

# TFHE (FHE), ZHEnith (ZK) and Nexus (MPC)

NIST Workshop on Multi-Party Threshold Schemes: Jan 27<sup>th</sup> 2026

Speaker: Nigel P. Smart (Zama and KU Leuven)

Team:

Mathieu Ballandras,

Tore Kasper Frederiksen,

Nigel P. Smart,

Carl Bootland,

Marc Joye,

Titouan Tanguy,

Kelong Cong,

Benoît Libert,

Samuel Tap,

Daniel Demmler,

Jean-Baptiste Orfila,

Michael Walter

# Zama's NIST Submission

We aim to submit a single document containing the description of three (related) Cryptosystems



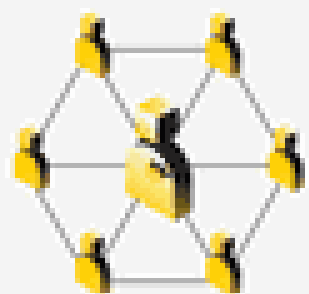
## **TFHE**

- A (relatively) full specification of the TFHE Fully Homomorphic Encryption scheme



## **ZHENith**

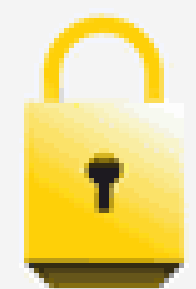
- A ZK proof system to establish proof of correctness, and plaintext knowledge, of encryption of TFHE ciphertexts.



## **Nexus**

- An MPC system over Galois Fields to (primarily) perform key generation and threshold decryption for FHE schemes (including BGV, BFV and TFHE)

# Cryptosystem : TFHE



# TFHE

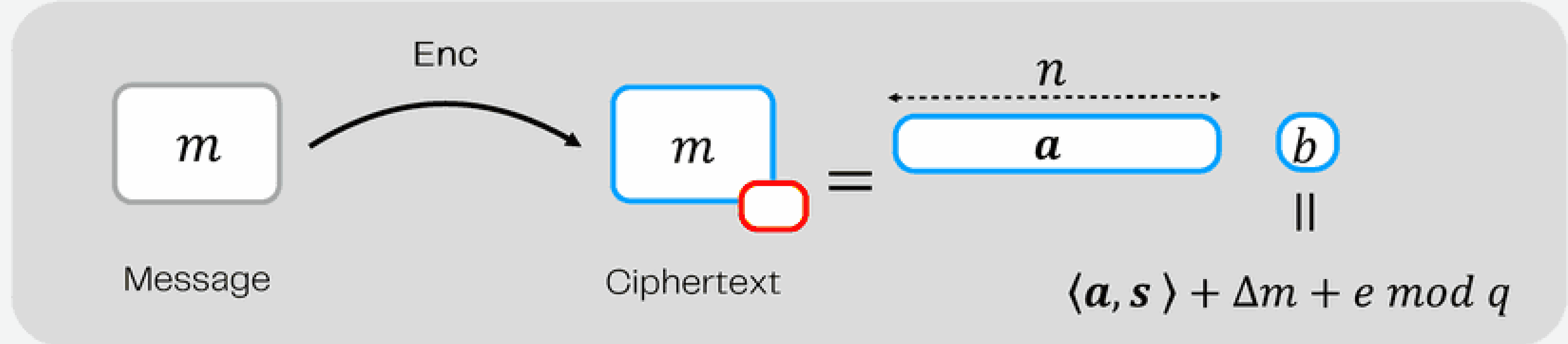
Aspects of our presentation of a public key variant of TFHE

- Full specification of all algorithms:
  - Efficient public key encryption, Programmable Bootstrapping, Switch-n-Squash, DeRandomized Evaluation (for sIND-CPA-D security)
- Derivation of parameters:
  - For plaintext spaces of one and four bits we provide specimen parameters
  - Parameters are 128-bit classical secure and NIST Level 1 Post-Quantum secure
- Security proofs of IND-CPA and sIND-CPA-D
- Example application: Confidential ERC-20 transfers on a blockchain
- Commercial Deployment: In Zama's Fhevm system

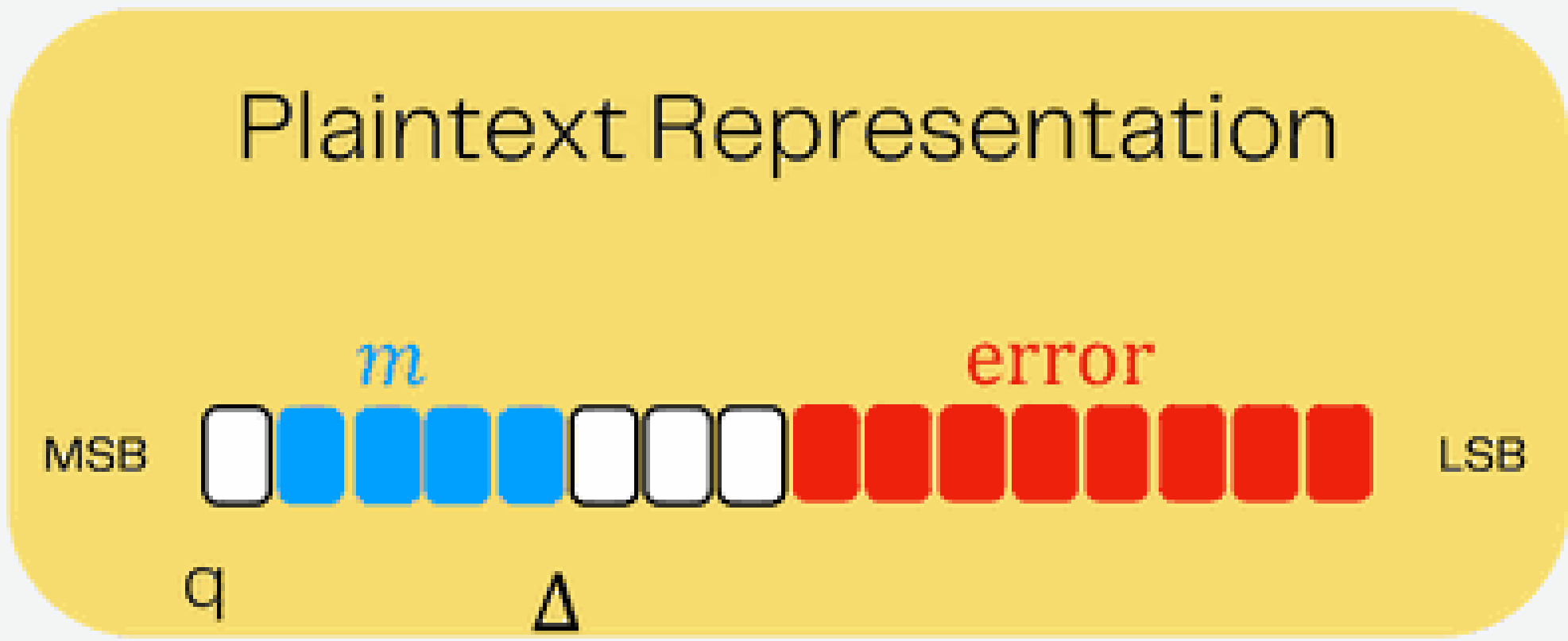


# LWE-based Ciphertexts

Learning With Errors

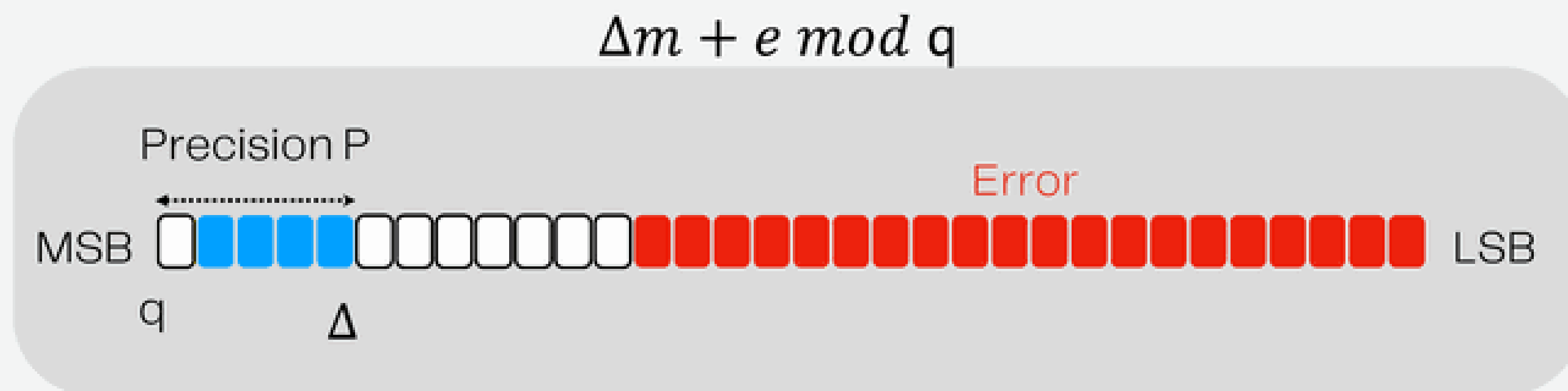


- Secret key  $\mathbf{s} \leftarrow \mathcal{U}(\{0,1\})^n$
- Mask  $\mathbf{a} \leftarrow \mathcal{U}(\mathbb{Z}_q)^n$
- Error  $e \leftarrow \mathcal{N}_\sigma$   
(a.k.a. noise)





# Plaintext Representation

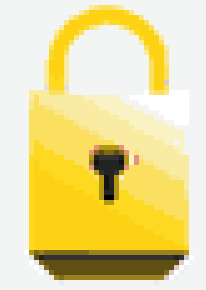


Example of parameters

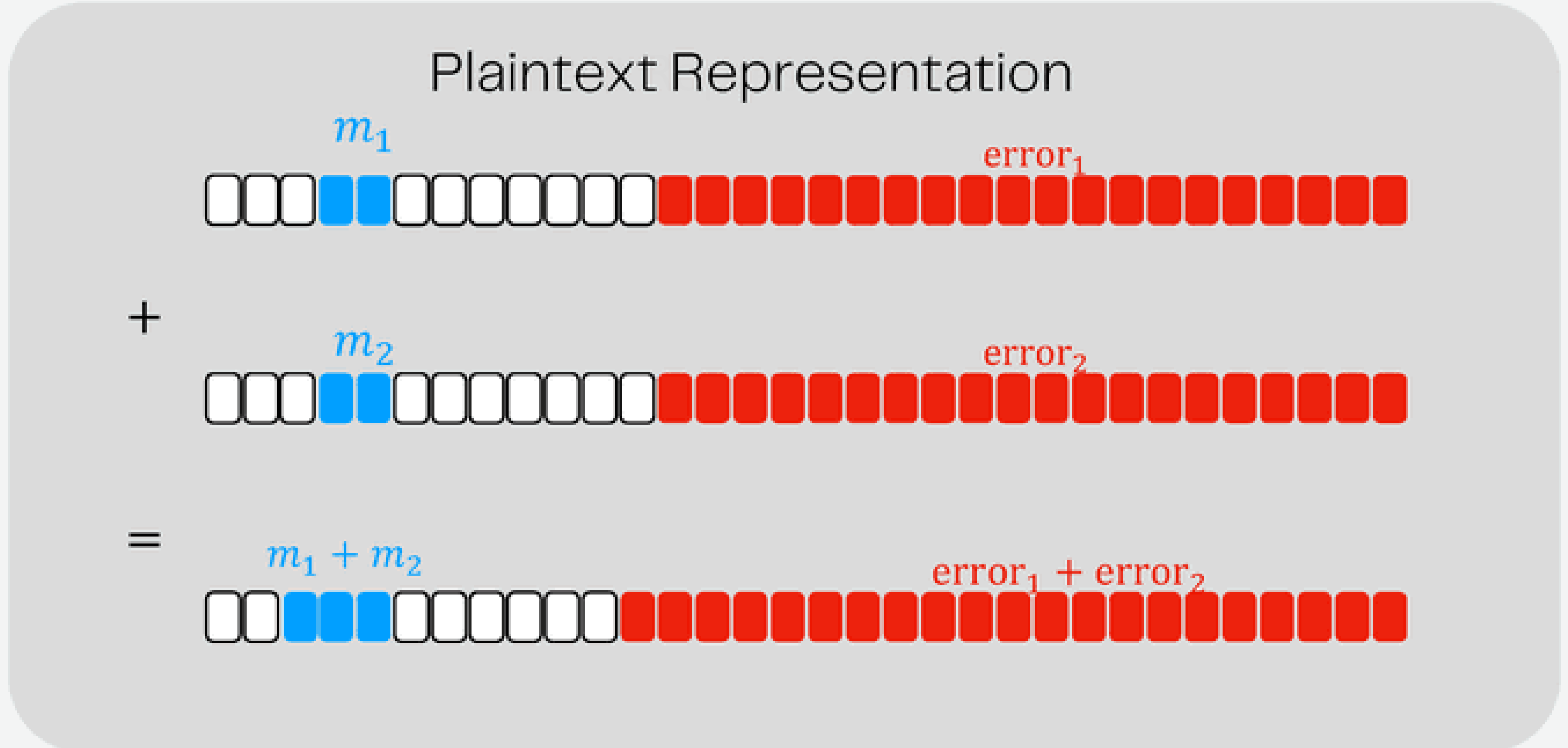
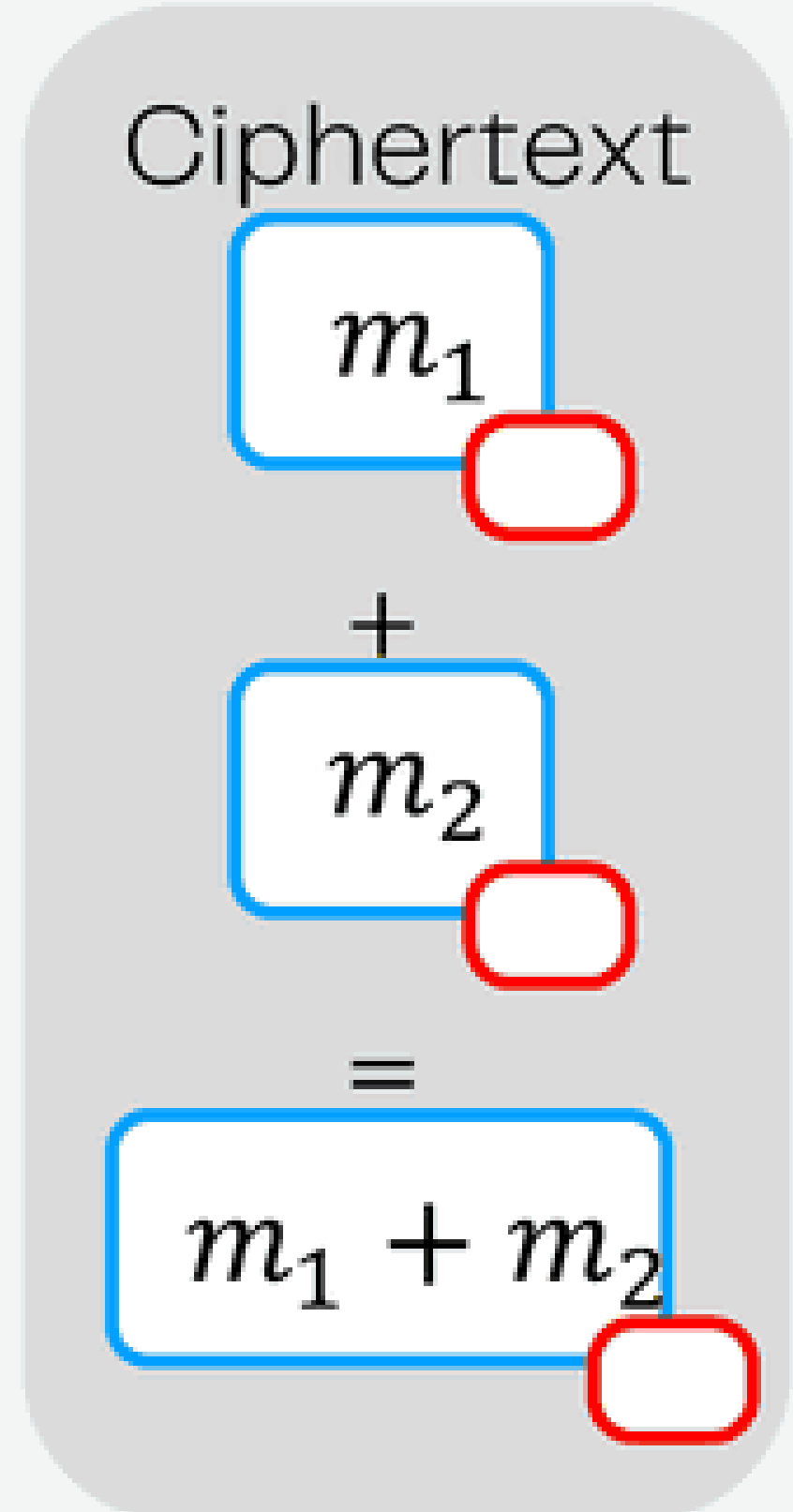
$$n \approx 900$$

$$q = 2^{64}$$

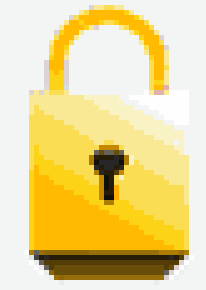
$$2 \leq P \leq 10$$



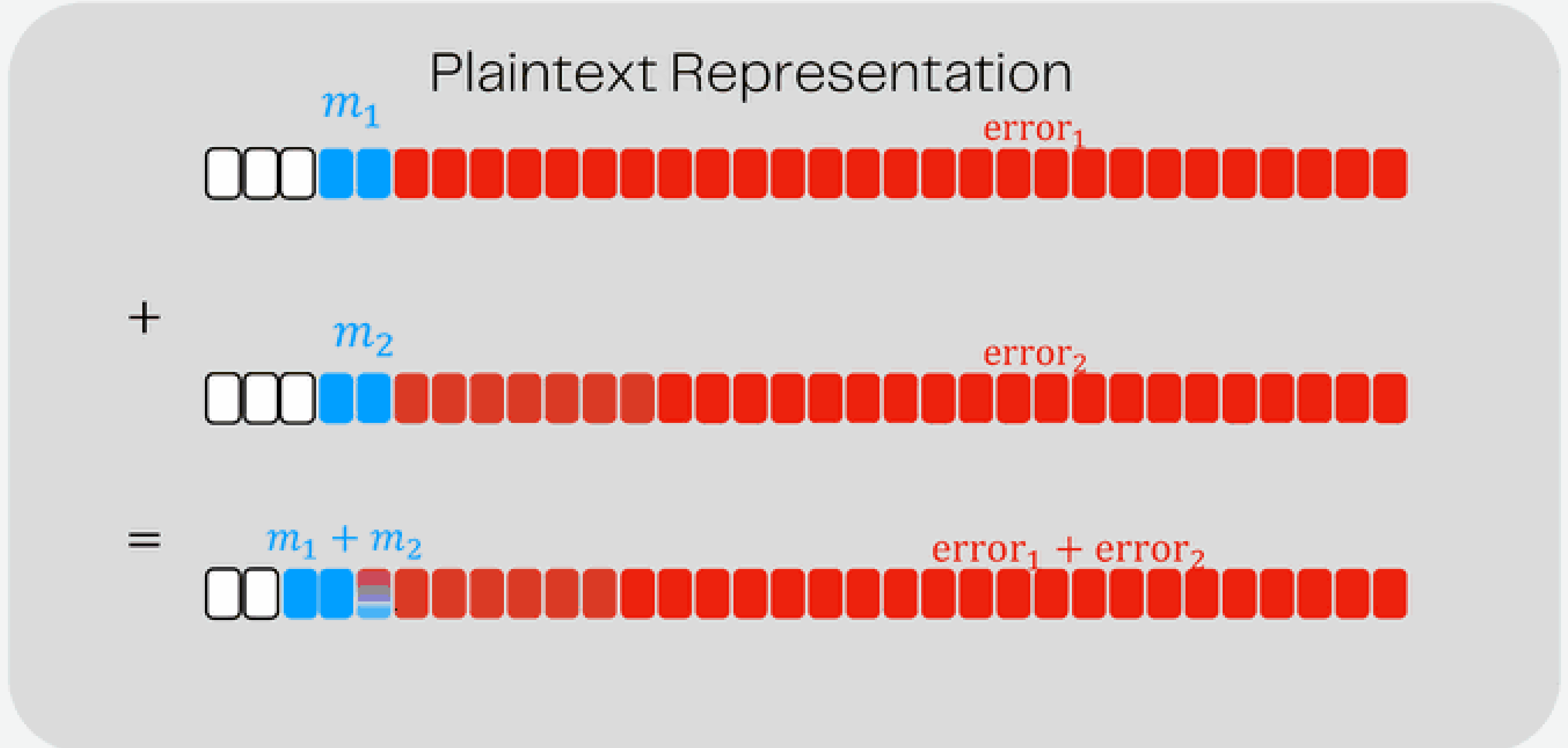
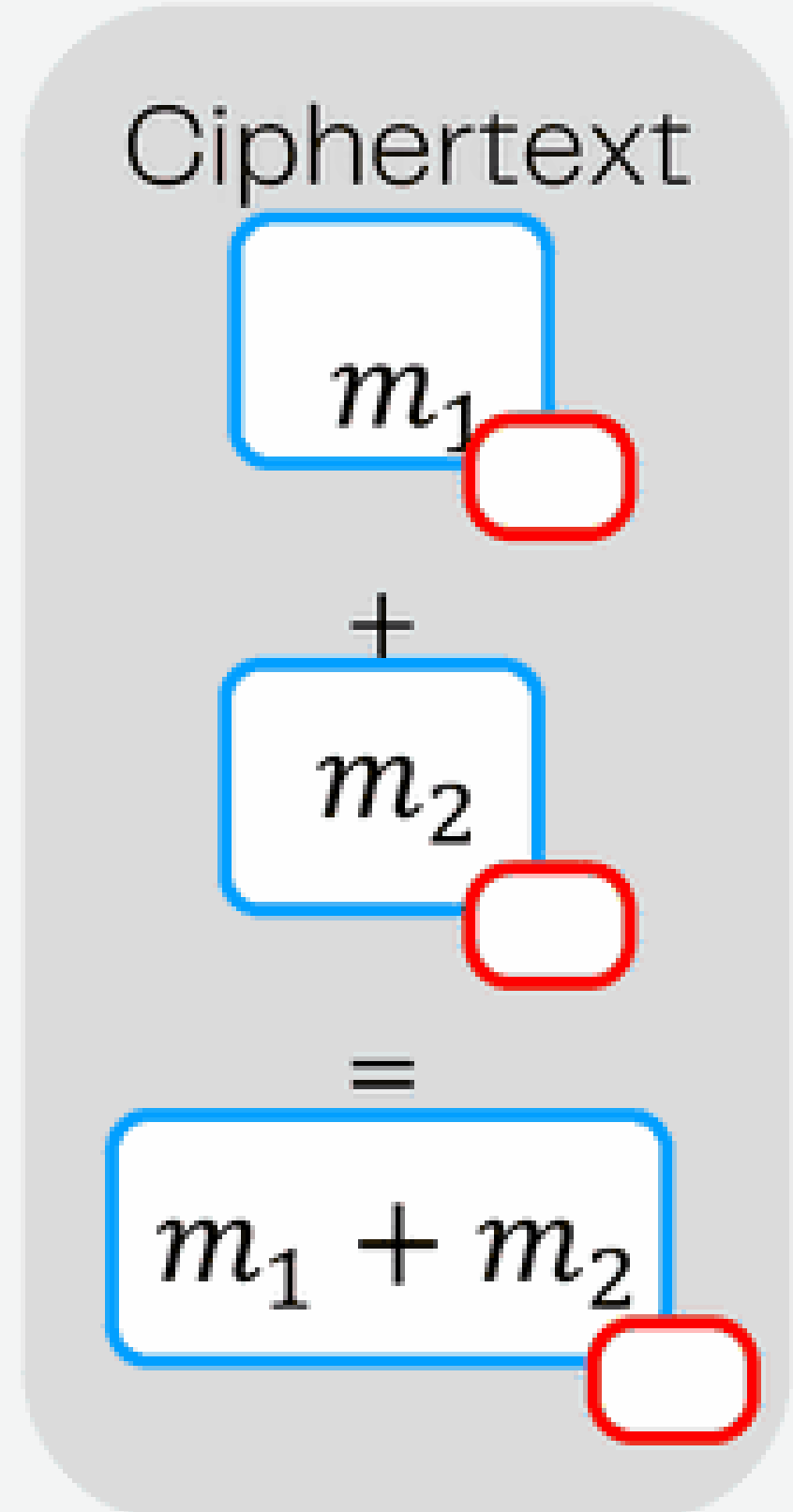
# Homomorphic Addition



Message & Error sizes increase

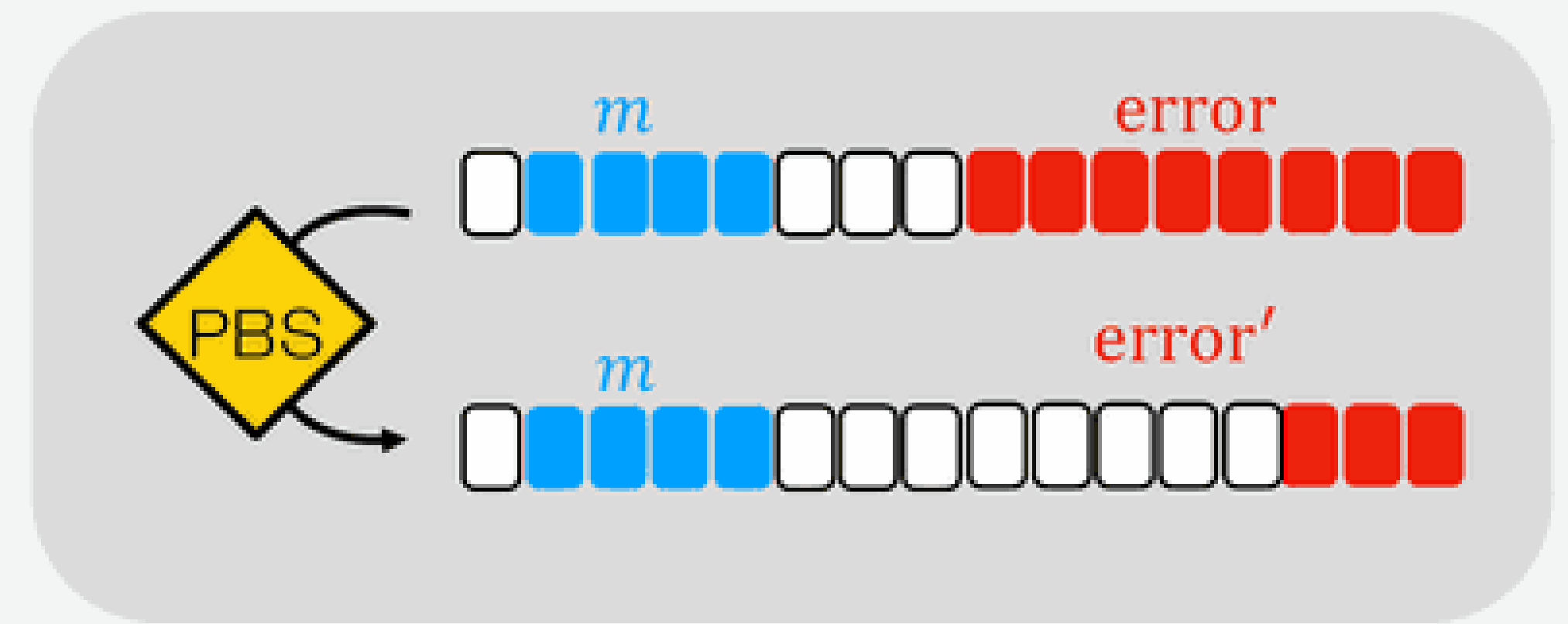
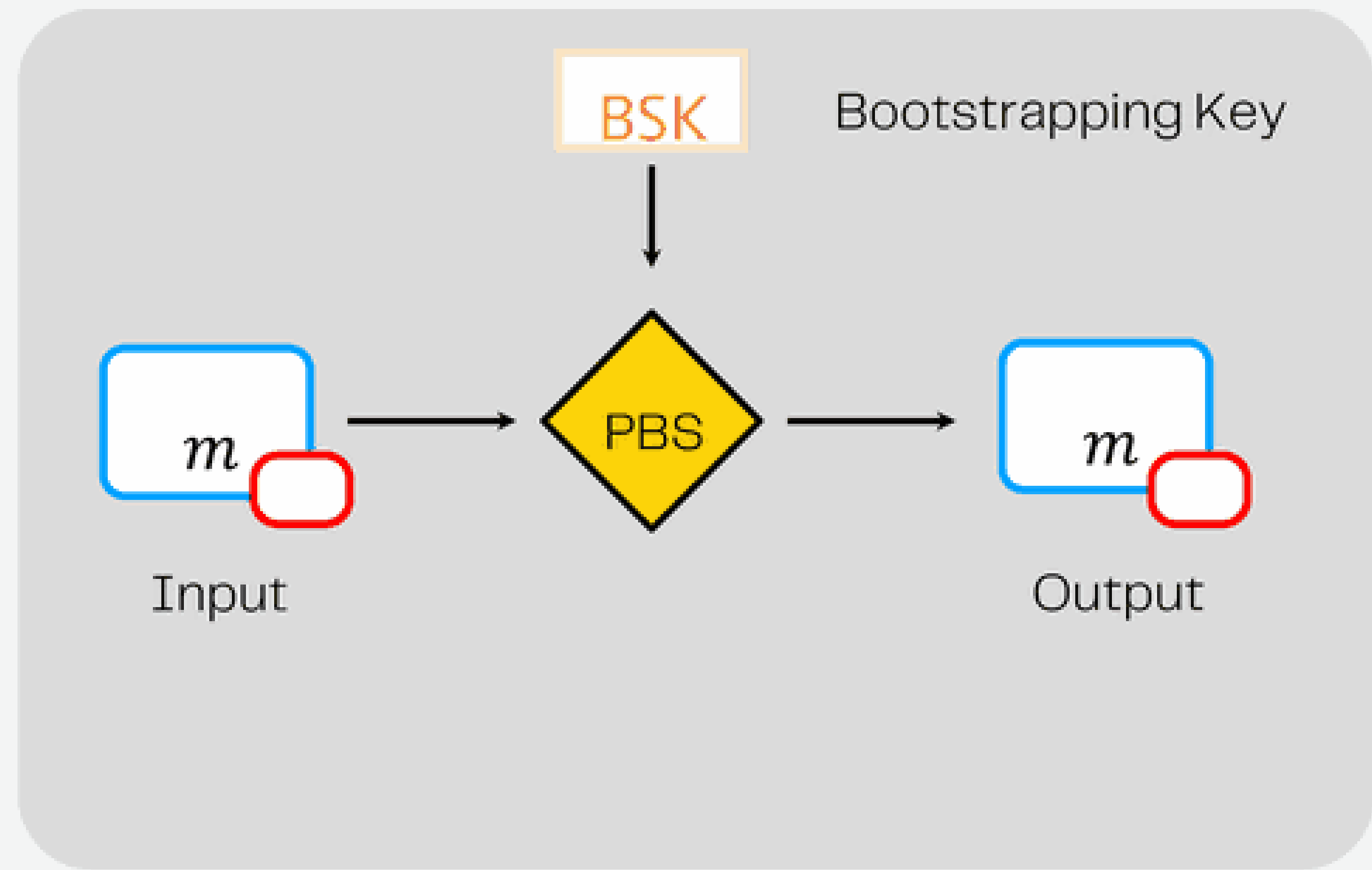


# Homomorphic Addition



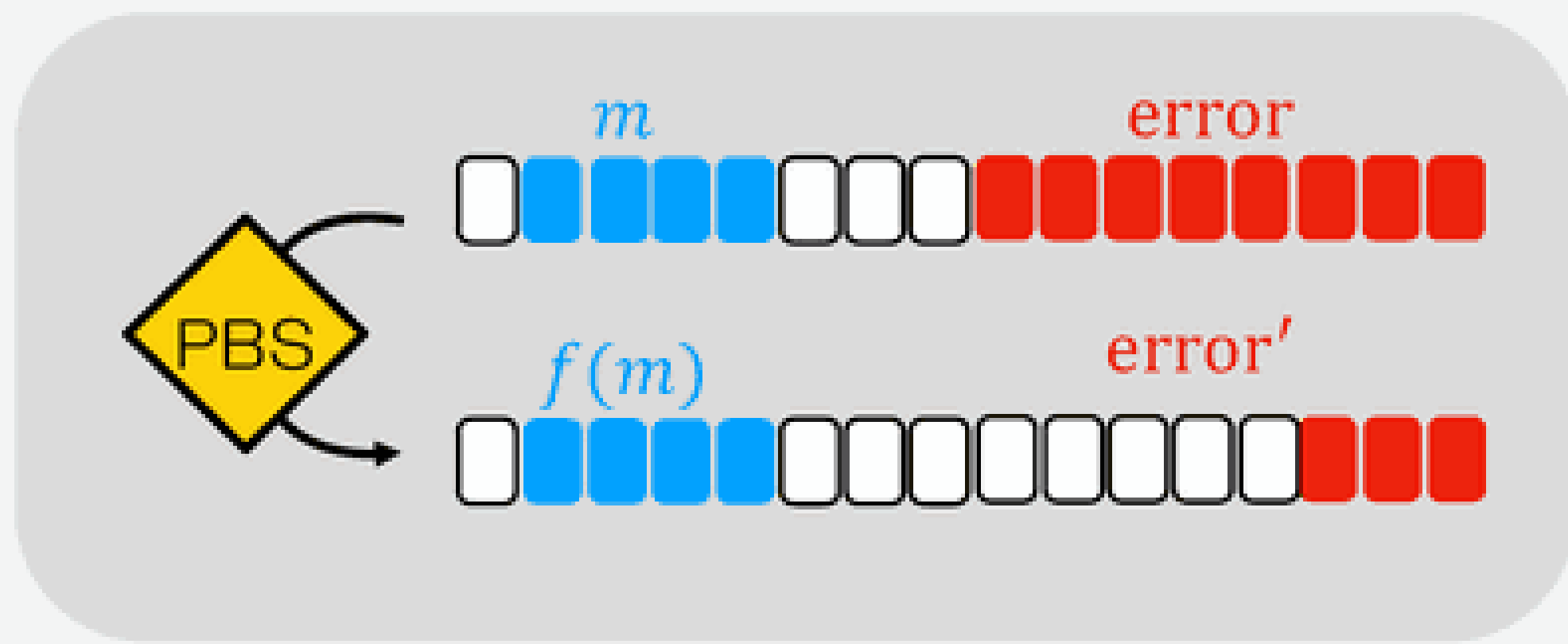
Noise management needed

# TFHE Programmable Bootstrapping (PBS)





# TFHE Programmable Bootstrapping (PBS)

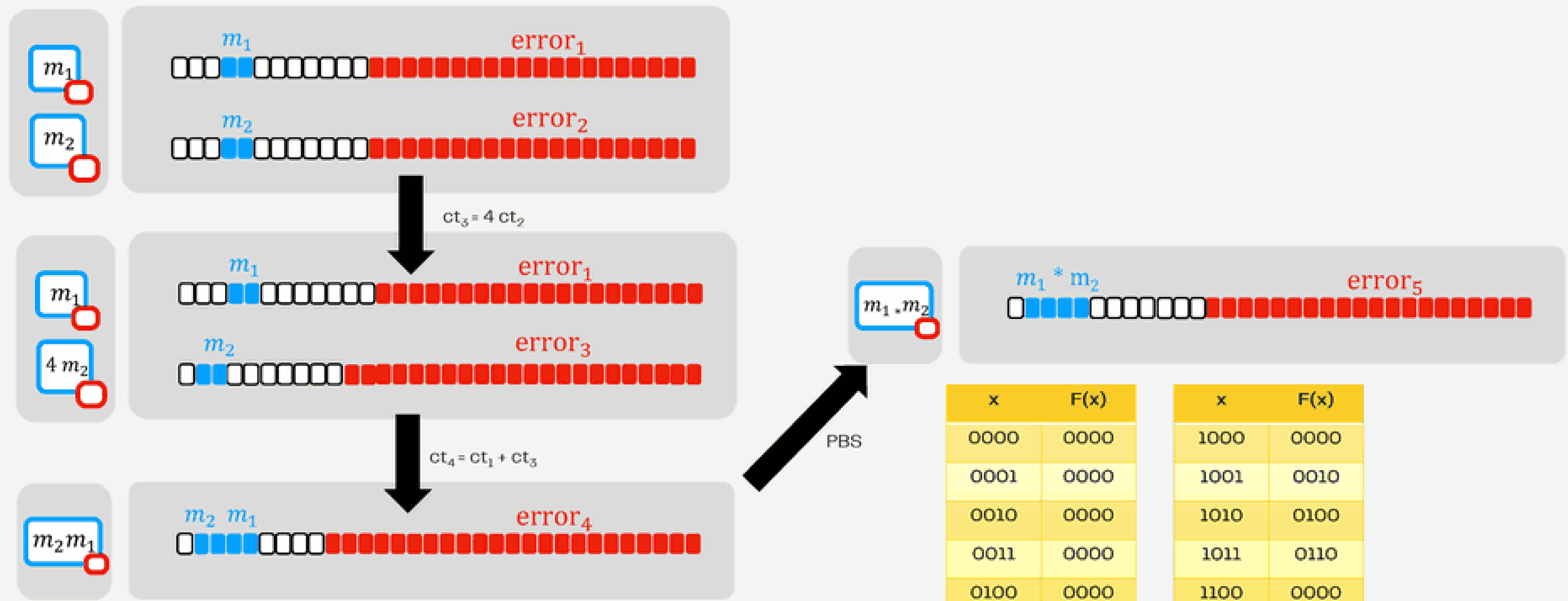


Low latency  
e.g.  $\approx 945\mu\text{s}$  for 4 bits

Small messages only:  
 $|m| \leq 10$  bits

Noise reduction &  
Homomorphic evaluation of any  $f(\cdot)$

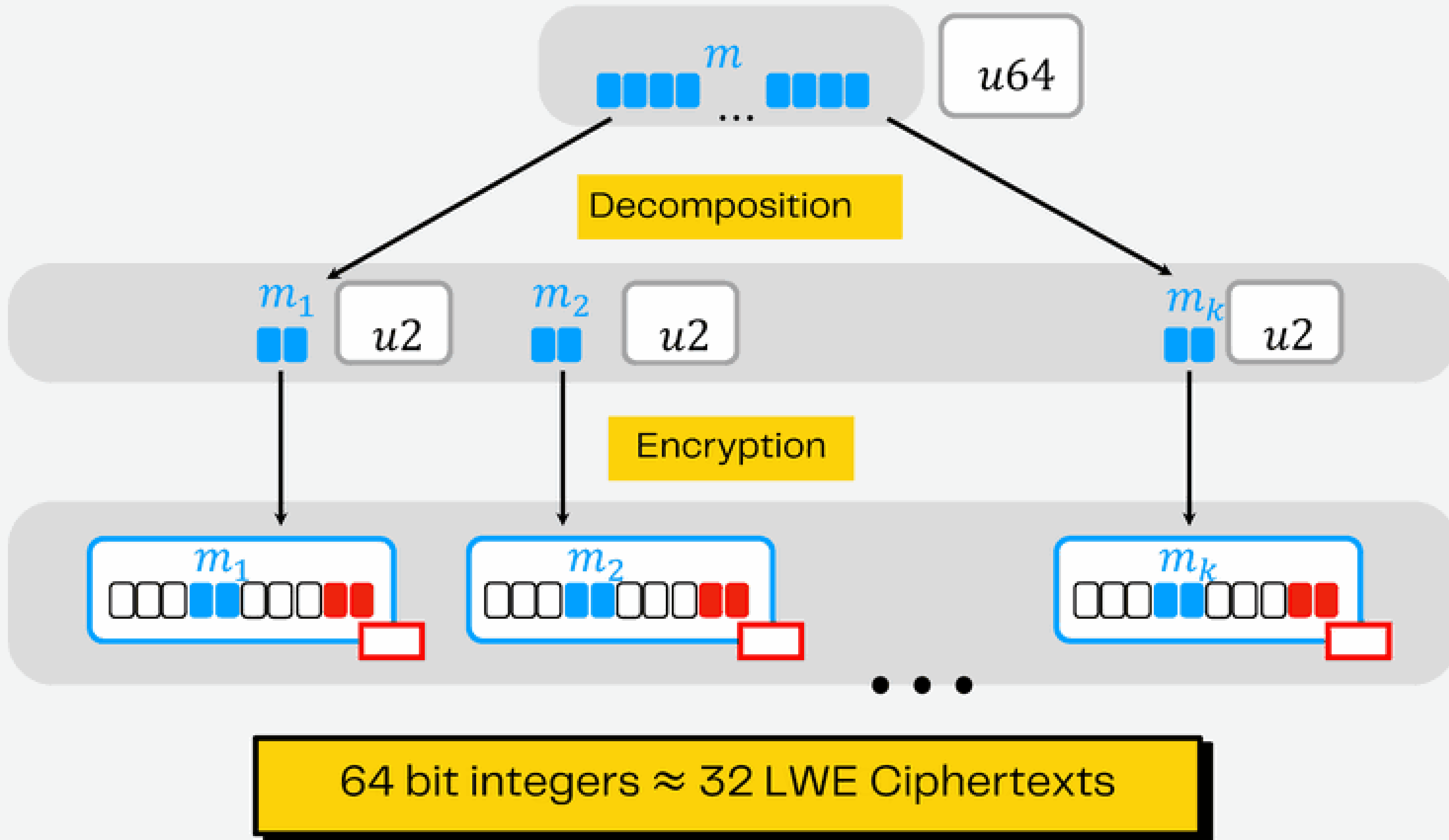
# Homomorphic Multiplication

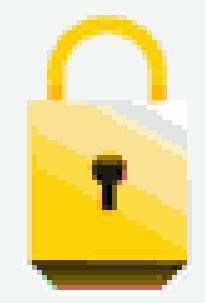


x	F(x)	x	F(x)
0000	0000	1000	0000
0001	0000	1001	0010
0010	0000	1010	0100
0011	0000	1011	0110
0100	0000	1100	0000
0101	0001	1101	0011
0110	0010	1110	0110
0111	0011	1111	1001



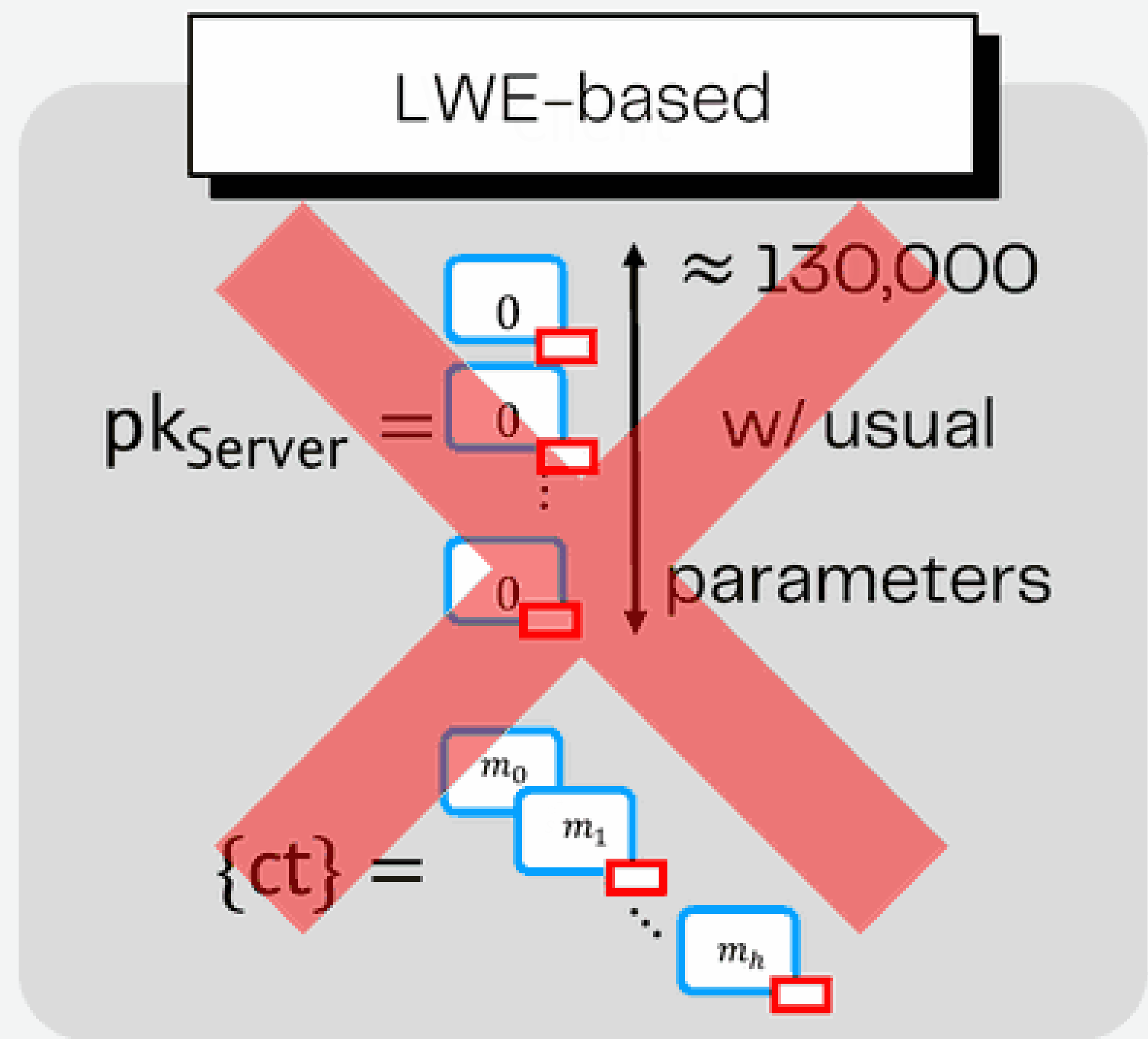
# Homomorphic Integers



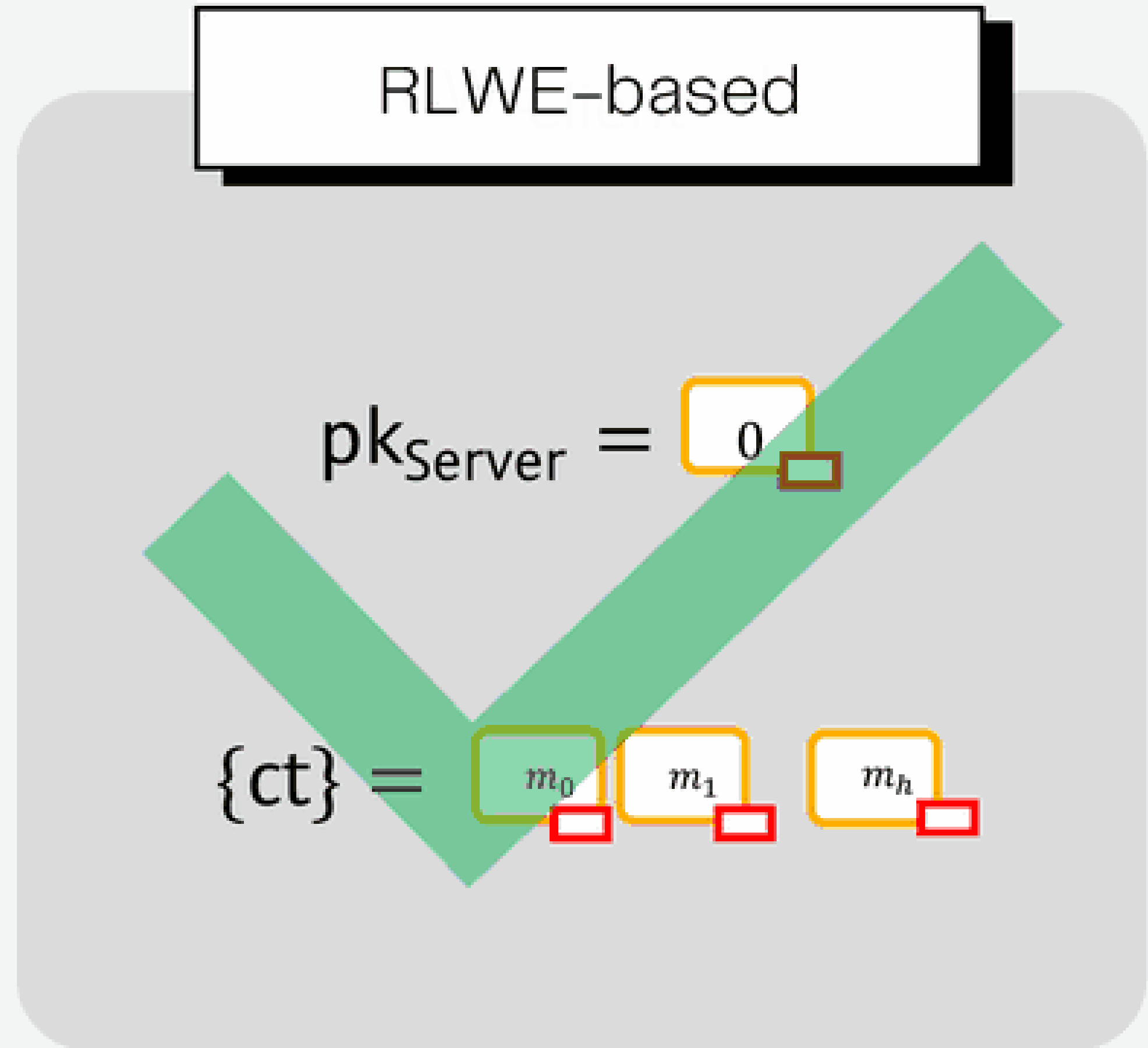


# Compact Public Key Encryption

TFHE, ZHEmith and Nexus



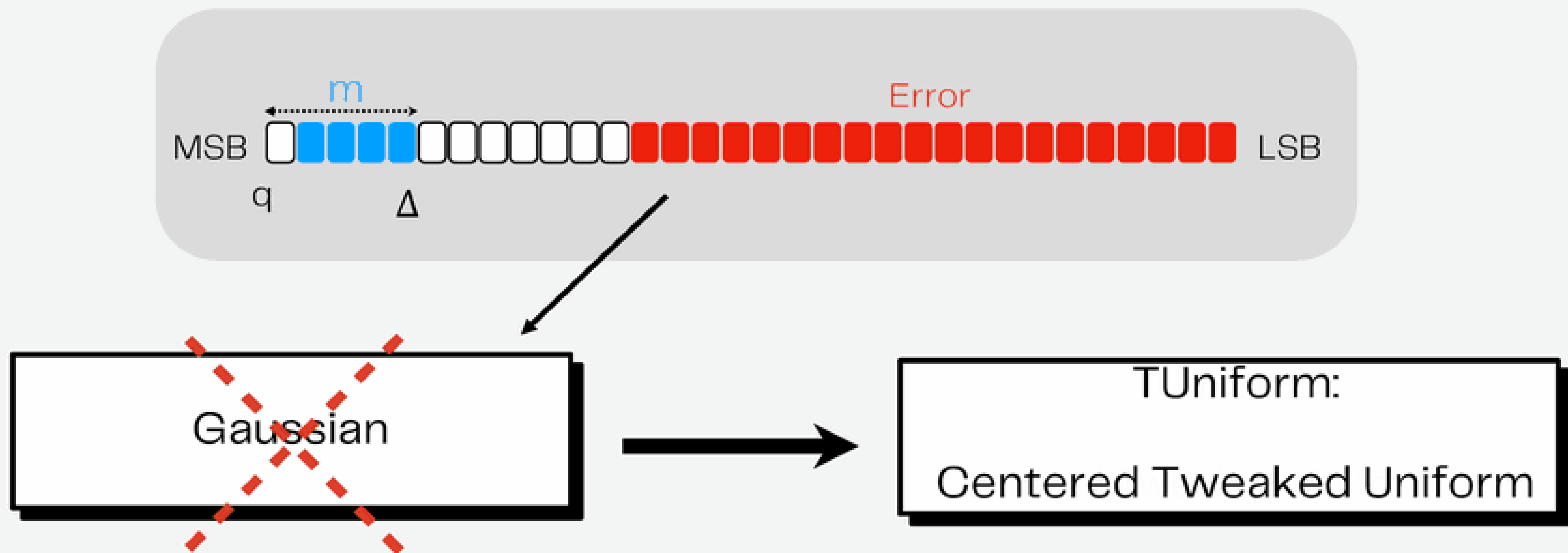
Large  $pk_{Server}$  & noisy ciphertexts



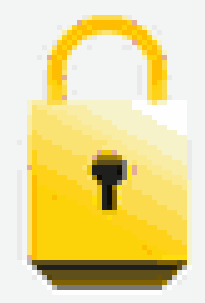
Smaller  $pk_{Server}$  & less noisy cts



# Tweaking Noise Distribution



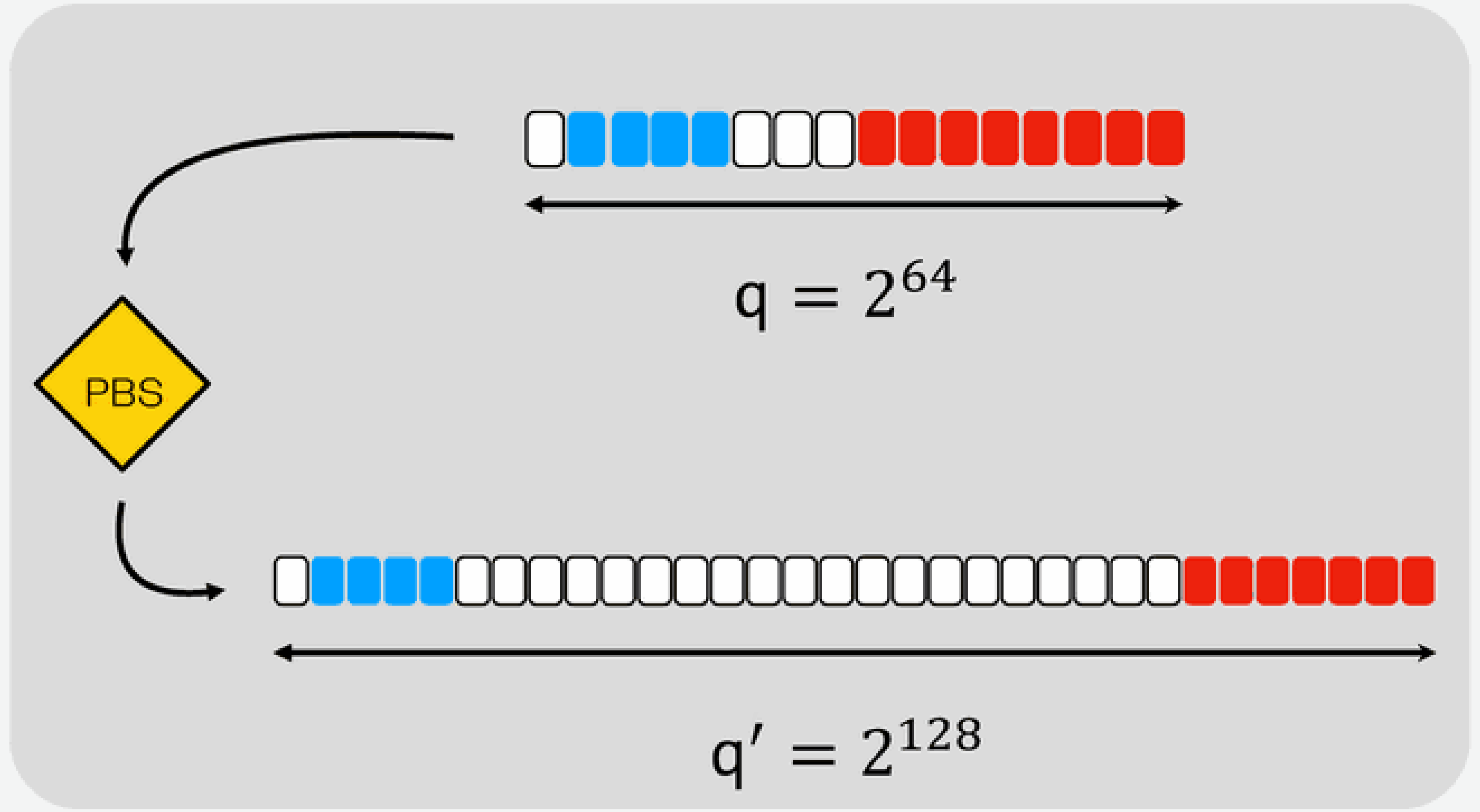
TUniform: Better suited for FHE and better fit for MPC key generation



# Switch-n-Squash

Enlarging  $q$

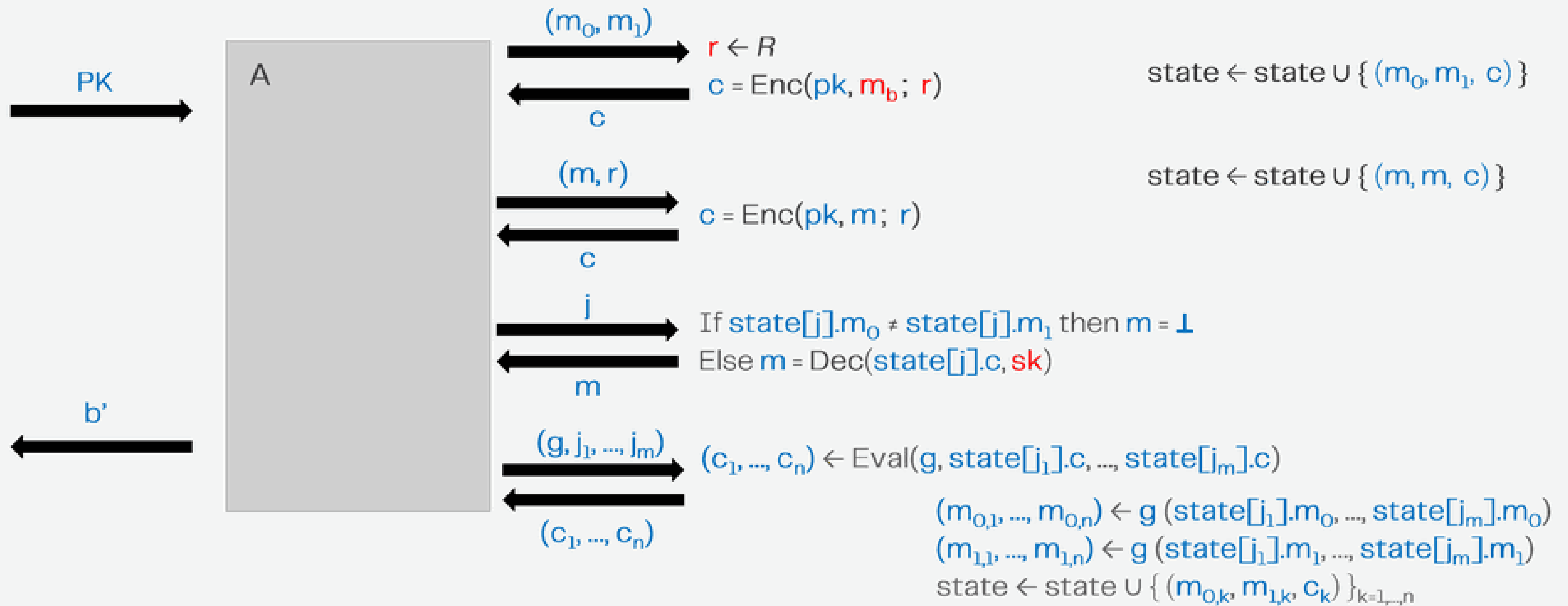
Reducing the error



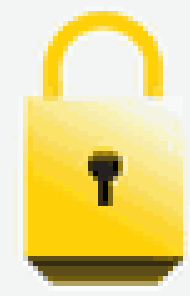
Now compliant with Noise Flooding for threshold decryption



# FHE Security Model : sIND-CPA-D



Adversary wins the game if  $b' = b$



# FHE Security Model : sIND-CPA-D

To achieve sIND-CPA-D security at a security level of 128 bits one needs

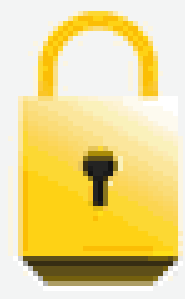
- IND-CPA security at a level of 128-bit security
- A failure probability for the evaluation operation of at most  $2^{-128}$
- The evaluation operation needs to be randomized (i.e. input ciphertexts are randomized before evaluation is performed).

This last point is a problem in practice as for verifiability one wants the FHE evaluation operation to be deterministic.

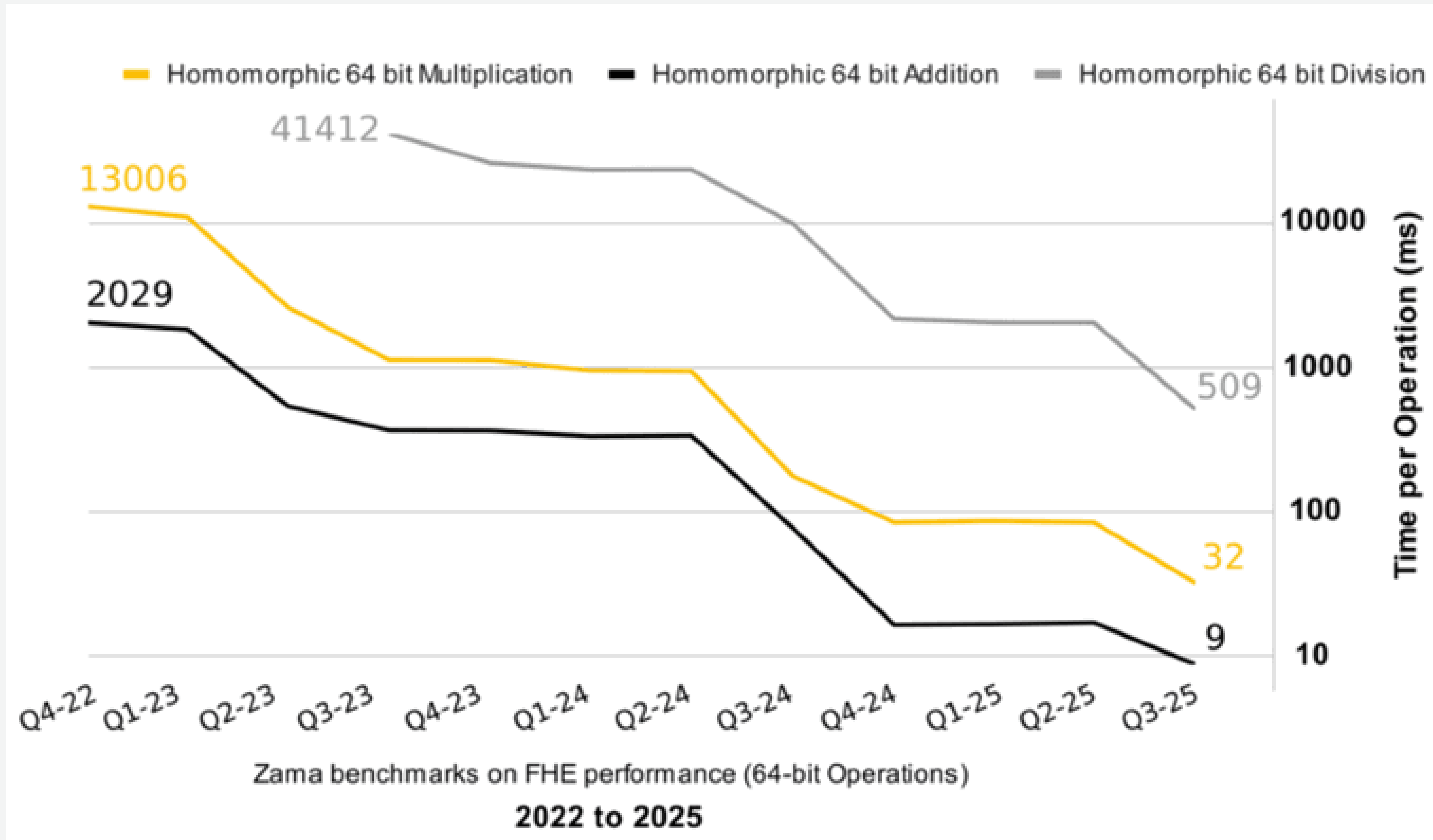
For TFHE we show this contradiction can be circumvented in the random oracle model.

Using the *lattice-estimator* we find parameters such that we obtain IND-CPA security at 128-bits and a failure probability of at most  $2^{-128}$

- Assuming lattice attacks are the best cryptanalytic attack on TFHE
- Assuming noise after various operations behaves much like Gaussians of a given standard deviation
- This can be partially justified by application to the CLT, it is also backed up by extensive experiments.



# FHE Computation Timings Over Time





# FHE Computation Latency

Operation \ Size	FheUint 8	FheUint 16	FheUint 32	FheUint 64	FheUint 128
Add / Sub (+,-)	4.74 ms	5.27 ms	6.91 ms	8.97 ms	13.4 ms
Mul (x)	9.26 ms	12.2 ms	18.3 ms	31.9 ms	79.0 ms
Equal / Not Equal (eq, ne)	3.6 ms	5.04 ms	5.42 ms	7.29 ms	8.3 ms
Comparisons (ge, gt, le, lt)	5.17 ms	6.63 ms	8.5 ms	10.6 ms	13.3 ms
Max / Min (max, min)	8.2 ms	10.0 ms	12.3 ms	14.9 ms	18.3 ms
Bitwise operations (&,  , ^)	1.44 ms	1.61 ms	1.73 ms	2.0 ms	2.28 ms
Div / Rem (/ , %)	50.0 ms	93.9 ms	205 ms	502 ms	1.36 s
Left / Right Shifts (<<, >>)	9.01 ms	11.3 ms	14.6 ms	17.9 ms	22.8 ms
Left / Right Rotations (left_rotate, right_rotate)	9.0 ms	11.3 ms	14.6 ms	17.8 ms	22.8 ms
Leading / Trailing zeros/ones	10.1 ms	13.3 ms	15.6 ms	20.0 ms	24.8 ms
Log2	9.63 ms	14.0 ms	16.4 ms	21.9 ms	26.6 ms
Select	3.13 ms	3.47 ms	3.93 ms	4.63 ms	5.66 ms



# FHE Computation Throughput

Operation \ Size	FheUint 8	FheUint 16	FheUint 32	FheUint 64	FheUint 128
Negation (-)	713 ops/s	703 ops/s	685 ops/s	648 ops/s	419 ops/s
Add / Sub (+,-)	679 ops/s	693 ops/s	655 ops/s	635 ops/s	415 ops/s
Mul (x)	542 ops/s	501 ops/s	248 ops/s	64.7 ops/s	13.1 ops/s
Equal / Not Equal (eq, ne)	564 ops/s	538 ops/s	546 ops/s	525 ops/s	376 ops/s
Comparisons (ge, gt, le, lt)	392 ops/s	382 ops/s	379 ops/s	347 ops/s	236 ops/s
Max / Min (max, min)	308 ops/s	305 ops/s	291 ops/s	247 ops/s	157 ops/s
Bitwise operations (&,  , ^)	2.35 k.ops/s	2.73 k.ops/s	2.67 k.ops/s	2.23 k.ops/s	1.55 k.ops/s
Div / Rem (/, %)	61.4 ops/s	38.4 ops/s	21.8 ops/s	9.6 ops/s	2.79 ops/s
Left / Right Shifts (<<, >>)	623 ops/s	581 ops/s	418 ops/s	213 ops/s	97.3 ops/s
Left / Right Rotations (left_rotate, right_rotate)	634 ops/s	586 ops/s	418 ops/s	213 ops/s	97.4 ops/s
Leading / Trailing zeros/ones	368 ops/s	355 ops/s	347 ops/s	308 ops/s	198 ops/s
Log2	409 ops/s	374 ops/s	364 ops/s	306 ops/s	199 ops/s
Select	863 ops/s	1.15 k.ops/s	1.06 k.ops/s	819 ops/s	392 ops/s

# Cryptosystem : ZHENith



# ZHEnith

When encrypting data in a public-key FHE application the encryptor may be dishonest

- He may create an invalid ciphertext, leading to selective failure attacks on the protocol.
- He may re-randomize someone else's ciphertext, leading to breaking independence of input for protocols.

Thus, an encryption should be accompanied by a Zero-Knowledge Proof of Correctness and Plaintext Knowledge.

The ZHEnith system accomplishes this for the TFHE scheme using a pairing-based scheme.

Such proofs are used in proofs of security of protocols based on FHE by the simulator.

- The proof allows the simulator to extract the plaintext of the dishonest parties ciphertext, and to ensure that dishonest parties enter valid ciphertexts.

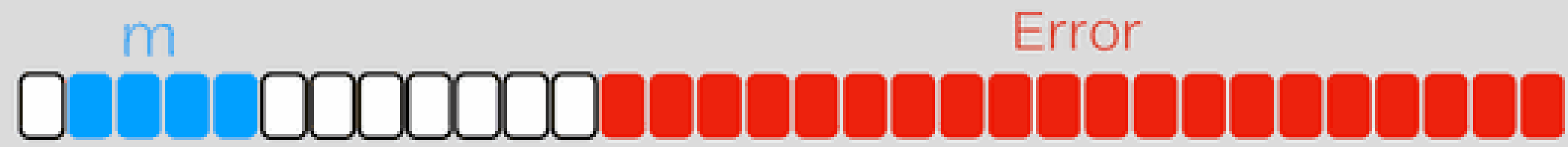
They are also needed in our security proof for sIND-CPA-D security too



# ZHEnith



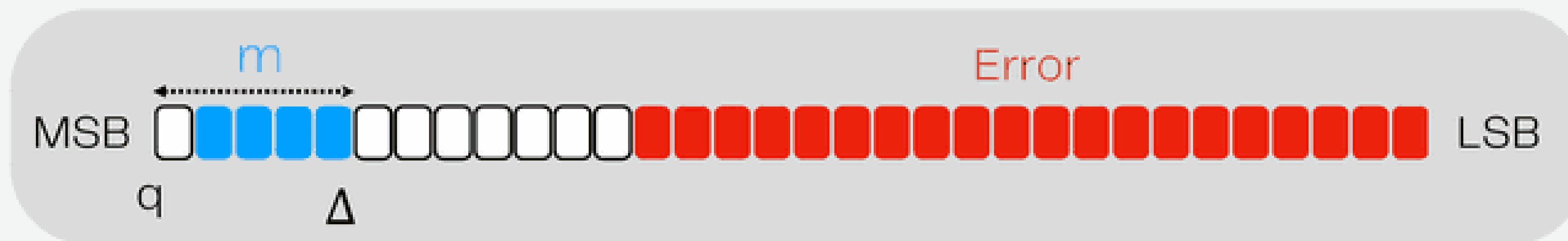
Bad encryption pattern: Message & Error mixed



Good encryption pattern: Distinct Error & Message



# ZHENith



Good encryption: Error & message bounded

Client

Server

$$\begin{aligned} ct &= \text{Enc}(m) \\ \pi &= \text{Prove}(ct) \end{aligned}$$



$$b = \text{Verify}(\pi)$$



# ZHENith

Aspects of our presentation of the ZKPoK system ZHENith

- The elliptic curve parameters are picked to be compatible with our TFHE parameters.
  - One could use ZHENith for BGV or BFV ciphertexts, but then the elliptic curves would need to be huge.
- Being based on elliptic curves the protocols soundness guarantee is only pre-quantum (at 128-bits of classical security)
- The zero-knowledge guarantee is however post-quantum.
  - Thus harvest-now/decrypt-later attacks are impossible to mount.
- Example application: Proving correctness of TFHE ciphertexts
- Commercial Deployment: In Zama's Fhevm system



# ZHEnith

Builds on vector commitments (Libert–Yung, TCC 2010)

- Uses a structured CRS

$$crs = (g, \{g_i = g^{\alpha^i}\}_{i \in [2n] \setminus \{n+1\}}, \{\hat{g}_i = \hat{g}^{\alpha^i}\}_{i \in [2n]})$$

- Commitment to  $\vec{x} \in Z_p^n$

$$C = g^\gamma \cdot \prod_{i=1}^n g_i^{x_i} = g^{\gamma + \sum_{i=1}^n x_i \cdot \alpha^i}$$

is opened at position  $i \in [n]$  by revealing  $\pi_i \in G$  such that

$$e(C, \hat{g}_{n+1-i}) = e(g_1, \hat{g}_n^{x_i}) \cdot e(\pi_i, \hat{g})$$

- Extends to prove  $\langle \vec{x}, \vec{y} \rangle = z$  for public  $\vec{y}, z \in Z_p$

$$e(C, \prod_{i=1}^n \hat{g}_{n+1-i}^{y_i}) = e(g_1, \hat{g}_n)^{\langle \vec{x}, \vec{y} \rangle} \cdot e(\prod_{i=1}^n \pi_i^{y_i}, \hat{g})$$



# ZHEnith

Goal: Let  $R = \mathbb{Z}[X]/(X^d + 1)$  and  $R_q = R/(qR)$  for a modulus  $q$ .

For public  $\vec{t}, \vec{a}_1, \dots, \vec{a}_M \in R_q^N$  prove knowledge of  $s_1, \dots, s_M \in R$  such that

$$\sum_{i=1}^M \vec{a}_i \cdot s_i = \vec{t} \pmod{q}$$

with  $\|s_i\|_\infty \leq B_i$  for all  $i \in [M]$ .

This allows proving (R)LWE relations, including validity of FHE ciphertexts when  $q$  is “small”.

- Thus, this includes the validity to TFHE ciphertexts



# ZHENITH

Idea: (del Pino – Lyubashevsky–Seiler: PKC 2019):

Consider the statement over  $\mathbb{Z}[X]/(X^d + 1)$

$$\sum_{i=1}^M \vec{a}_i \cdot s_i = \vec{t} + \vec{r} \cdot q \pmod{(p, X^d + 1)}$$

with  $\|s_i\|_\infty \leq B_i$  and  $\|\vec{r}\|_\infty \leq d \cdot M/2 \cdot \max_i(B_i)$

- Commit to  $((s_1, \dots, s_M) \mid \vec{r})$  in a DLOG-hard group  $G$  of order  $p \gg q$ .
- Prove that  $\|s_i\|_\infty \ll p$  and  $\|\vec{r}\|_\infty \ll p$ .
- Prove that

$$\sum_{i=1}^M \vec{a}_i \cdot s_i = \vec{t} + \vec{r} \cdot q \pmod{(p, X^d + 1)}$$

Remainder

Quotient



# ZHENITH

Rewrite the statement as a linear relation with a binary witness

$$\begin{array}{c} \text{A} \end{array} \left[ A_1, \dots, A_M \right] - q \cdot (I \otimes (1, 2, 4, \dots)) \cdot \begin{array}{c} \vec{s}_1 \\ \vdots \\ \vec{s}_M \\ \vec{r}_1 \\ \vdots \\ \vec{r}_N \end{array} = \vec{t} \pmod{p} \begin{array}{c} \vec{w} \end{array}$$



# ZHENITH

First Idea: Prove that a committed  $\vec{w} \in \{0,1\}^n$  is binary and satisfies the above equation

The equation is turned into an inner product relation  $\langle \theta^T \cdot \mathbf{A}, \vec{w} \rangle = \theta^T \cdot \vec{t} \bmod p$  for a random  $\theta$ .

The quantity  $\theta^T \cdot \mathbf{A}$  can be computed in  $O(d \cdot \log d)$  time when the  $\{A_i\}_{i=1}^M$  are structured.

Aggregation yields a proof  $(\hat{C}, C_y, \pi) \in \hat{G} \times G^2$  satisfying

$$e(\pi, \hat{g}) = e\left(C_y^{\delta_y} \cdot \prod_{i=1}^n g_{n+1-i}^{(\delta_{eq} \cdot t_i - \delta_y) \cdot y_i + \delta_\theta \cdot \bar{a}_\theta[i]}, \hat{C}\right) \cdot e\left(C_y^{\delta_{eq}}, \prod_{i=1}^n \hat{g}_i^{t_i}\right)^{-1} \cdot e(g_1, \hat{g}_n)^{-t_\theta \cdot \delta_\theta}$$



# ZHENith

**Second Idea:** Decrease the CRS size and the number of exponentiations by proving witness smallness by Lagrange's four-square theorem and approximate range-proofs (Lyubashevsky-Nguyen-Seiler: PKC 21)

Tight norm bounds proven using  $\ell_2$ -norm, this introduces some soundness slack when we are interested in the application in the  $\ell_\infty$ -norm.

- This is dealt with by modifying the noise analysis for TFHE, which in the end causes zero change in the final parameters

Tradeoff with  $O(1)$  exponentiations for the verifier and slightly longer proofs in  $(\hat{G}^5 \times G^8)$

- This gives proofs of size roughly 1KB using the BLS12-446 curve
- Uses KZG polynomial commitments



# ZHENith

Proving validity of a TFHE ciphertext with  $q = 2^{64}$  and  $n = 2048$ .

- Shortest SNARKs (Groth; EC'16): weak simulation-extractability in the AGM.
  - Arithmetic circuit with 110,000 R1CS constraints
  - Structured CRS of  $\approx 45,000$  KB
  - Prover computes  $\approx 800,000$  exponentiations
  - Verifier computes  $\approx 8200$  exponentiations
- New solution: simulation-extractability in the AGM+ROM.
  - Structured CRS of  $\approx 2,300$  KB using compression (4,500 KB without compression)
  - Prover computes  $\approx 97,000$  exponentiations.
  - Verifier computes 128 exponentiations and 8 pairings

Rust implementation using BLS12-446 curves for proving ciphertext validity

Prover runs in 536ms (on laptop using 16 cores)

Verifier runs in 45ms (using 16 cores on an c6i.4xlarge AWS instance)



# ZHEnith: CRS Generation

On Nov 25<sup>th</sup> 2025 thirteen companies executed the ZHEnith Common Reference String generation protocol for the launch of the Zama MainNet.

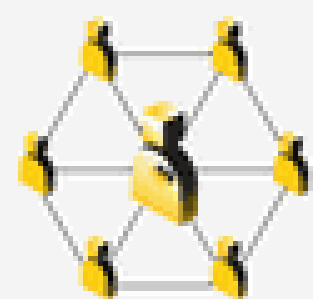
Protocol executed on c7a.16xlarge machines within AWS's eu-central-1 region.

- 64 Virtual CPUs
- 128 GB Memory

This was way more powerful than needed, but we did the Nexus DKG (see later) at the same time which needed such powerful machines.

Protocol took under a minute to execute, with each party contributing one randomization round to the CRS generation

# Cryptosystem : Nexus



# Nexus

To execute distributed key generation and decryption we utilize a generic robust MPC protocol over Galois Rings.

$$R = Z_q[X]/F(X)$$

The value  $q$  is a composite number

- A power of two in the case of TFHE
- A product of distinct medium size primes in the case of BGV and BFV

The polynomial  $F(X)$  is irreducible modulo all the prime factors of  $q$ .

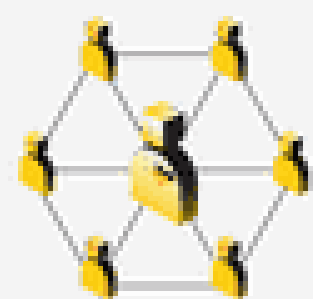
If  $p$  is the small prime factor of  $q$  then this allows us to create an exceptional set of size

$$p^{\deg F}$$

The exceptional set is the set of interpolation points for the underlying Shamir Secret Sharing scheme

So we need

$$n + 1 < p^{\deg F}$$



# Nexus

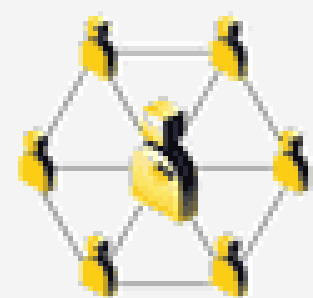
Our document will fully detail this MPC protocol, as well as how to use it to perform distributed key generation and decryption for BGV, BFV and TFHE.

The base MPC protocol is in the offline/online paradigm and comes in two variants

- **Variant 1:** For “small” values (say less than 10,000) of  $\text{binomial}(n,t)$ 
  - Uses PRSS for VSS execution
- **Variant 2:** For “large” values (say greater than 10,000) of  $\text{binomial}(n,t)$ 
  - Uses a classical VSS protocol

We call these variants **nSmall** and **nLarge**

The MPC protocol has different properties depending on whether one is in the nSmall or nLarge regime.



# Nexus: Generic MPC Protocol

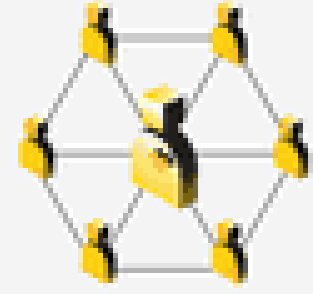
The generic MPC Nexus protocol has the following properties:

MPC Property	nSmall Regime	nLarge Regime
Offline Phase Threshold	$t < n/3$	$t < n/4$
Offline Phase Network	Synchronous	Synchronous
Online Phase Threshold	$t < n/3$	$t < n/3$
Online Phase Network	Asynchronous	Asynchronous
Adversary	Fully Malicious	Fully Malicious
Security Guarantee	Robust (G.O.D.)	Robust (G.O.D.)
Random Shared Bit Generation	All FHE-interesting moduli	Only for prime power moduli

The protocol is based on the Damgard–Nielsen 07 protocol (for  $t < n/4$ ), with a trick using PRSS keys to obtain robust security when  $t < n/3$  in the nSmall regime at low cost.

- We do not use player elimination framework to obtain robustness, simplifying the code significantly.

Broadcast is only required in the offline phase, for which we provide a simple synchronous broadcast protocol based on Bracha's asynchronous protocol



# Nexus: FHE Key Generation

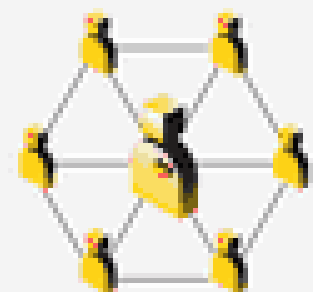
Distributed Key Generation protocols are given for BGV, BFV and TFHE

- Requires offline phase to generate Beaver triples
- Generation of secure random bits also done in the offline phase in the implementation
  - Theoretically could also be considered as part of the online phase

Due to complexities in generating secret shared secure random bits modulo a non-prime power, composite modulus the BGV and BFV distributed Key Generation only works in the profile nSmall.

- In general, secure random bit generation is complex, but an easy trick allows us to fix the issue, for non-prime power, composite moduli, in profile nSmall.

Commercial Deployment: In Zama's Fhevm system to generate a shared TFHE key.



# Nexus: FHE Key Generation

Between 19:06 Nov 25<sup>th</sup> 2025 and 01:46 Nov 28<sup>th</sup> 2025 thirteen companies executed the TFHE DKG protocol for the launch of the Zama MainNet.

Protocol executed on c7a.16xlarge machines within AWS's eu-central-1 region.

- 64 Virtual CPUs
- 128 GB Memory

Offline phase took **54 hours and 5 minutes**

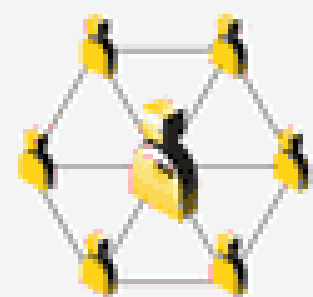
Online phase took **25 minutes**

Memory usage of each party was about **24 GigaBytes** for the offline phase, increased to **50 GigaBytes** during the execution of the online phase.

CPU usage was about 60% for the offline phase, and 80% for the online phase.

Data transmission at around 1 GigaBit per second between the parties.

Each party sent **10.5 TeraBytes** of information during the protocol execution.



# Nexus: Distributed Decryption

## In the clear decryption

Ciphertext  $(a, b = a \cdot s + e + \Delta m)$

Secret Key  $s$

Decryption

- $p = b - a \cdot s$
- $m = \lfloor p/\Delta \rfloor$

## Naïve threshold decryption

Ciphertext  $(a, b = a \cdot s + e + \Delta m)$

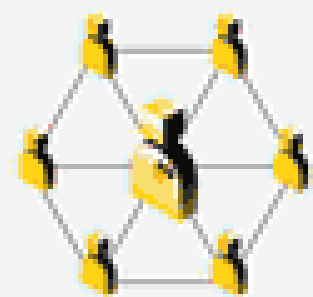
Secret Key  $[s]$

Decryption

- $[p] = b - a \cdot [s]$
- Open  $[p]$  to obtain  $p$
- $m = \lfloor p/\Delta \rfloor$

Insecure as revealing  $p$  reveals  $e$ , and hence information about  $s$ .

After a few decryptions the secret key will be revealed



# Nexus: Distributed Decryption – Method 1

## In the clear decryption

Ciphertext  $(a, b = a \cdot s + e + \Delta m)$

Secret Key  $s$

Decryption

- $p = b - a \cdot s$
- $m = \lfloor p / \Delta \rfloor$

## Flooded threshold decryption

Ciphertext  $(a, b = a \cdot s + e + \Delta m)$

Secret Key  $[s]$

Decryption

- $[p] = b - a \cdot [s] + [E]$
- Open  $[p]$  to obtain  $p$
- $m = \lfloor p / \Delta \rfloor$

Secure as long as  $E$  drowns out the  $e$  value.

Requires the value of  $E$  to be exponentially bigger than  $e$ .

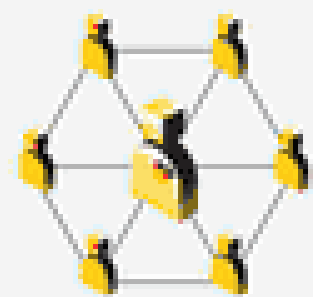
Hence, we need a lot of space for extra noise

For BGV and BFV one can fix parameters to ensure there is enough extra space for this noise.

For TFHE one applies the [Switch-n-Squash](#) operation to create the extra space.

- Can treat the [Switch-n-Squash](#) operation as pre-processing to obtain a super fast online phase.

Using a PRSS to generate  $[E]$  enables a one-round protocol for threshold decryption.



# Nexus: Distributed Decryption – Method 2

## In the clear decryption

Ciphertext  $(a, b = a \cdot s + e + \Delta m)$

Secret Key  $s$

Decryption

- $p = b - a \cdot s$
- $m = \lfloor p / \Delta \rfloor$

## MPC-based threshold decryption

Ciphertext  $(a, b = a \cdot s + e + \Delta m)$

Secret Key  $[s]$

Decryption

- $[p] = b - a \cdot [s] + [E]$
- $[m] = \lfloor [p] / \Delta \rfloor$
- Open  $[m]$  to obtain  $m$

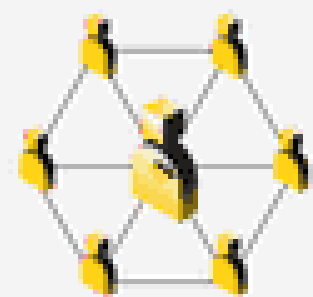
We compute the rounding in MPC

This requires us to be able to generate shared random bits in an offline phase.

This results in a many-round online protocol which is slower in WAN networks than Method – 1

Also requires an efficient offline phase to generate shared random bits

- Hence, we do not have this for BGV and BFV in profile nLarge.



# Nexus: Distributed Decryption

Two Distributed Decryption protocols are given for BGV, BFV and TFHE

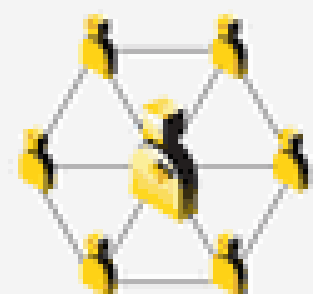
The first method

- The online phase is a **single round protocol**
- Requires no offline phase in profile nSmall
  - Thus, works for  $t < n/3$  and over asynchronous networks
- **In profile nLarge requires an offline phase**
- For BGV and BFV only works in profile nSmall, due to the need for an offline phase which generated shared random bits.

The second method

- Has a **multi-round online phase** over an asynchronous network
- The offline phase needs to generate secret shared random bits
  - So, this method also does not apply to BGV and BFV in profile nLarge

Commercial Deployment: In Zama's Fhevm system to execute TFHE decryptions.



# Nexus: Distributed Decryption

In our production system (running on AWS Hpc7a machines) we perform a Switch-n-Squash operation in about 850ms for a 64-bit encrypted integer

- Equating to an **amortized** time of  $850/32 \approx 26\text{ms}$  per ciphertext
- In our application this is treated as offline pre-processing
- The application machines are processing more than just Switch-n-Squash operations at the same time, so this is a real timing, as opposed to an “academic” timing.

The latency of our 13 party MPC protocol to perform the threshold decryption (assuming the Switch-n-Squash operation has been performed) is roughly 25 decryptions of 64-bit encrypted integers per second in a full production system

- Equation to an **amortized** time of  $1000/(25*32) \approx 1.25\text{ms}$  per ciphertext
- Due to ping-times the real latency is around **10ms**.

When just performing threshold decryption and no other operations we can achieve rates of **100-300** threshold decryptions of 64-bit encrypted integers per second, i.e. **3000-9000** ciphertext decryptions per second.

# Let's Get Together...

# Zama's Test and Main Nets

Zama's TFHE, ZHENith and Nexus technologies are used to secure our encrypted blockchain application

TFHE, ZHENith and Nexus

Metric	TestNet	MainNet
Launch Date	July 2025	Jan 2026
Transactions	6,676,045	38,668
Contracts Deployed	26,978	123
Users	207,030	8,756
ZHENith ZKPoKs Created	482,434	30,454
Public Threshold Decryptions	422,395	4,175
Threshold Decryptions to User	702,206	29,347
Total Transaction Fees	8,923 Sep ETH	15 ETH

Main Data From: [https://dune.com/zama\\_fhe/protocol-overview](https://dune.com/zama_fhe/protocol-overview) data taken at 09.00 on Jan 26th 2026



# Dutch Auction

Between Jan 21<sup>st</sup> and 24<sup>th</sup> 2026 Zama executed a sealed-bid Dutch Auction for 10% of the token supply of the token \$ZAMA.

- Run on top of the Zama MainNet

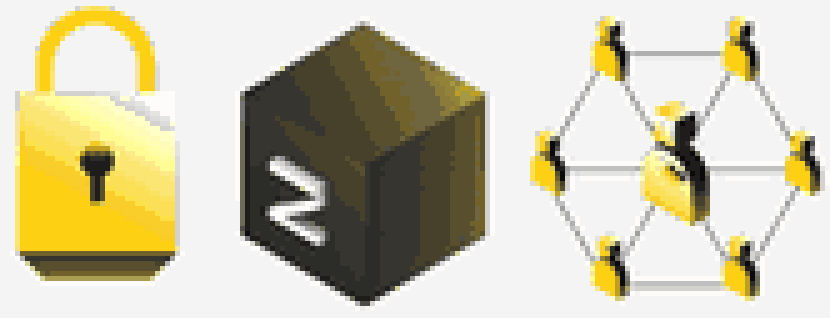
Parties entered an encrypted amount as to how much \$ZAMA they wished to purchase at a given price point.

The market clearing price was computed as the lowest price for which all the tokens were allocated.

- Encrypted bids via TFHE accompanied by ZHENith proofs
- Homomorphic computation of the clearing price via smart contracts using TFHE
- Homomorphic computation of the settlement process using TFHE

In addition to the auction running on MainNet, parties who did not want to engage in the blind auction could make bids on the exchanges Coinlist and KuCoin

- These bids were then merged with the encrypted bids on MainNet to compute the final clearing price



# Dutch Auction

Some statistics on the MainNet auction

Statistic	Value
Total Number of Registrations	18,752
Number of Bidders	7,653
Total Number of Bids <sup>1</sup>	17,559
Cancelled Bids	2,959
Final Number of Bids <sup>1</sup>	14,600
Total Value Shielded <sup>2</sup>	\$121 million
Time to Compute the Clearing Price	771 seconds
Final Clearing Price	\$ 0.05
Total Value of Bids Fulfilled	\$ 44 million

1) Parties could bid multiple times, at multiple price points.

- About 57% of users made one bid, 20% made two bids.
- Maximum number of bids per user was ten.
- Number of bids registered on the auction app (excluding ones from Coinlist and KuCoin).

2) Value Shielded: Total amount of USD which was encrypted and held in encrypted form

- Note, not all the shielded value was necessarily bid in the auction.

Main Data From: [https://dune.com/zama\\_fhe/zama-public-auction](https://dune.com/zama_fhe/zama-public-auction) (on 09:00 UST 26th Jan 2026)

# Issues

# Reproducibility Issues

We see two issues with reproducibility of implementation results (both timings and KAT vectors).

1. TFHE uses floating point arithmetic to accelerate operations via FFTs
  - The accuracy is hardware dependent, thus unless the same hardware is used then the ciphertexts results are not comparable, even if the underlying plaintexts are the same.
  - We will attempt to mitigate this issue, but depending on the precise difference between the NIST test machine and our machines KAT vectors may not be comparable.
2. FHE keys are huge, thus running distributed key generation on a single machine is infeasible.
  - Recall in our commercial deployment we utilized 13 AWS large instances, each run by a separate company, and the total run time was over two days.
  - To mitigate this issue, we will provide “baby” (i.e. insecure parameters) in order for the software to be run in test mode on NISTs single desktop machine.

Thank you

**ZAMA**

# CONTACT

[nigel@zama.org](mailto:nigel@zama.org)

[zama.org](https://zama.org)

[github.com/zama-ai](https://github.com/zama-ai)

[zama.org/community](https://zama.org/community)