Towards Practical Transciphering for FHE with Setup Independent of the Plaintext Space

Speaker: **Jeongeun Park** (COSIC, KU Leuven, Belgium)

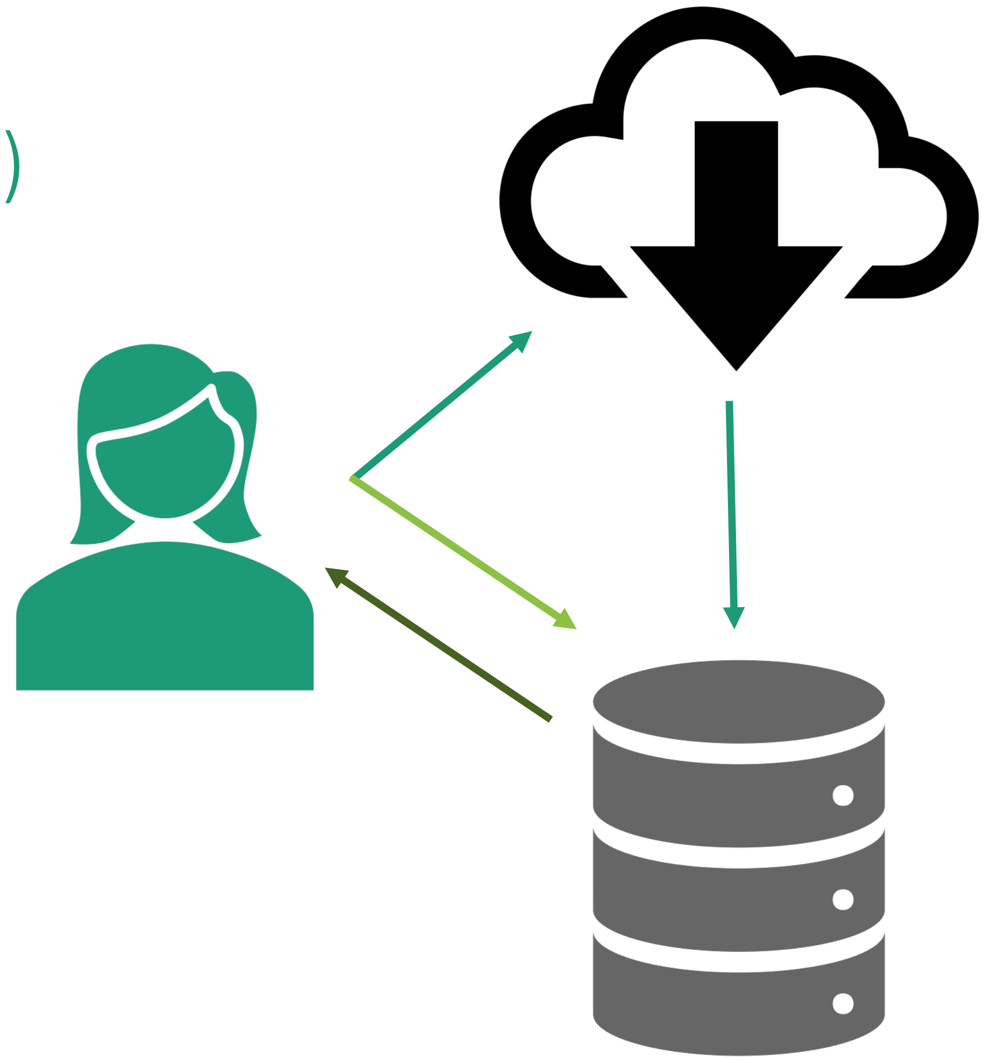
Joint work with **Pierrick Méaux** (University of Luxembourg, Luxembourg)
and **Hilder V. L. Pereira** (Universidade Estadual de Campinas, Brazil)

Table of Contents

(Fully) Homomorphic Encryption (FHE)

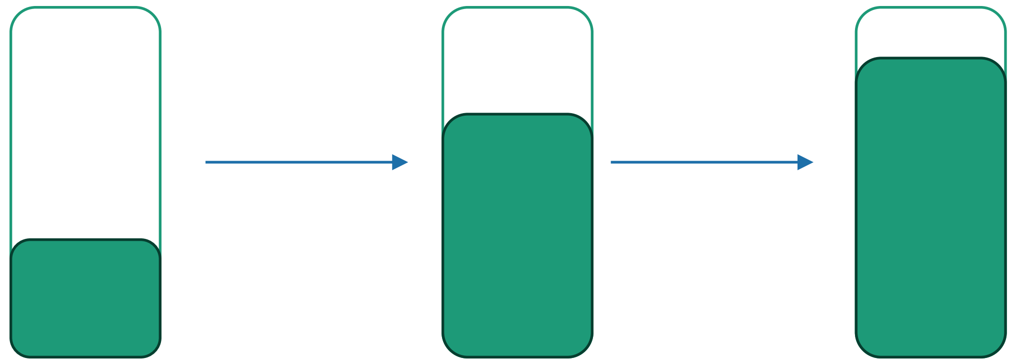
- Allows computations over encrypted data:
  - No need to (fully) trust the computing party when private/confidential data is handled.

- Application: privacy preserving outsourced computation such as secure machine learning (cloud service model)



Fully Homomorpic Encryption

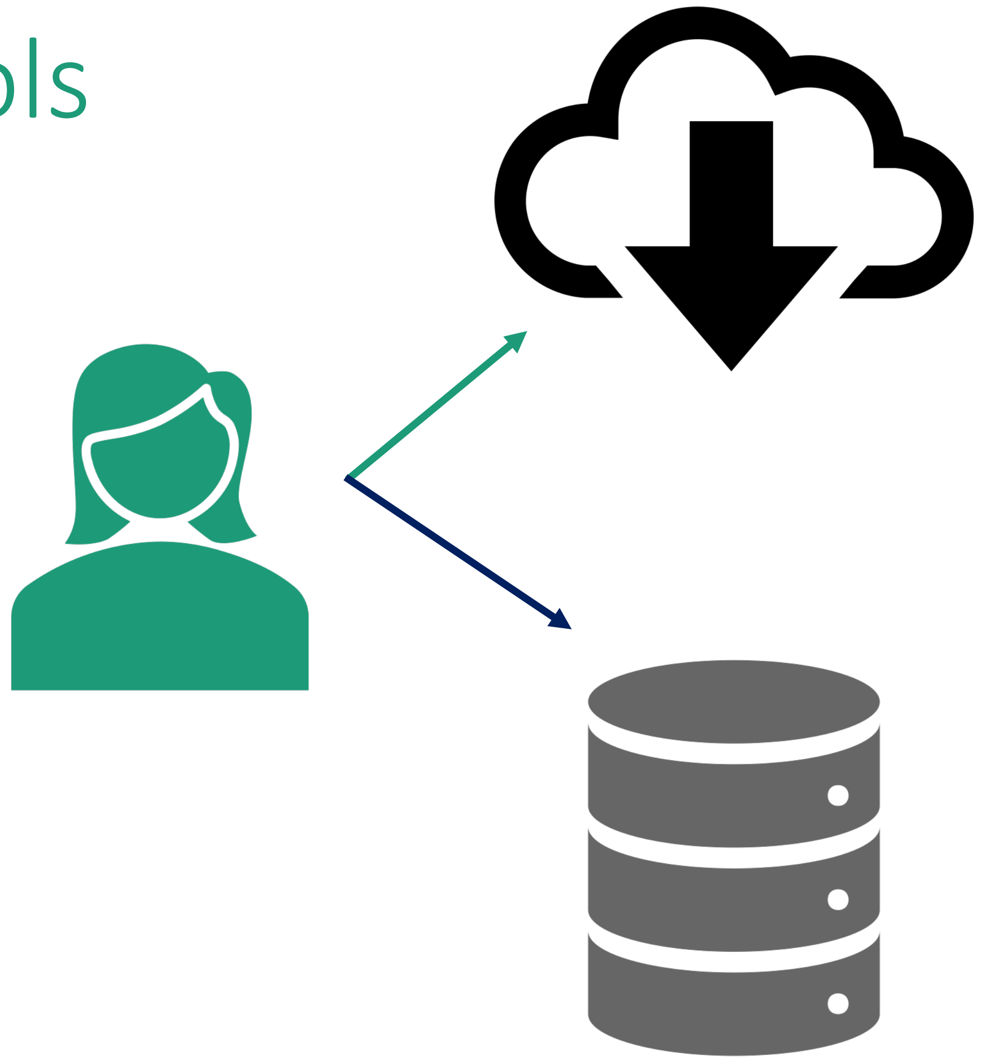
- Ciphertexts contain noise to guarantee the security.
- The noise grows over operations, resulting in decryption failure if it is too high.



- To handle the reasonable number of operations, the initial size of ciphertext is set much bigger than the size of plaintext.
- Expansion factor F = $\frac{|Ciphertext|}{|Plaintext|} \geq 10$ in many cases..

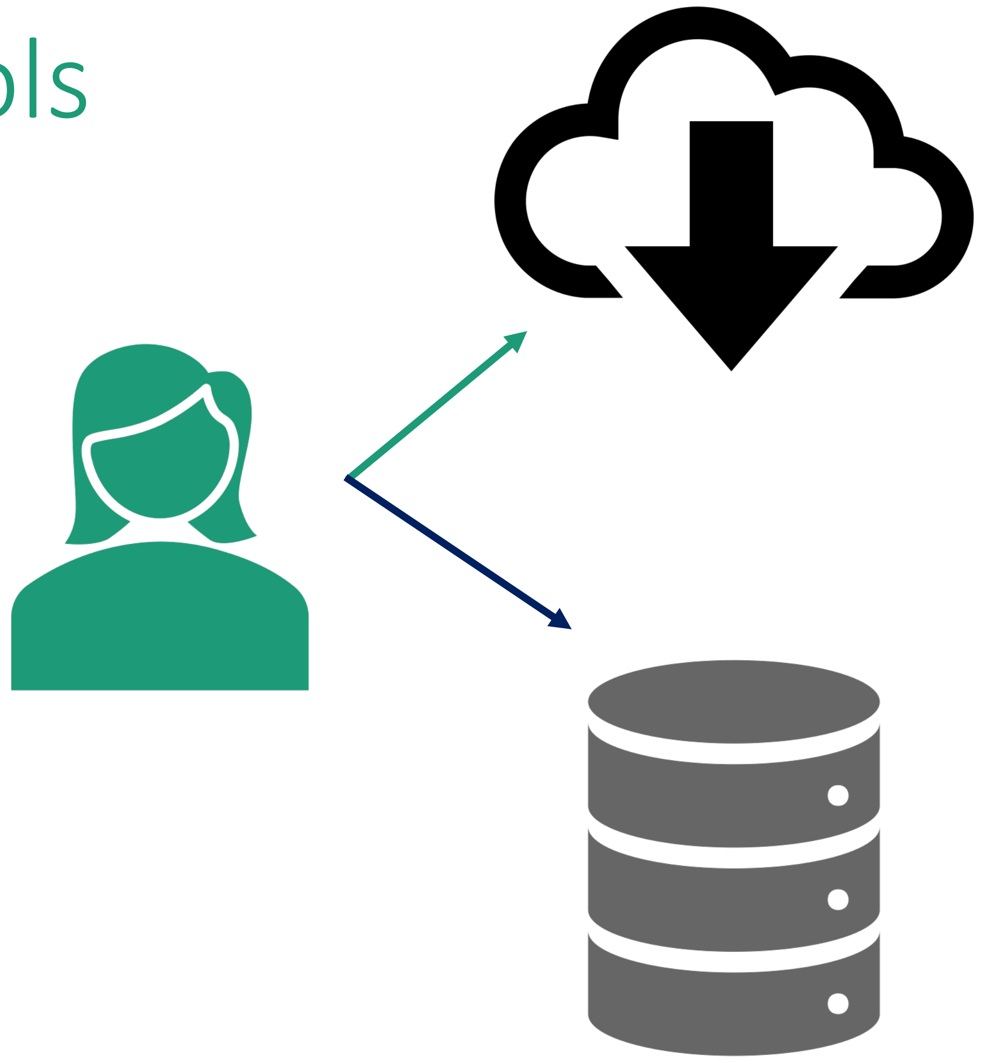
Privacy preserving protocols

- Back to our applications based on FHE:
  - Client sends "encrypted query" to a data holder or a computing party (server).
  - Server runs the requested protocol to give the correct answer to the client.
  - It is well known that optimal communication complexity is achievable with FHE.
  - What about actual communication cost?



Privacy preserving protocols

- Back to our applications based on FHE:
  - Client sends “encrypted query” to a data holder or a computing party (server).
  - Ciphertext size on the client side is too large compared to its underlying message.
  - In the worst case, 2.5KB is uploaded to send one single bit.
  - Too slow for upload!

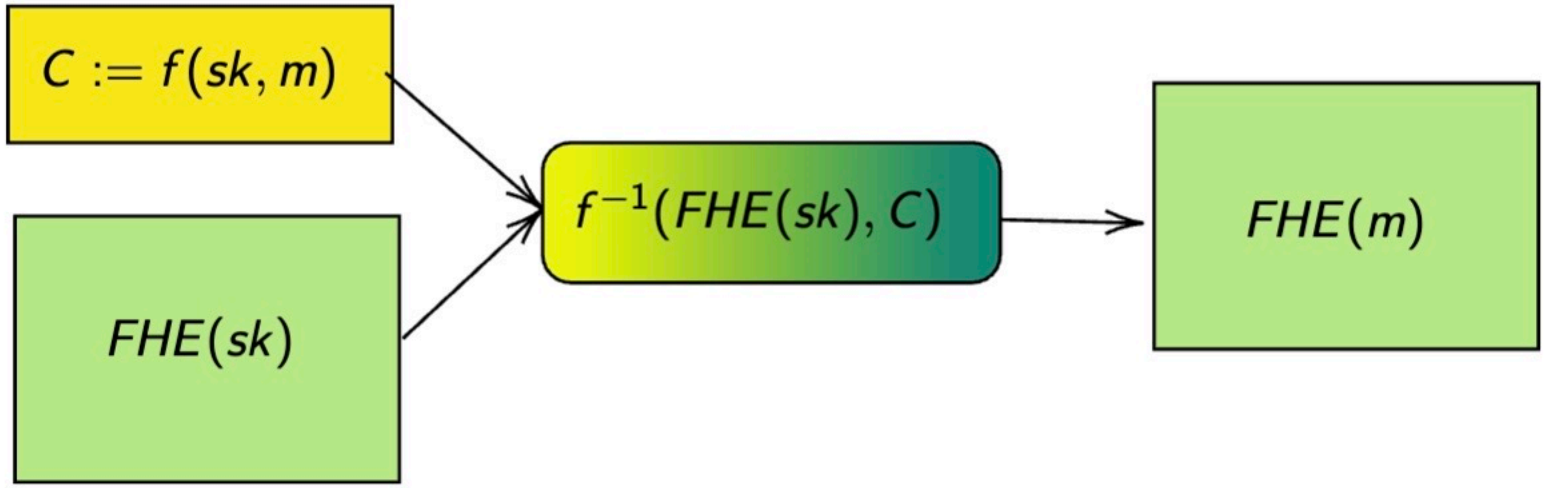


Transciphering
(a.k.a Hybrid Homomorphic Encryption)

- We know that the expansion factor of block/stream cipher is close to 1.
- Why don't we use block/stream cipher for upload,
  then let server transform the query into FHE ciphertext
  by running its decryption homomorphically? [NLV11]

- Then we could achieve (almost) optimal communication cost!

Transciphering



f = encryption function of a block/stream cipher

Existing works

- Homomorphically Evaluating standard ciphers with algorithmic optimization
  - Homomorphic AES evaluations [GHS12, TCB+23,WWL+23]
    - Running time improvement : from 4.1 minutes to 30 seconds over 10 years.
  - Homomorpihic Trivium evaluation [BOS23]

- Desigining FHE friendly ciphers for efficient transciphering
  - Block ciphers: LowMC, PRINCE, CHAGHRI, etc.
  - Stream ciphers: FiLIP, PASTA, Elisabeth, Kreyvium, etc.
  - And optimizing their homomorphic evaluations…[CDP+22,BOS23]

[GHS12] Homomorphic Evaluation of the AES Circuit
[TCB+23] At Last! A Homomorphic AES Evaluation in Less than 30 Seconds by Means of TFHE
[WWL+23] Fregata: Faster Homomorphic Evaluation of AES via TFHE
[CDP+22] SortingHat: Efficient Private Decision Tree Evaluation via Homomorphic Encryption and Transciphering
[BOS23] Trivial Transciphering With Trivium and TFHE

Check list to build efficient transciphering

Note that transciphering is a preprocessing for FHE based applications.

1. Noise after transciphering
    - It is directly related to what kind of operations we can do after transciphering.
2. Computation overhead
    - If it takes too long time compared to the main protocol, it is not useful.

How efficient are current designs?

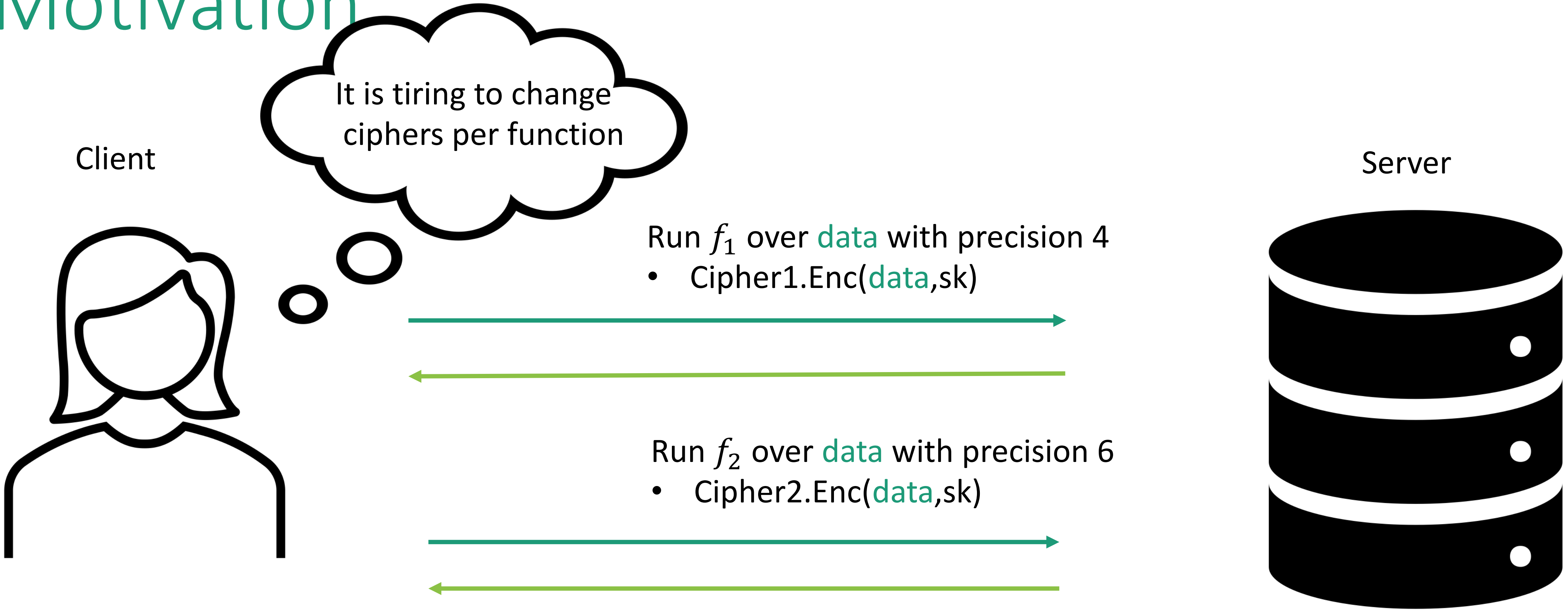
Cipher	Throughput (bit/s)	Transciphering (ms)	Multithreading	Noise budget (bits)
Kreyvium (64 bits) [BOS23]	427	291	Yes	N/A (Bootstrapping)
Trivium (64 bits) [BOS23]	529	259	Yes	N/A (Bootstrapping)
Elisabeth-4 (4 bits) [CHM+22]	44	91	Yes	N/A (Bootstrapping)
FiLIP (1 bit) [CDP+22]	382	2.62	No	6 $(\log q \approx 16.5)$

This table is not about comparing with one another in one metric but about showing how efficient state-of-art designs are.

Motivation

- What is FHE friendly message space?
  - $\mathbb{Z}_p^n$ or $R_p := \frac{\mathbb{Z}_p[X]}{\Phi_m(X)} = \frac{\mathbb{Z}_p[X]}{X^N+1}$, $m = 2N, N$= a power of 2
    - $p(\geq 2)$ any positive integer for TFHE, FINAL
    - $p(\geq 2)$ any positive integer such that $p \equiv 1 \; mod \; m$ for BGV/FV.
    - F_p is not an ideal choice for existing FHE.
      - $p = 2^{63}$ and $m = 48133$ (Chaghri)
      - p such that $p - 1 \nmid 3$ (PASTA)
      - Fixed p=2 for most of ciphers (FiLIP,LowMC, Kreyvium, etc.)
  - Elisabeth cipher is designed for $\mathbb{Z}_{2^4}$ (TFHE functional bootstrapping).

Motivation



Run f_1 over data with precision 4
- Cipher1.Enc(data,sk)

Run f_2 over data with precision 6
- Cipher2.Enc(data,sk)

For a fixed database encrypted under client's key

Motivation

- With these approaches,
  - Client has to send key materials for all ciphers per precision. (huge memory blowup on server side)
  - Client has to run several different setups. (not user-friendly)
  - Client has to rely on the security of all used ciphers. (not ideal from the security point of view)

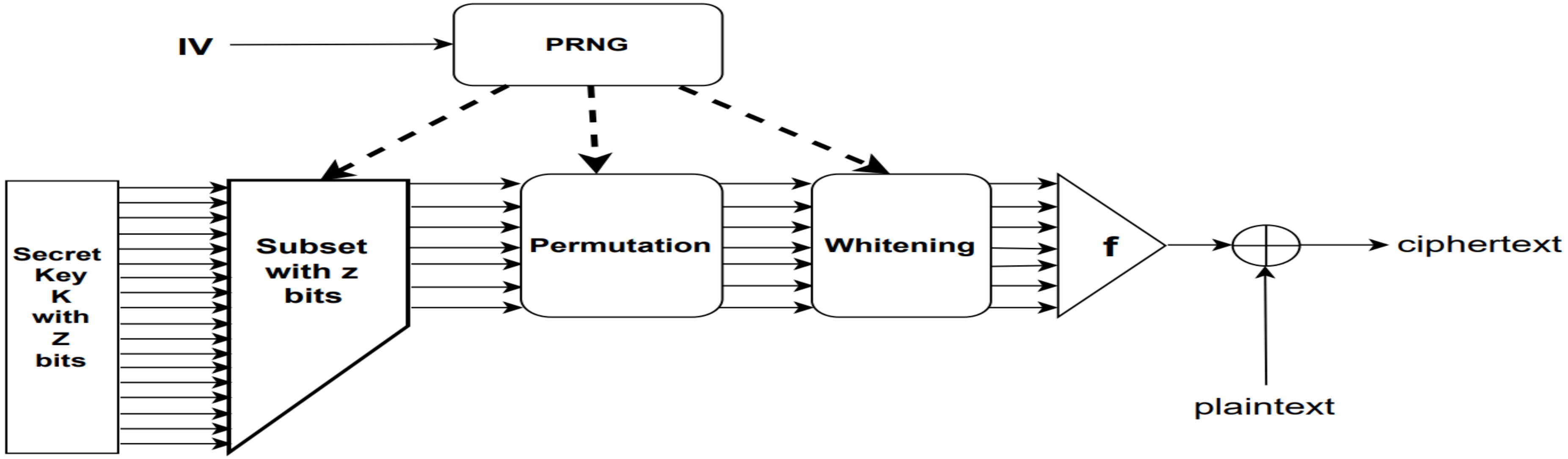
Our design

- We design an efficient transciphering technique which does not require a fixed message space of FHE schemes.
- Our solution is to "compose" bits into an integer homomorphically.
  - Client encrypts each bit of its input data (e.g. 32/64bits), using an FHE friendly cipher for $\mathbb{Z}_2$ and sends them to the server.
  - The server
    1. homomorphically transforms them into FHE ciphertexts encrypting bits.
    2. grabs the upper logp bits from the encrypted bits, transforms them into an logp-bit integer.
    3. uses the output for a target application which requires logp bits precision for input.
  - Server repeats the above per application, without interacting with the client.

Our instantiation: FiLIP cipher ($\mathbb{Z}_2$)

- We chose FiLIP cipher for encrypting data elements bit-by-bit since its homomorphic decryption is already optimized in [CDP+22].
- We modify their algorithms for further optimization of our design.
- $\mu \coloneqq \sum_{j=0}^{t-1} b_j \cdot 2^j$
- The goal is to generate FHE.Enc($\bar{\mu} \coloneqq \mu - (\mu \; mod \; 2^{t-logp})$)
- Instead of generating FHE.Enc(b_j) for the desired bits, we generate FHE.Enc($b_j \cdot 2^j$) by modifying homomorphic FiLIP decryption algorithm.
    - Parallel computation with logp threads.
- Adding up all logp ciphertexts returns FHE.Enc($\bar{\mu}$).

Technical detail



Definition 2 (XOR-THR Function (Definition 11 of [HMR20])). *For any positive integers k, d, and s such that $d \leq s+1$, and for all $z = (x_1, \ldots, x_k, y_1, \ldots, y_s) \in \mathbb{F}_2^{k+s}$, $\mathsf{XTHR}_{k,d,s}$ is defined as:*

$$\mathsf{XTHR}_{k,d,s}(z) = \mathsf{XOR}_k(x_1, \ldots, x_k) + \mathsf{T}_{d,s}(y_1, \ldots, y_s) \in \mathbb{F}_2,$$

where $\mathsf{XOR}_k(x_1, \ldots, x_k) = x_1 + \cdots + x_k \in \mathbb{F}_2$.

Technical detail

- To do:
  1. homomorphically evaluate two Boolean functions; XOR function and Threshold function over $\mathbb{Z}_p$ (we need FHE.Enc$(b_j \cdot 2^j)$),
  2. and combine them efficiently.
- Note that the previous work (SortingHat) dealt with bits only, therefore XOR was computed via homomorphic addition over $\mathbb{Z}_2$.
- Detail of the functions:
  1. Homomorphic XOR over $\mathbb{Z}_p$: $C_0 + C_1 - 2C_0 \cdot C_1$
     - We used external product for the multiplication.
  2. Homomorphic Threshold function over $\mathbb{Z}_p$: $T_d(a) = \begin{cases} 1, & if\ W_H(a) \geq d \\ 0, & otherwise \end{cases}$
     - where $W_H(a)$ the Hamming weight of a binary vector a.

Definition 2 (XOR-THR Function (Definition 11 of [HMR20])). *For any positive integers k, d, and s such that $d \leq s+1$, and for all $z = (x_1, \ldots, x_k, y_1, \ldots, y_s) \in \mathbb{F}_2^{k+s}$, $\mathsf{XTHR}_{k,d,s}$ is defined as:*

$$\mathsf{XTHR}_{k,d,s}(z) = \mathsf{XOR}_k(x_1, \ldots, x_k) + \mathsf{T}_{d,s}(y_1, \ldots, y_s) \in \mathbb{F}_2,$$

where $\mathsf{XOR}_k(x_1, \ldots, x_k) = x_1 + \cdots + x_k \in \mathbb{F}_2$.

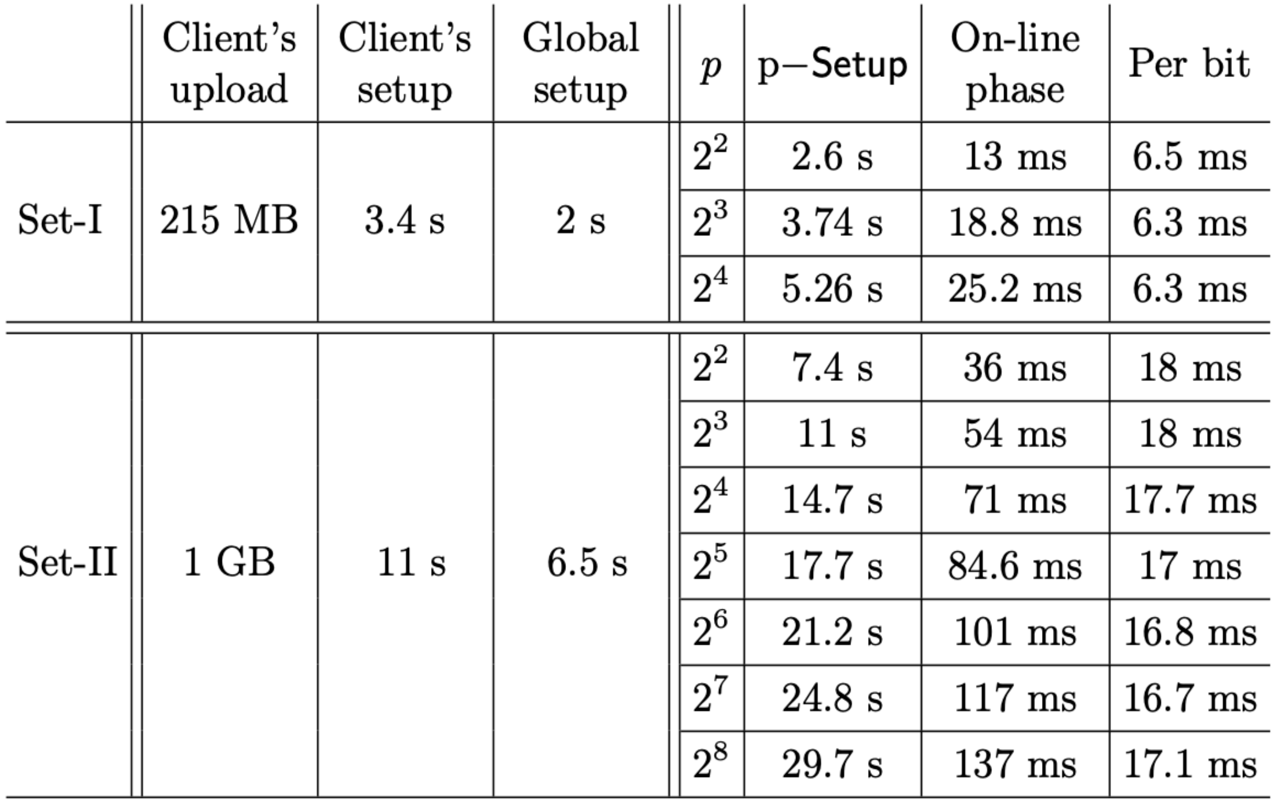
- Solution:
  - We use so-called test-polynomial $T(X)$ to evaluate $\mathrm{FHE.Enc}(f(m))$ from $T(X) \cdot FHE.Enc(X^m)$, where $T(X)$ depends on f .
  - We evaluate $T(X) \cdot FHE.Enc(XOR(x_1, x_2, \ldots, x_k))$ by applying homomorphic XOR over $\mathbb{Z}_p$ k times.
    - We obtain $\mathrm{C} := FHE.Enc(T(X) \cdot XOR(x_1, x_2, \ldots, x_k))$
    - and we can lift the result to the exponent ; $\mathrm{C}' := FHE.Enc(T(X) \cdot X^{XOR(x_1,\ldots,x_k)})$ by computing $(X-1) \cdot C + T(X)$.
  - Then we multiply C' by $FHE.Enc(X^{2 \cdot y_i})$ for each y_i.
  - Finally we obtain $FHE.Enc(T(X) \cdot X^{XOR(x_1,\ldots,x_k)+2 \cdot W_H(y_1,\cdots,y_s)})$.
  - Due to the way $T(X)$ is defined, we get $FHE.Enc(2^j \cdot b_j)$.

Implementation result

- We used FINAL library for our proof-of-concept implementation to use their practical implementation of external product.
  - External product: efficiently designed for a product of an integer and a bit.
  - One can use other libraries (e.g. SEAL, HElib, TFHE-rs, …) to implement our design with larger parameters.

Implementation result

Table 3: *Running times and upload depending on different parameter sets*



	Client's upload	Client's setup	Global setup	p	p−Setup	On-line phase	Per bit
Set-I	215 MB	3.4 s	2 s	2^2	2.6 s	13 ms	6.5 ms
				2^3	3.74 s	18.8 ms	6.3 ms
				2^4	5.26 s	25.2 ms	6.3 ms
Set-II	1 GB	11 s	6.5 s	2^2	7.4 s	36 ms	18 ms
				2^3	11 s	54 ms	18 ms
				2^4	14.7 s	71 ms	17.7 ms
				2^5	17.7 s	84.6 ms	17 ms
				2^6	21.2 s	101 ms	16.8 ms
				2^7	24.8 s	117 ms	16.7 ms
				2^8	29.7 s	137 ms	17.1 ms

NTRU params	N	Q	q
Set-I	2^{10}	$912829 \approx 2^{19.8}$	$92683 \approx 2^{16.5}$
Set-II	2^{11}	$1073741827 \approx 2^{30}$	$9209716 \approx 2^{21}$

Noise after transciphering

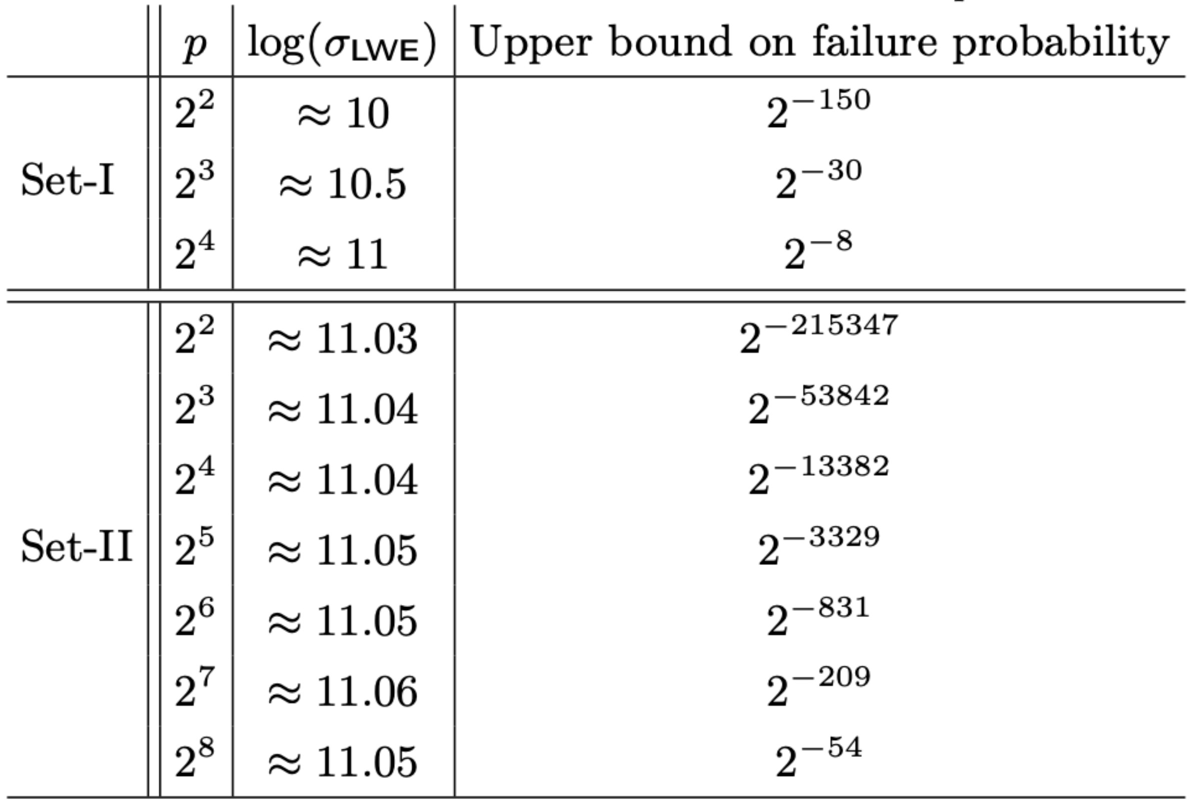


Table 4: *Failure probability of output of* $\mathbb{Z}_p$Transcipher.

	p	$\log(\sigma_{\mathsf{LWE}})$	Upper bound on failure probability
Set-I	2^2	≈ 10	2^{-150}
	2^3	≈ 10.5	2^{-30}
	2^4	≈ 11	2^{-8}
Set-II	2^2	≈ 11.03	$2^{-215347}$
	2^3	≈ 11.04	2^{-53842}
	2^4	≈ 11.04	2^{-13382}
	2^5	≈ 11.05	2^{-3329}
	2^6	≈ 11.05	2^{-831}
	2^7	≈ 11.06	2^{-209}
	2^8	≈ 11.05	2^{-54}

Comparison with other FiLIP evaluation

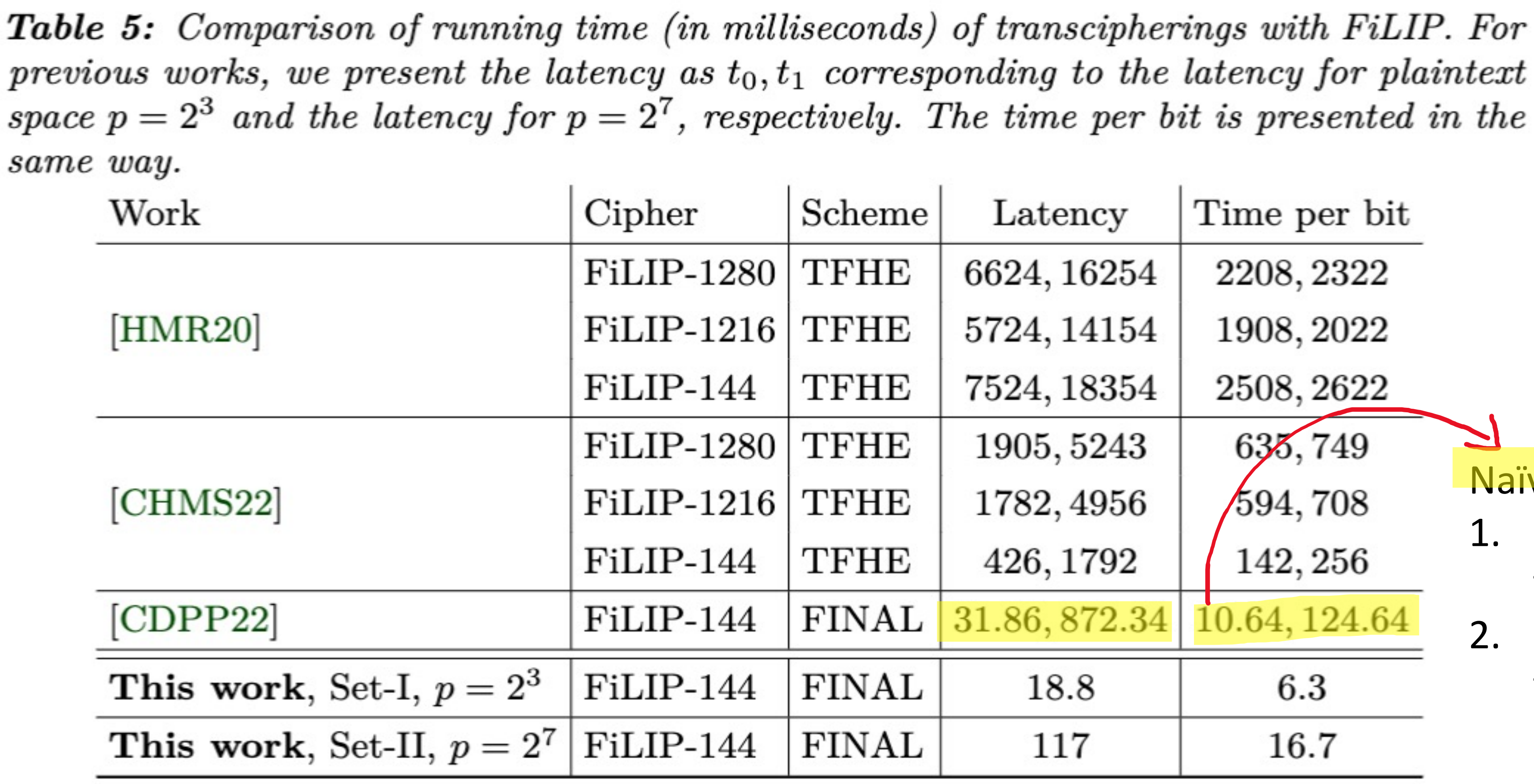


Table 5: *Comparison of running time (in milliseconds) of transcipherings with FiLIP. For previous works, we present the latency as t_0, t_1 corresponding to the latency for plaintext space $p = 2^3$ and the latency for $p = 2^7$, respectively. The time per bit is presented in the same way.*

Work	Cipher	Scheme	Latency	Time per bit
[HMR20]	FiLIP-1280	TFHE	$6624, 16254$	$2208, 2322$
	FiLIP-1216	TFHE	$5724, 14154$	$1908, 2022$
	FiLIP-144	TFHE	$7524, 18354$	$2508, 2622$
[CHMS22]	FiLIP-1280	TFHE	$1905, 5243$	$635, 749$
	FiLIP-1216	TFHE	$1782, 4956$	$594, 708$
	FiLIP-144	TFHE	$426, 1792$	$142, 256$
[CDPP22]	FiLIP-144	FINAL	$31.86, 872.34$	$10.64, 124.64$
This work, Set-I, $p = 2^3$	FiLIP-144	FINAL	18.8	6.3
This work, Set-II, $p = 2^7$	FiLIP-144	FINAL	117	16.7

Naïve approach:

1. Run binary transciphering $\log p$ times
2. run bootstrapping to change the message space from modulo 2 to modulo p.

[HMR20] Transciphering, using FiLIP and TFHE for an efficient delegation of computation.
[CHMS22] Towards case-optimized hybrid homomorphic encryption - featuring the elisabeth stream cipher.
[CDPP22] Sortinghat: Efficient private decision tree evaluation via homomorphic encryption and transciphering.

Comparison with Elisabeth-4

Evaluation	Mode	Latency (ms)	Time per bit (ms)
Elisabeth-4	2 KS, multithreaded	91.143	22.786
Elisabeth-4	Single KS, multithreaded	103.810	25.953
Elisabeth-4	2 KS, monothreaded	1485.0	371.25
Elisabeth-4	Single KS, monothreaded	1648.6	412.15
Ours	Set-I, monothreaded	25.2	6.3
Ours	Set-II, monothreaded	71	17.7

- Elisabeth-4 is designed for 4-bits of integers.
- Multithreaded versions of Elisabeth-4 were executed on 12, or 48, or 64 threads.

Summary and conclusion

- We designed a new transciphering method which can be used for any message precision, for the first time.
- Client does not need to specify the message precision in advance.
- The server can reuse the given data for several application algorithms
  by taking only necessary upper bits of data,
  depending on the target application,
  without running different setups with the client.
- Our method is not a scheme specific design.
  - Therefore, one can use any FHE schemes to realize it.

Thank you for your attention!

https://eprint.iacr.org/2023/1531.pdf