction from $\mathbb{Z}_{p^r}$ to $\mathbb{Z}_p$

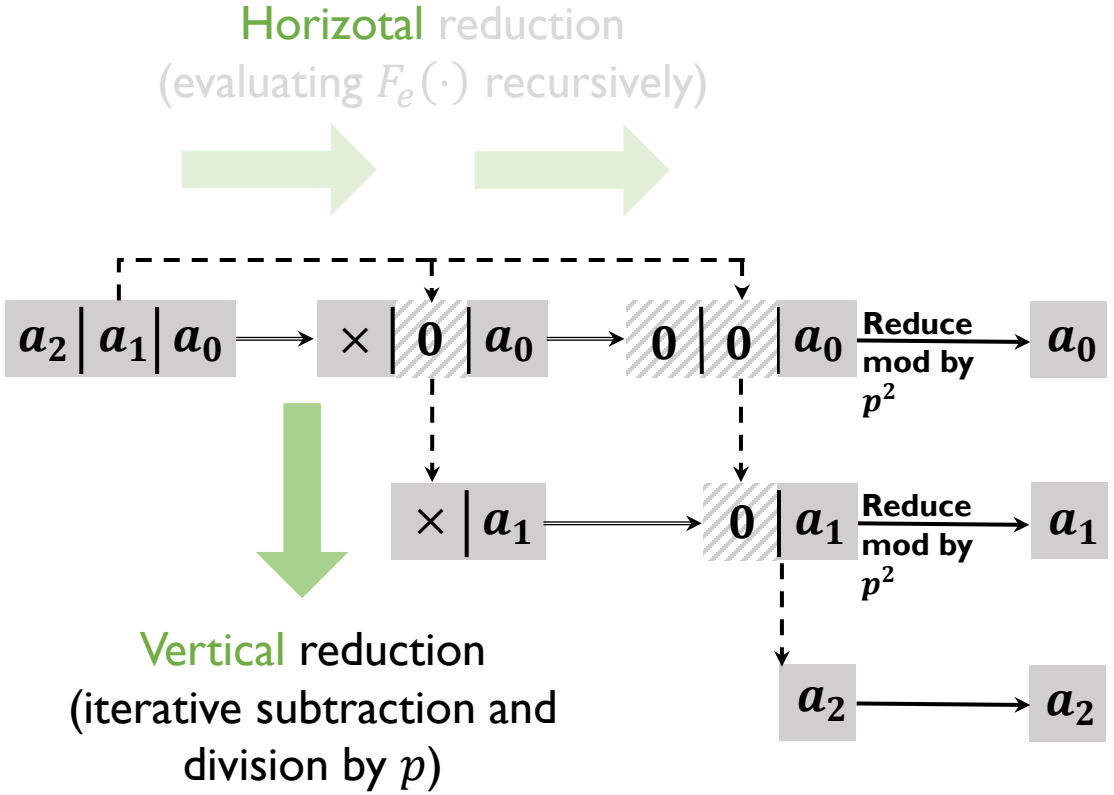
Input : The integer message $z \pmod{p^r}$.
Output: All digits of z, $\{z_i\}_{i=0}^{r-1} \pmod{p}$ such that $z = \sum_{i=0}^{r-1} z_i p^i$.

$w_{0,0} \leftarrow z$;
for $i \leftarrow 0$ **to** $r-2$ **do**
  for $j \leftarrow i$ **to** $r-2$ **do**
    $w_{i,j+1} \leftarrow F_e(w_{i,j})$; // Horizontal reduction
  $y \leftarrow z$;
  for $j \leftarrow 0$ **to** i **do**
    $y \leftarrow \text{DivideByP}(y - w_{j,i+1})$; // Vertical reduction
  $w_{i+1,i+1} \leftarrow y$;
  $z_i \leftarrow w_{i,r-1}$;
  for $j \leftarrow i$ **to** $r-2$ **do**
    $z_i \leftarrow \text{DivideModByP}(z_i)$; // Modulus reduction
$z_{r-1} \leftarrow w_{r-1,r-1}$;
return $\{z_i\}_{i=0}^{r-1}$

HE-Bridge: Reduction

Based on the polynomial $F_e(\cdot)$, we can extract the degree-p digits from an integer $\boldsymbol{a} \in \mathbb{Z}_{p^r}$.

- This consists of horizontal reduction, vertical reduction and modulus reduction.



Algorithm 2: Reduction from $\mathbb{Z}_{p^r}$ to $\mathbb{Z}_p$

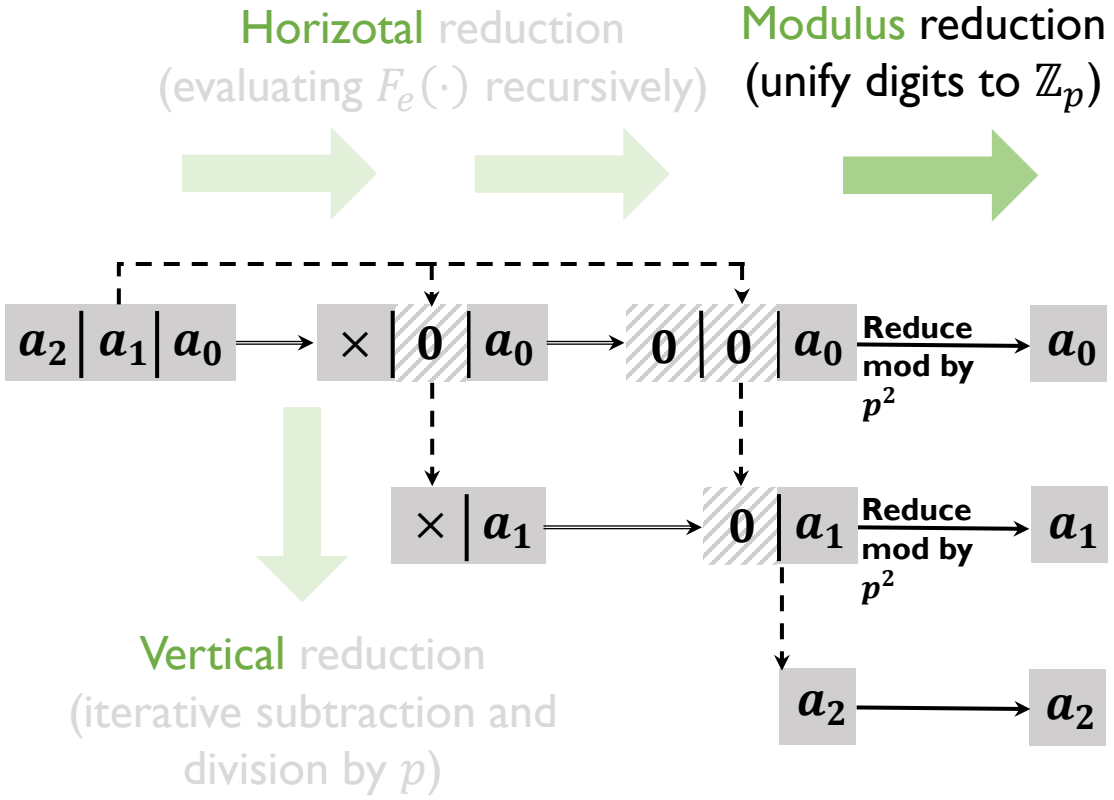
Input : The integer message $z \pmod{p^r}$.
Output: All digits of z, $\{z_i\}_{i=0}^{r-1} \pmod{p}$ such that $z = \sum_{i=0}^{r-1} z_i p^i$.

$w_{0,0} \leftarrow z$;
for $i \leftarrow 0$ **to** $r-2$ **do**
  for $j \leftarrow i$ **to** $r-2$ **do**
    $w_{i,j+1} \leftarrow F_e(w_{i,j})$; // Horizontal reduction
  $y \leftarrow z$;
  for $j \leftarrow 0$ **to** i **do**
    $y \leftarrow \text{DivideByP}(y - w_{j,i+1})$; // Vertical reduction
  $w_{i+1,i+1} \leftarrow y$;
  $z_i \leftarrow w_{i,r-1}$;
  for $j \leftarrow i$ **to** $r-2$ **do**
    $z_i \leftarrow \text{DivideModByP}(z_i)$; // Modulus reduction
$z_{r-1} \leftarrow w_{r-1,r-1}$;
return $\{z_i\}_{i=0}^{r-1}$

HE-Bridge: Reduction

Based on the polynomial $F_e(\cdot)$, we can extract the degree-p digits from an integer $\boldsymbol{a} \in \mathbb{Z}_{p^r}$.

- This consists of horizontal reduction, vertical reduction and modulus reduction.



Algorithm 2: Reduction from $\mathbb{Z}_{p^r}$ to $\mathbb{Z}_p$

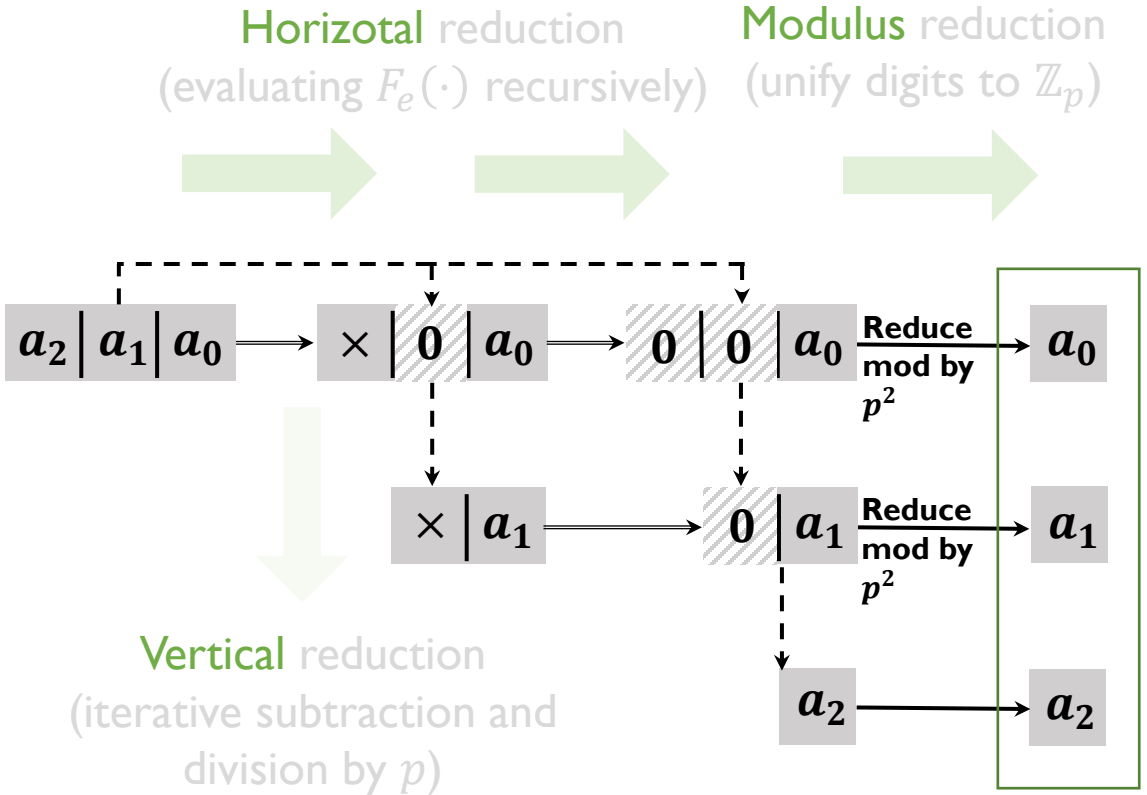
Input : The integer message $z \pmod{p^r}$.
Output: All digits of z, $\{z_i\}_{i=0}^{r-1} \pmod{p}$ such that $z = \sum_{i=0}^{r-1} z_i p^i$.

$w_{0,0} \leftarrow z$;
for $i \leftarrow 0$ **to** $r-2$ **do**
  for $j \leftarrow i$ **to** $r-2$ **do**
    $w_{i,j+1} \leftarrow F_e(w_{i,j})$; // Horizontal reduction
  $y \leftarrow z$;
  for $j \leftarrow 0$ **to** i **do**
    $y \leftarrow \text{DivideByP}(y - w_{j,i+1})$; // Vertical reduction
  $w_{i+1,i+1} \leftarrow y$;
  $z_i \leftarrow w_{i,r-1}$;
  for $j \leftarrow i$ **to** $r-2$ **do**
    $z_i \leftarrow \text{DivideModByP}(z_i)$; // Modulus reduction
$z_{r-1} \leftarrow w_{r-1,r-1}$;
return $\{z_i\}_{i=0}^{r-1}$

HE-Bridge: Reduction

Based on the polynomial $F_e(\cdot)$, we can extract the degree-p digits from an integer $\boldsymbol{a} \in \mathbb{Z}_{p^r}$.

- This consists of horizontal reduction, vertical reduction and modulus reduction.



Digits in beFV
$a_i \in \mathbb{Z}_p$

Algorithm 2: Reduction from $\mathbb{Z}_{p^r}$ to $\mathbb{Z}_p$

Input : The integer message $z \pmod{p^r}$.
Output: All digits of z, $\{z_i\}_{i=0}^{r-1} \pmod{p}$ such that $z = \sum_{i=0}^{r-1} z_i p^i$.

$w_{0,0} \leftarrow z$;
for $i \leftarrow 0$ **to** $r-2$ **do**
  for $j \leftarrow i$ **to** $r-2$ **do**
    $w_{i,j+1} \leftarrow F_e(w_{i,j})$; // Horizontal reduction
  $y \leftarrow z$;
  for $j \leftarrow 0$ **to** i **do**
    $y \leftarrow \text{DivideByP}(y - w_{j,i+1})$; // Vertical reduction
  $w_{i+1,i+1} \leftarrow y$;
  $z_i \leftarrow w_{i,r-1}$;
  for $j \leftarrow i$ **to** $r-2$ **do**
    $z_i \leftarrow \text{DivideModByP}(z_i)$; // Modulus reduction
$z_{r-1} \leftarrow w_{r-1,r-1}$;
return $\{z_i\}_{i=0}^{r-1}$

HE-Bridge: Lifting

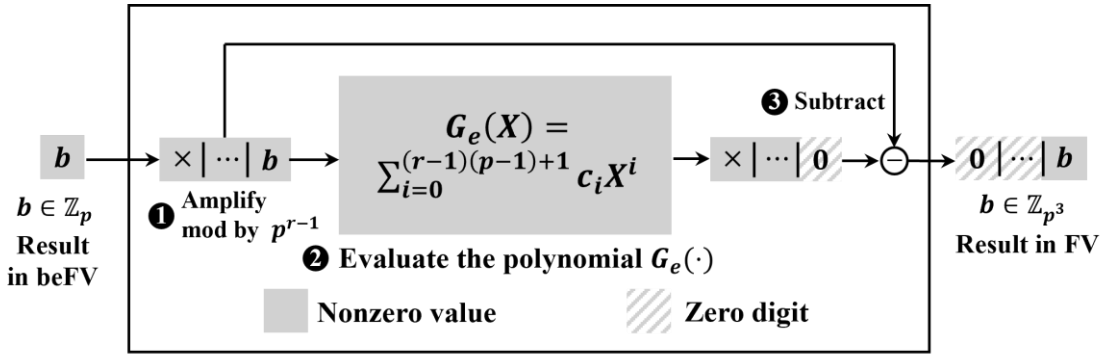
The comparison result is on the least significant digit.

- The result is correct modulo-p, but not modulo-p^r.

Lifting results from beFV to FV.

- The higher digits of the result are irrelevant values.
- Clearing the higher digits with a lifting polynomial.

- Approach1: evaluate $F_e(\cdot)$ for $r-1$ times. (The horizontal reduction in the first row)
- Approach2: evaluate another lifting polynomial $G_e(\cdot)$ only once.



Plaintext Space Lifting

HE-Bridge: Complexity Analysis

The complexity of each part mainly determines the degree of polynomials.

- Suppose the plaintext modulus in FV is p^r

Operation	Polynomial	Multiplications	Depth
Reduction	$F_e(\cdot)$ of degree p	$\frac{1}{2}r^2\sqrt{2p}$	$r\log_2 p$
Comparison	Interpolation polynomial of degree p	$r\sqrt{2p}$	$\log_2 r + \log_2 p$
Lifting	$G_e(\cdot)$ of degree $(r-1)(p-1)+1$	$\sqrt{2(r-1)(p-1)}$	$\log_2 r + \log_2 p$

Remarks:

- Recent works [1] have proposed simpler polynomials other than $F_e(\cdot)$ and $G_e(\cdot)$. Incorporating these polynomials can also lower the complexity of the reduction and lifting function in HEBridge.

[1] Geelen, Robin, et al. "On polynomial functions modulo pe and faster bootstrapping for homomorphic encryption."

HE-Bridge: Experimental Setup

Experiment environments

- HEBridge is built upon HElib.
- All parameters are set to ensure at least 120-bit of security.
- Experiments are done on a server with AMD Ryzen Thread-ripper PRO 3955WX with 125GB RAM running at 2.2GHz.

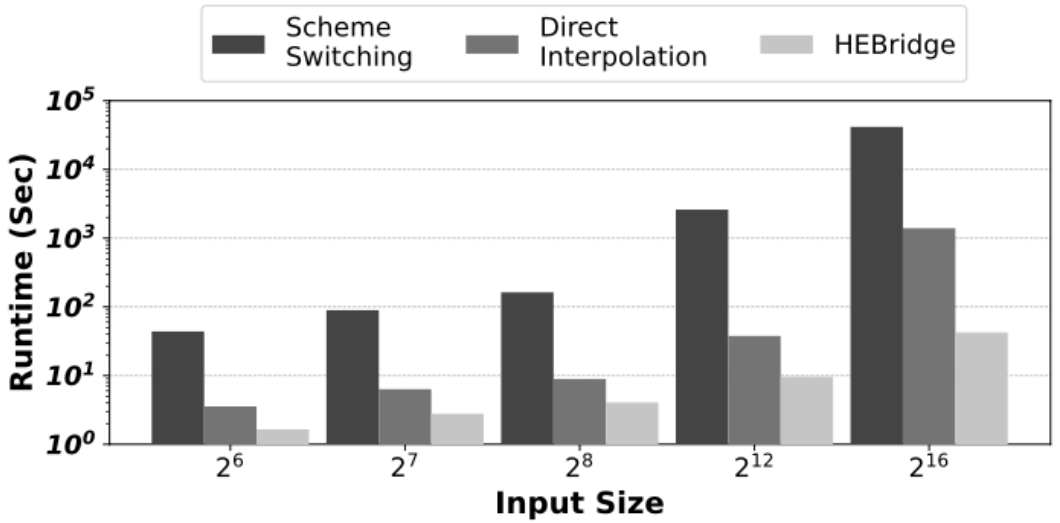
Benchmarks

- We mainly evaluate HEBridge in the context of private machine learning.
- We focus only a continuous evaluation of a convolution operation and a ReLU function.
- The convolution takes as input an image of size 28×28 and has 5 kernels of size 4×4.
- The ReLU function can be computed via the comparison function as $ReLU(x) = x \cdot CMP(x, 0)$.

HE-Bridge: Outperforming Prior Works

Comparison with prior works on Arith-ReLU function

- HEBridge is 1~3 orders of magnitudes faster than scheme switching
- Direct interpolation method takes more than 23.2 minutes for 16-bit inputs
- HEBridge only takes 42.3s for 16-bit inputs.



HE-Bridge: Runtime Breakdown

- HEBridge is overall lightweight.
- The runtime of reduction is comparable to the interpolation polynomials.
- The runtime of lifting is comparable to the arithmetic operation.

Table 3: The runtime breakdown of the Arith-ReLU function. Time is in seconds.

Bit-width	(p,r)	FV	HEBridge	beFV		HEBridge	Total
		Arithmetic	Reduction	Interpolation	Aggregation	Lifting	
8	(17,2)	1.11	1.01	0.91	0.12	0.92	4.06
12	(67,2)	1.41	3.02	3.46	0.18	1.58	9.65
16	(257,2)	4.55	23.34	27.58	0.81	4.99	61.26
20	(131,3)	10.99	52.07	29.07	1.64	13.53	107.3

HE-Bridge: Parameters and Memory

Ablation on encryption parameters

- HEBridge scales well with different input size and encryption parameters.
- For a fixed input bit-width, larger p leads to smaller r. This means fewer digits needs to be extracted.
- Both p and r need to be considered to minimize the runtime.

Memory cost

- HEBridge consumes as little as 0.68 GB memory for 8-bit inputs.
- HEBridge consumes 6.63 GB memory for 20-bit inputs.

Table 2: The runtime of Arith-ReLU function using different encryption parameters.

Bit-width	(p,r)	logq	(m,n)	Total Time (Sec)	Amortized Time (ms)
8	(5,4)	320	(16151, 15600)	6.56	12.62
	(7,3)	256	(13001, 13000)	4.21	32.4
	(17,2)	256	(13201, 12852)	4.06	6.63
12	(7,5)	488	(25301, 25300)	22.32	88.21
	(17,3)	411	(20567, 20196)	13.84	22.62
	(67,2)	413	(19801, 19800)	9.65	48.25
16	(17,4)	648	(31621, 31212)	42.97	70.21
	(41,3)	684	(34861, 34300)	50.13	102.31
	(257,2)	768	(37837, 37440)	61.26	127.63
20	(17,5)	995	(48199, 47736)	148.51	121.33
	(31,4)	879	(42407, 42406)	94.39	28.93
	(131,3)	864	(42799,42336)	107.3	212.89

Conclusion

- We present an in-depth analysis of the plaintext spaces in the FV-style HE schemes and their relationship with arithmetic and logic operations.
- We propose HEBridge to connect arithmetic and logic operations in FV-style HE schemes. To the best of our knowledge, we are the first to explore universal HE within the FV for large inputs
- Experiments show that HEBridge is 32.9 times faster than the direct interpolation methods and 1 to 3 orders magnitude faster than scheme switching methods.

Future work:

- More efficient reduction and lifting
- Support more complex functions and more applications

HEBridge: Connecting Arithmetic and Logic Operations in FV-style HE Schemes

Yancheng Zhang[1], Xun Chen[2], **Qian Lou[1]**

[1]University of Central Florida

[2]Samsung Research America

Code: https://github.com/UCF-Lou-Lab-PET/HE-Bridge