

Disclaimer: A small number of fixes  
have been made compared to  
the slides used in the presentation.

# THRESHOLD-FHE FROM CKKS

DAMIEN STEHLÉ

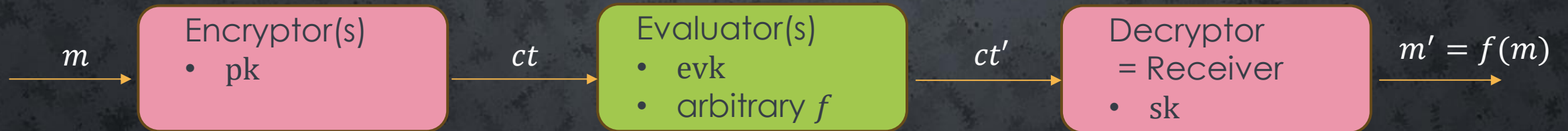
[DAMIEN.STEHLE@CRYPTOLAB.CO.KR](mailto:DAMIEN.STEHLE@CRYPTOLAB.CO.KR)

( TALK BASED ON [HHN+26,CKSS25,CPS26,MHP+25], AND ONGOING WORKS )

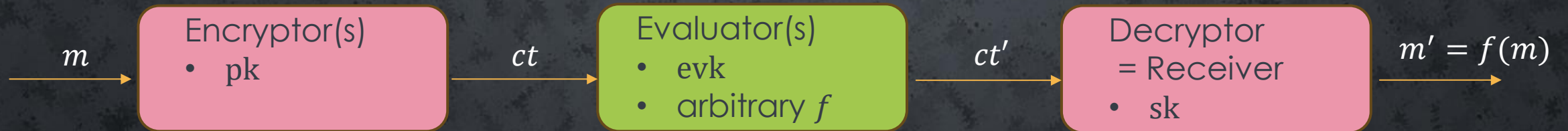
NIST MPTS26 - ONLINE - JANUARY 27, 2026



# FULLY HOMOMORPHIC ENCRYPTION (FHE)

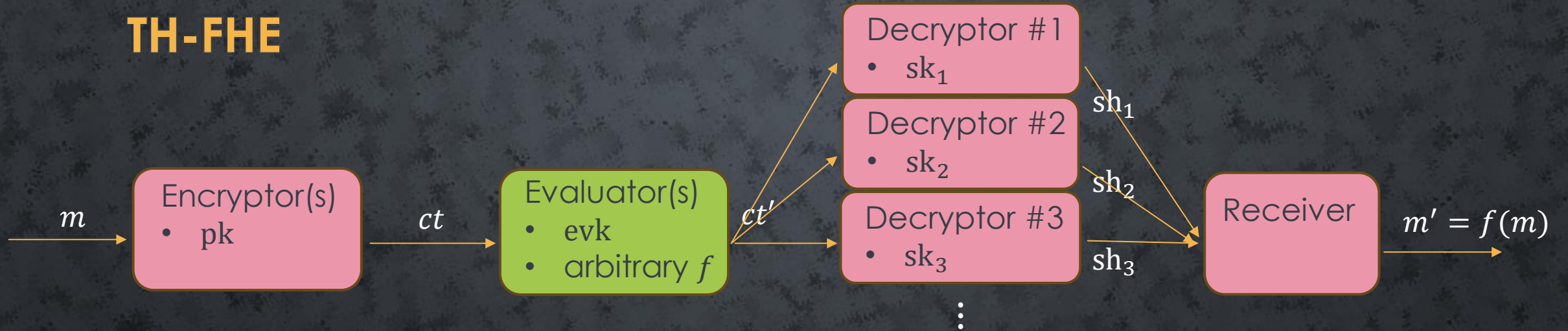


# FULLY HOMOMORPHIC ENCRYPTION (FHE)



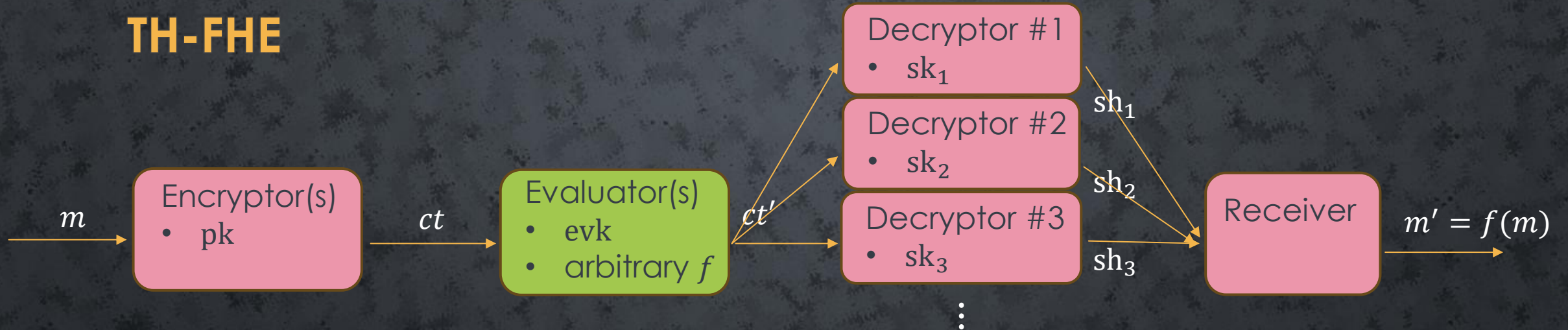
- ☺ Evaluator does not manipulate any sensitive data, as **everything is encrypted**
- ☹ **Decryptor is a single point of failure**

# TH-FHE



- ☺ Evaluator does not manipulate any sensitive data, as **everything is encrypted**
- ☺ **Decryptors secret-share the decryption key**

# TH-FHE

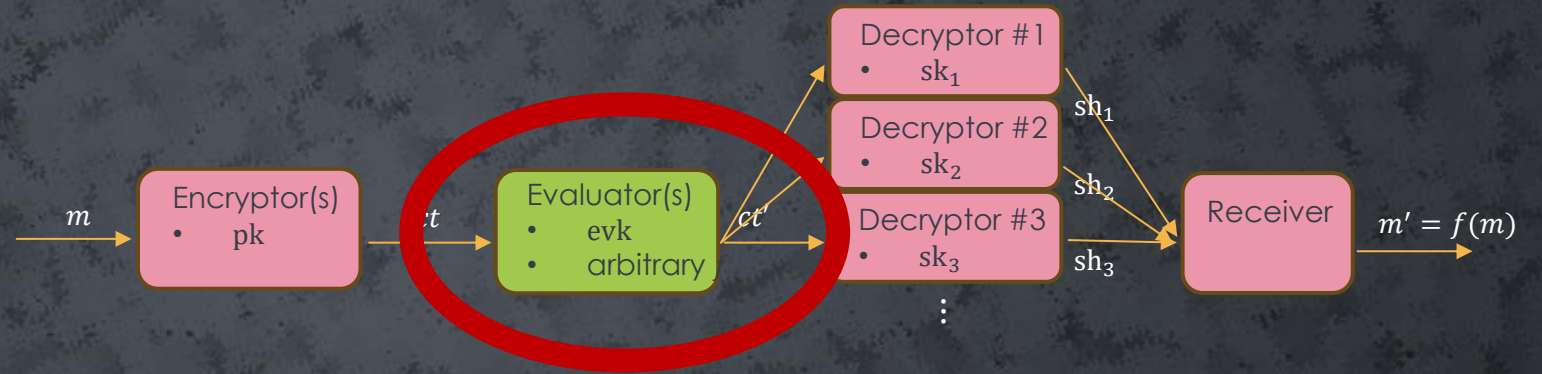


- ☺ Evaluator does not manipulate any sensitive data, as **everything is encrypted**
- ☺ **Decryptors secret-share the decryption key**

Two major applications:

- ❖ enables MPC with **minimalistic communications** [AJL+12]
- ❖ gives a threshold-**BLAH**, by FHE-evaluating **BLAH** [BGG+18]

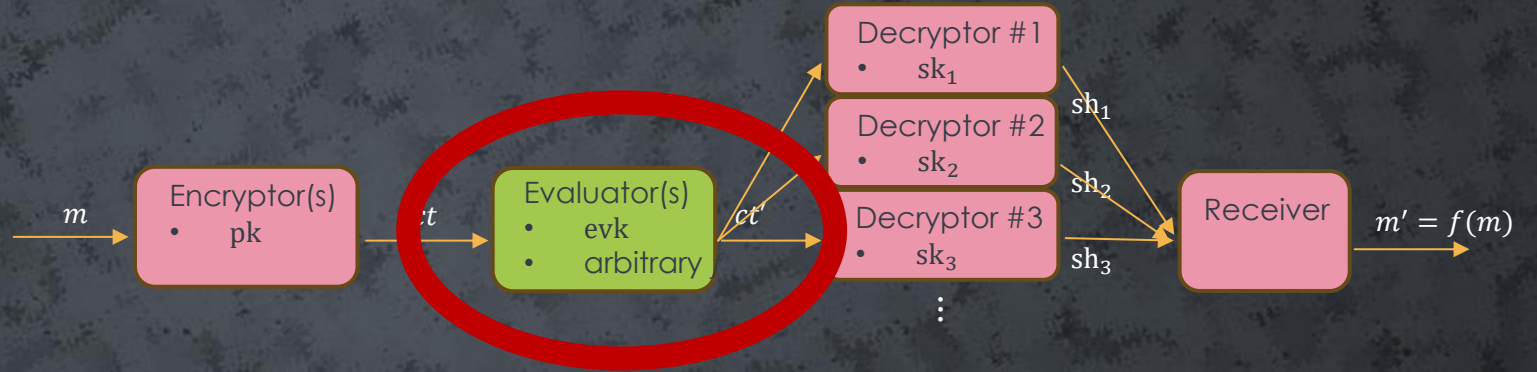
# TH-CKKS



Th-FHE is built from FHE semi-generically -- Th-FHE evaluation exactly matches FHE evaluation:

**Let's use CKKS!** [CKKS17]

# TH-CKKS



Th-FHE is built from FHE semi-generically -- Th-FHE evaluation exactly matches FHE evaluation:

## Let's use CKKS! [CKKS17]

**Plaintext space:**  $\mathbb{C}^{N/2}$  or  $\mathbb{R}^N$  (up to some precision)

- add in //
- multiply in //
- conjugate in //
- rotate the coordinates

CKKS is **level-based**

- mult consumes 1 level
- add, conj & rot consume 0 level
- bootstrapping (BTS) regains levels

# CKKS EFFICIENCY

- ❖ Every time we multiply, we lose a level
- ❖ Levels can be regained by bootstrapping (BTS)

**BTS** | RTX-5090 | HEAAN2

---

Number of reals	$N = 2^{16}$
Precision	19 bits
Non-BTS levels	15
Failure probability	$< 2^{-128}$
Security	128 bits

---

10.5ms

$\approx 94\text{M ct-ct mults/sec}$

(for 19-bit precision)

# EFFICIENCY ENABLES LARGE COMPUTATIONS: IRIS RECOGNITION

Encrypted DB of iris templates  
Encrypted iris template query



Does the query ``match'' one of the DB entries?

# EFFICIENCY ENABLES LARGE COMPUTATIONS: IRIS RECOGNITION

Encrypted DB of iris templates  
Encrypted iris template query } Does the query ``match'' one of the DB entries?

- ❖ A template is a  $\{0,1\}$ -vector of dimension  $\approx 2^{14}$
- ❖ Matrix-vector product & comparison to a cutoff
- ❖ Output: YES or NO

# EFFICIENCY ENABLES LARGE COMPUTATIONS: IRIS RECOGNITION

Encrypted DB of iris templates  
Encrypted iris template query } Does the query ``match'' one of the DB entries?

- ❖ A template is a  $\{0,1\}$ -vector of dimension  $\approx 2^{14}$
  - ❖ Matrix-vector product & comparison to a cutoff
  - ❖ Output: YES or NO
    - **FHE** enables encrypted processing
    - **Threshold** decryption enables to control what is decrypted
- => Th-FHE protects queries and DB

# EFFICIENCY ENABLES LARGE COMPUTATIONS: IRIS RECOGNITION

**IRIS recog.** | 8x RTX-5090 | HEAAN2

---

Template dimension	$2^{14}$
Number of query vectors	992
Number of DB entries	115K
Security	128 bits

---

1.8s

[HHN+26]

# EFFICIENCY ENABLES LARGE COMPUTATIONS: IRIS RECOGNITION

**IRIS recog.** | 8x RTX-5090 | HEAAN2

Template dimension	$2^{14}$
Number of query vectors	992
Number of DB entries	115K
Security	128 bits

1.8s

[HHN+26]

## Comparison to secret-sharing MPC

[BGK+24]

- Much fewer rounds: 3 v dozens
- Improved scaling with:
  - DB size (communication)
  - number of parties (comm. & comput.)
- The FHE computation is public
  - no need to protect it
  - can be distributed arbitrarily

# EXACT-CKKS

CKKS may be used for exact computations => thresholdization of cryptographic primitives

# EXACT-CKKS

CKKS may be used for exact computations => thresholdization of cryptographic primitives

**AES-128** | RTX-5090 | HEAAN2

---

Throughput: 8000 blocks / s

Latency: 1 block in less than 80ms

---

=> 1-round Threshold-AES

# EXACT-CKKS

CKKS may be used for exact computations => thresholdization of cryptographic primitives

**AES-128** | RTX-5090 | HEAAN2

---

Throughput: 8000 blocks / s

Latency: 1 block in less than 80ms

---

=> 1-round Threshold-AES

**ML-DSA-44** | RTX-5090 | HEAAN2

---

Pre-challenge: 0.9s

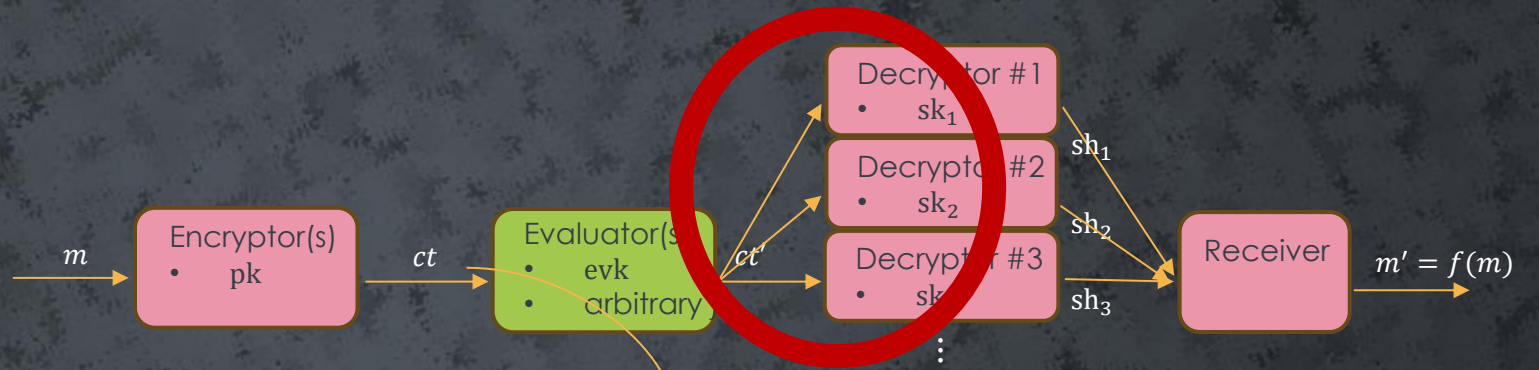
Post-challenge: 0.2s

Success probability: 0.86

---

=> 2-round Th-Sign that is  
interchangeable with ML-DSA

# DECRYPTION SECURITY

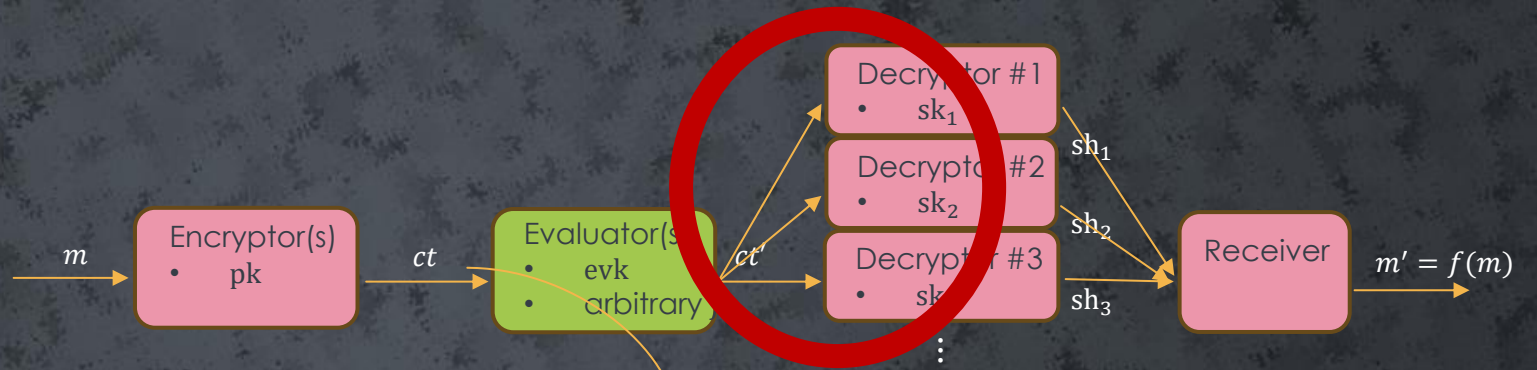


- To hide the computation noise and the partial key  $sk_i$ , the  $i$ -th decryptor should flood the noise

$$a \mapsto a \cdot sk_i + e_i$$

$$ct = (a, b) \text{ over } \mathbb{Z}_q[X]/(X^N + 1)$$

# DECRYPTION SECURITY



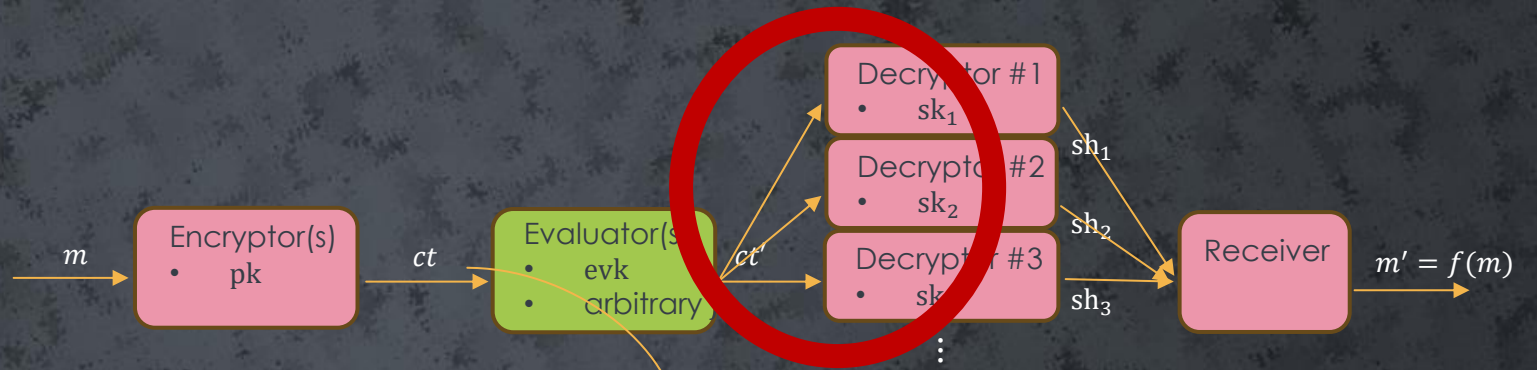
- To hide the computation noise and the partial key  $sk_i$ , the  $i$ -th decryptor should flood the noise

$$a \mapsto a \cdot sk_i + e_i$$

$$ct = (a, b) \text{ over } \mathbb{Z}_q[X]/(X^N + 1)$$

- The noise  $e_i$  can be  $2^{\lambda/2}$  or more, where  $\lambda$  is the security parameter.
- For correctness, the **plaintext** to be decrypted should be **very precise**.

# DECRYPTION SECURITY



- To hide the computation noise and the partial key  $sk_i$ , the  $i$ -th decryptor should flood the noise

$$a \mapsto a \cdot sk_i + e_i$$

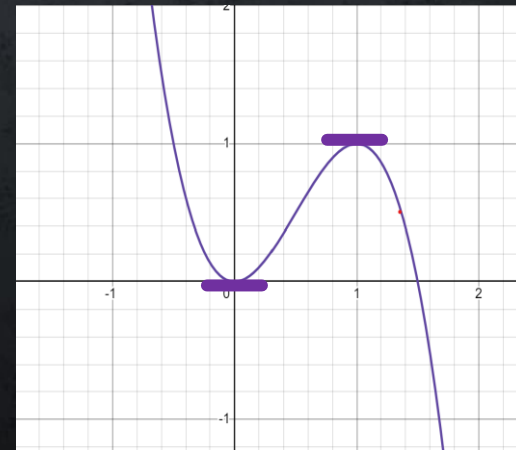
$$ct = (a, b) \text{ over } \mathbb{Z}_q[X]/(X^N + 1)$$

- The noise  $e_i$  can be  $2^{\lambda/2}$  or more, where  $\lambda$  is the security parameter.
- For correctness, the **plaintext** to be decrypted should be **very precise**.

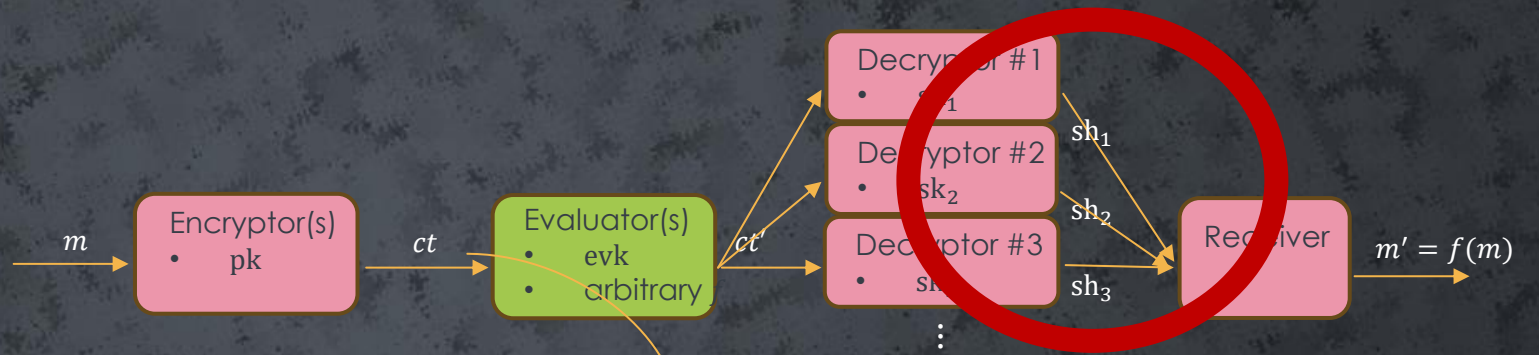
For binary output plaintexts

- ❖ Compute with low precision
- ❖ Increase precision before decryption, iteratively applying  $x \mapsto 3x^2 - 2x^3$

[CKSS25]



# THRESHOLD POLICY



Assume that  $sk = \sum_{j \in T} \lambda_j \cdot sk_j$ , where  $T$  is the set of decryptors ( $\#T = t$ )

$ct = (a, b)$  over  $\mathbb{Z}_q[X]/(X^N + 1)$

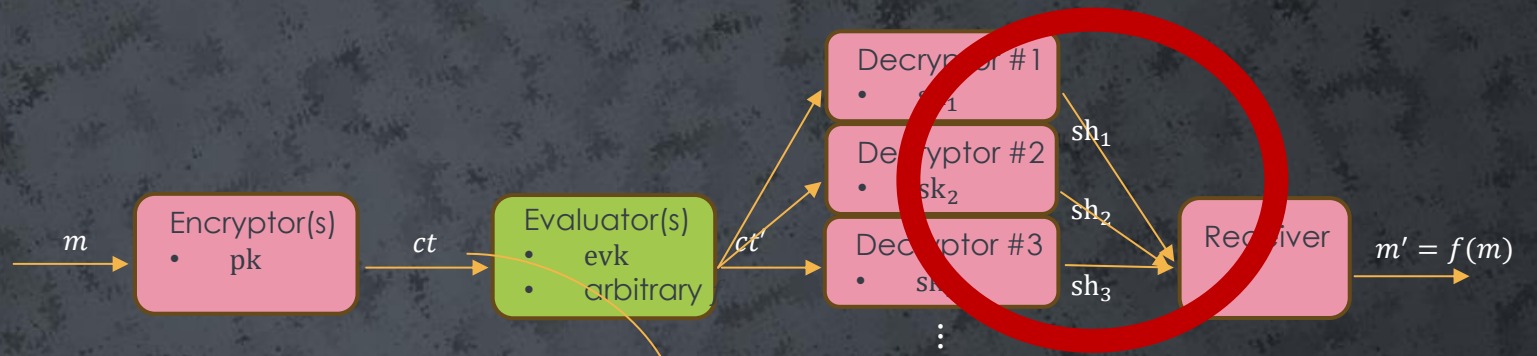
**Asynchronous decryption**

**Synchronous decryption**

The decryptors **do not know  $T$**

The decryptors **know  $T$**

# THRESHOLD POLICY



Assume that  $sk = \sum_{j \in T} \lambda_j \cdot sk_j$ , where  $T$  is the set of decryptors ( $\#T = t$ )

$ct = (a, b)$  over  $\mathbb{Z}_q[X]/(X^N + 1)$

## Asynchronous decryption

The decryptors **do not know  $T$**

$$sh_j = a \cdot sk_j + e_j$$

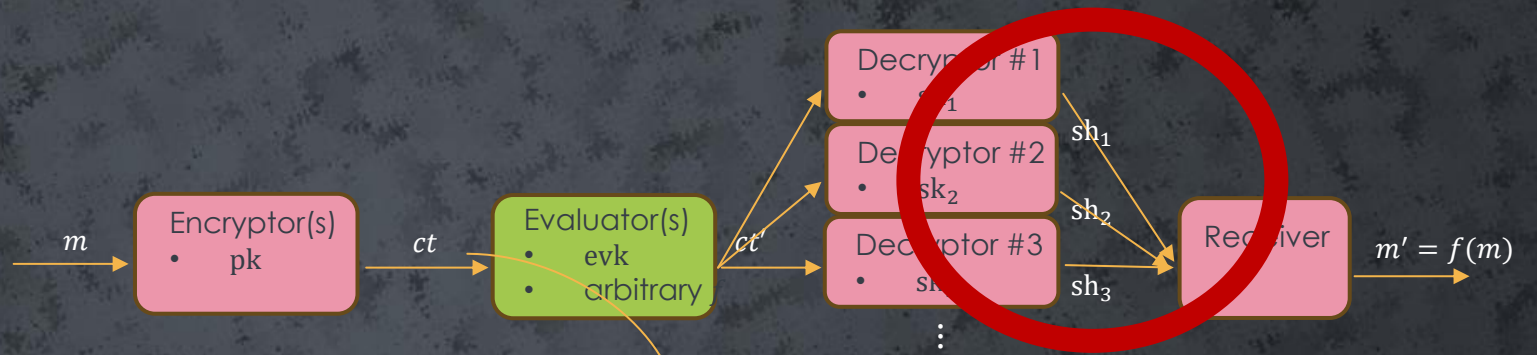
$$m = b + \sum_{j \in T} \lambda_j \cdot sh_j$$

The huge  $\lambda_j$ 's amplify the  $e_j$ 's  
 => Not practical for growing  $(t, n)$

## Synchronous decryption

The decryptors **know  $T$**

# THRESHOLD POLICY



Assume that  $sk = \sum_{j \in T} \lambda_j \cdot sk_j$ , where  $T$  is the set of decryptors ( $\#T = t$ )

$ct = (a, b)$  over  $\mathbb{Z}_q[X]/(X^N + 1)$

## Asynchronous decryption

The decryptors **do not know  $T$**

$$sh_j = a \cdot sk_j + e_j$$

$$m = b + \sum_{j \in T} \lambda_j \cdot sh_j$$

The huge  $\lambda_j$ 's amplify the  $e_j$ 's  
 $\Rightarrow$  Not practical for growing  $(t, n)$

## Synchronous decryption

The decryptors **know  $T$**

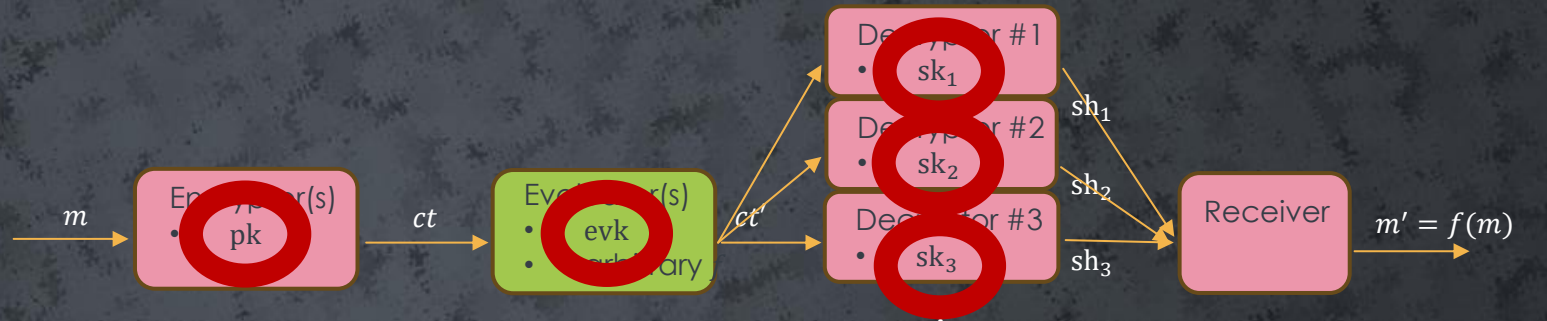
$$sh_j = \lambda_j \cdot a \cdot sk_j + e_j$$

$$m = b + \sum_{j \in T} sh_j$$

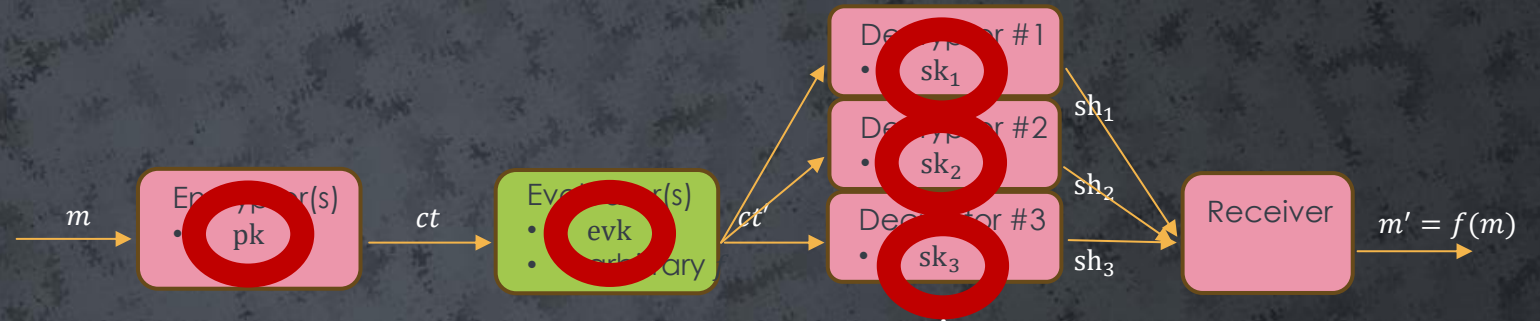
$\Rightarrow$  Can handle very large  $(t, n)$

Mmmh... as is, this is insecure, but can be fixed [CPS26]

# DISTRIBUTED KEY GENERATION

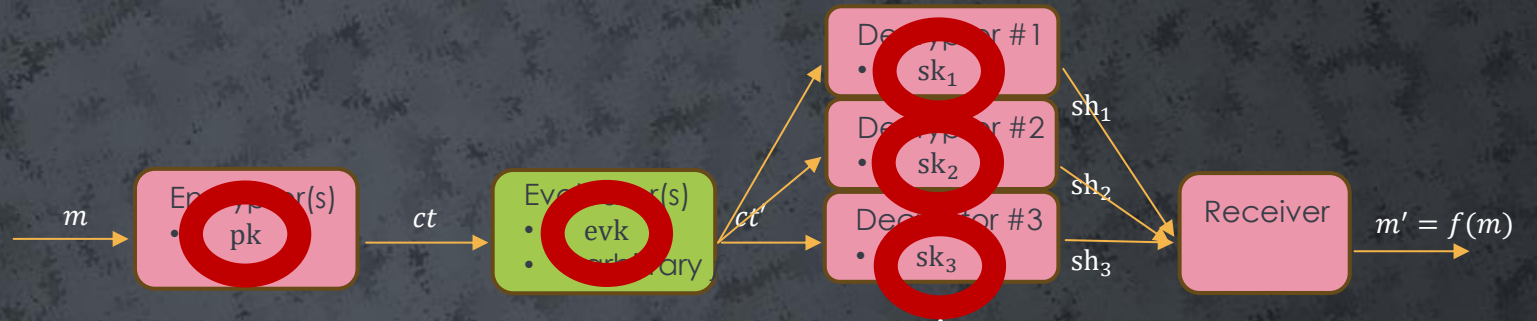


# DISTRIBUTED KEY GENERATION



- ❖ For efficient computations,  $sk$  consists of **sparse polynomials**
- ❖ Naïve DKG:
  - ❖ Decryptor # $i$  samples its  $sk_i$  and broadcasts  $a \cdot sk_i + e_i$
  - ❖  $pk = \sum_i (a \cdot sk_i + e_i)$  ... corresponds to an ill-distributed  $sk$
- ❖ Better solution (high level): bootstrap KeyGen!
  - ❖ Generate temporary bad keys
  - ❖ Homomorphically run KeyGen for sparse keys
  - ❖ Th-decrypt to get sparse-secret key material

# DISTRIBUTED KEY GENERATION



- ❖ For efficient computations,  $sk$  consists of **sparse polynomials**
- ❖ Naïve DKG:
  - ❖ Decryptor # $i$  samples its  $sk_i$  and broadcasts  $a \cdot sk_i + e_i$
  - ❖  $pk = \sum_i (a \cdot sk_i + e_i) \dots$  corresponds to an ill-distributed  $sk$
- ❖ Better solution (high level): bootstrap KeyGen!
  - ❖ Generate temporary bad keys
  - ❖ Homomorphically run KeyGen for sparse keys
  - ❖ Th-decrypt to get sparse-secret key material

## DKG | RTX-4090 | HEAAN2

Number of rounds	4
Number of parties	32
Computation	2.1s

[MHP+25]

# CONCLUSION

Th-FHE based on CKKS can be very efficient:

- most of the cost comes from FHE evaluation
- can be handled with GPU + HEAAN2
- decryption security & efficiency:
  - flooding
  - synchronous setting
- DKG via homomorphic KeyGen

# QUESTIONS?

[damien.stehle@cryptolab.co.kr](mailto:damien.stehle@cryptolab.co.kr)

# BIBLIOGRAPHY

- [AJL+12] G. Asharov, A. Jain, A. López-Alt, E. Tromer, V. Vaikuntanathan, and D. Wichs. *Multiparty computation with low communication, computation and interaction via threshold FHE*. In EUROCRYPT, 2012.
- [BGG+18] D. Boneh, R. Gennaro, S. Goldfeder, A. Jain, S. Kim, P. M. R. Rasmussen, and A. Sahai. *Threshold cryptosystems from threshold fully homomorphic encryption*. In CRYPTO, 2018.
- [CKKS17] J. H. Cheon, A. Kim, M. Kim, and Y. Song. *Homomorphic encryption for arithmetic of approximate numbers*. In ASIACRYPT, 2017.
- [HHN+26] J. Ha, G. Hanrot, T. Noh, J. H. Cheon, J. W. Kim, and D. Stehlé. *Private iris recognition with high-performance FHE*. Arxiv, 2026.
- [BGK+24] R. Bloemen, B. Gillespie, D. Kales, P. Sippl, and R. Walch. *Large-scale MPC: Scaling private iris code uniqueness checks to millions of users*. IACR eprint, 2024
- [CKSS25] H. Choe, J. Kim, D. Stehlé, and E. Suvanto. *Leveraging discrete CKKS to bootstrap in high precision*. CCS, 2025.
- [CPS26] F. Colin de Verdière, A. Passelègue, and D. Stehlé. *On threshold fully homomorphic encryption with synchronized decryptors*. IACR eprint, 2026
- [MHP+25] S. Min, G. Hanrot, J. H. Park, A. Passelègue, and D. Stehlé. *Distributed key generation for efficient threshold-CKKS*. IACR eprint, 2025