X-Cipher: Achieving Data Resiliency in Homomorphic Ciphertexts

<u>Adam Caulfield</u>, Nabiha Raza, Peizhao Hu

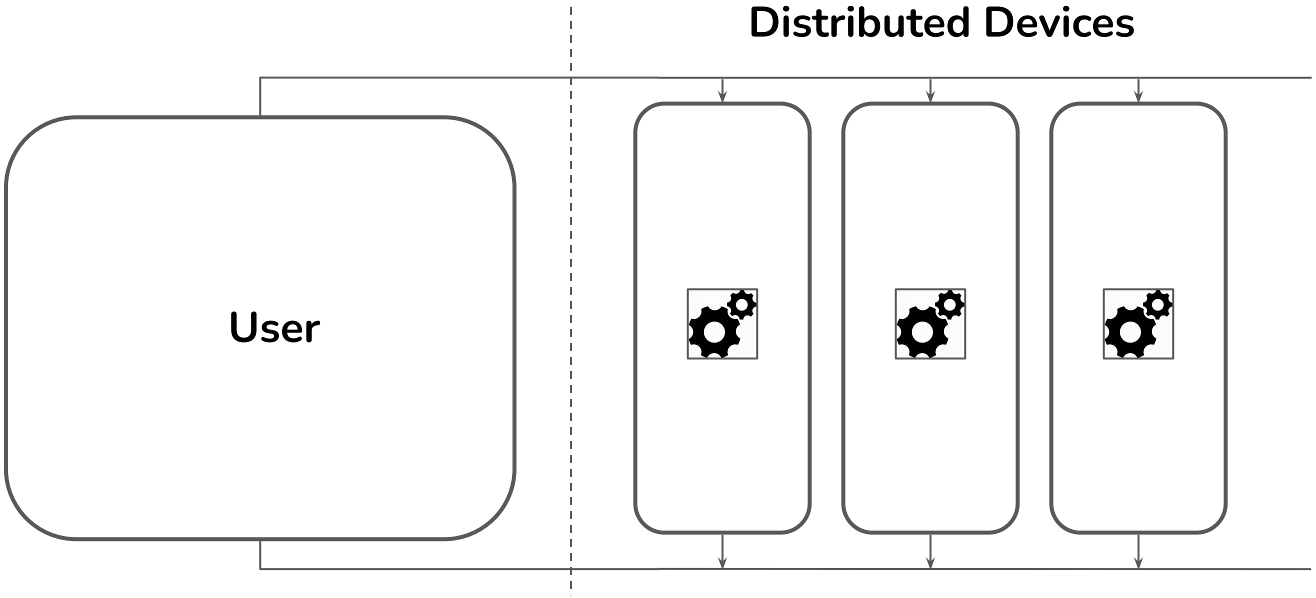
Rochester Institute of Technology
Rochester, New York, USA

ICICS
Mytilene, Greece
2024

RIT | Rochester Institute of Technology

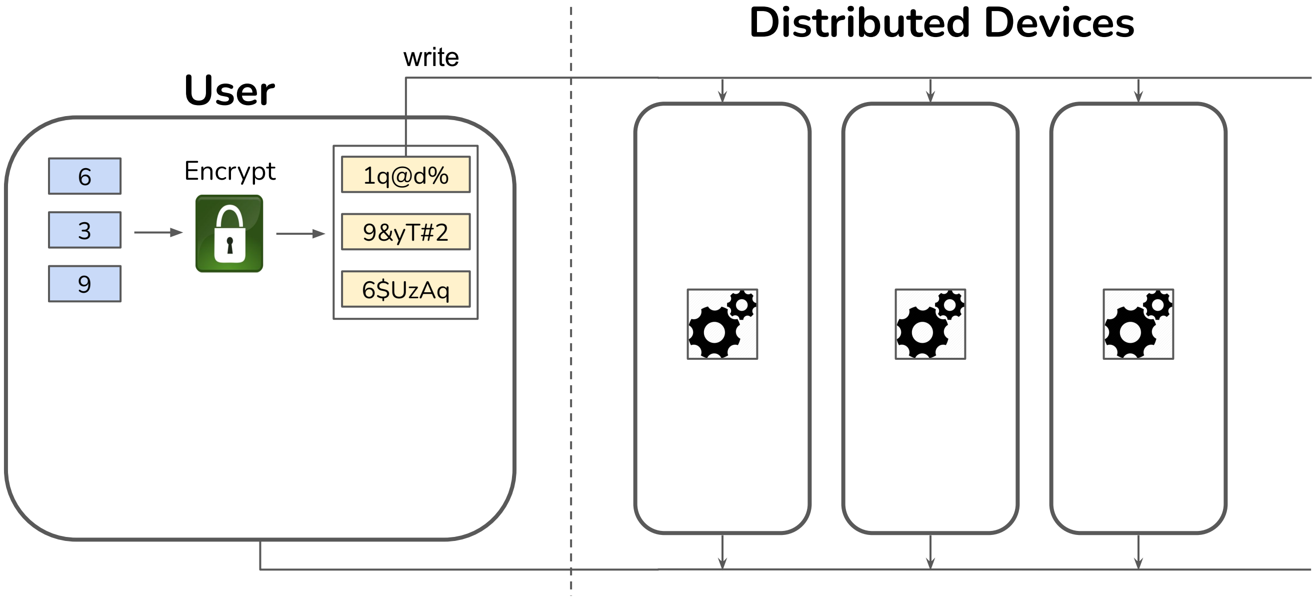
Computing on the Cloud

- Outsourcing computations to the cloud has become incredibly common



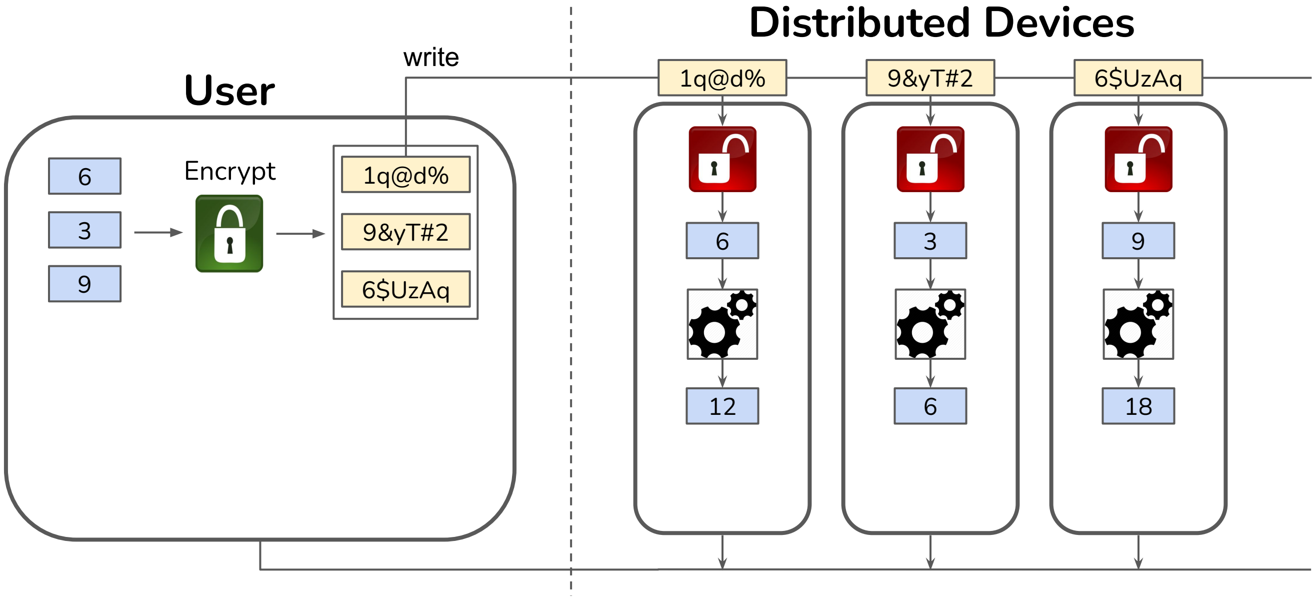
Computing on the Cloud

- To leverage cloud services on secrets, User's might encrypt their data in-transit



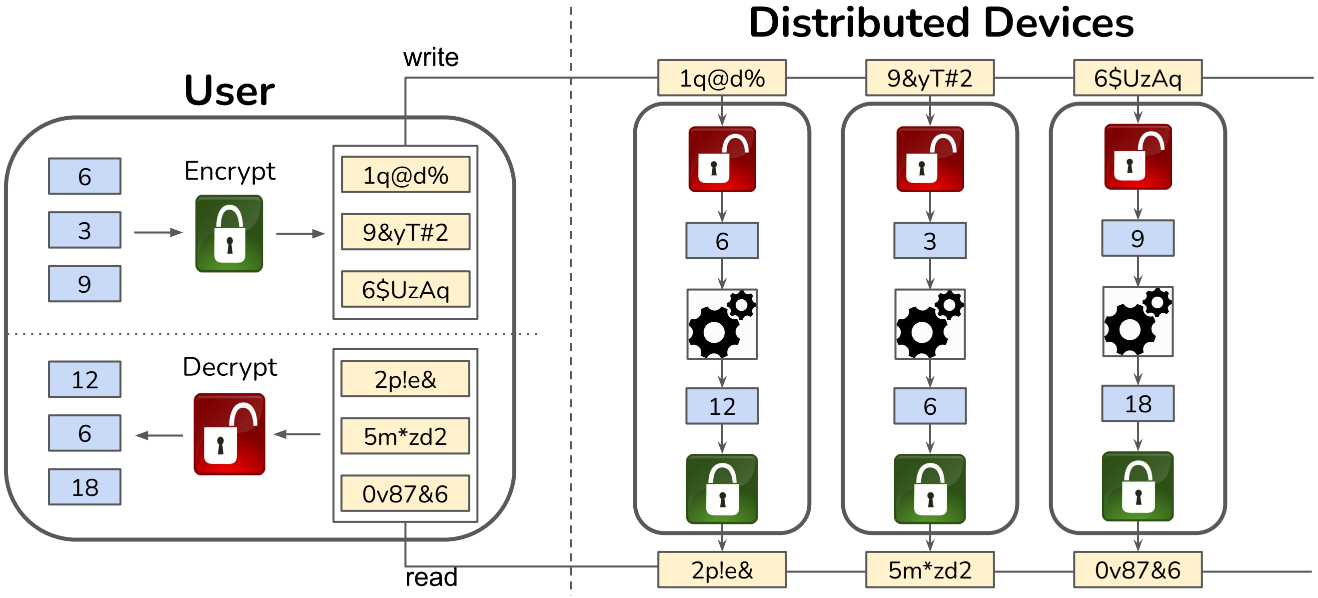
Computing on the Cloud

- Using a shared shared key, the cloud would decrypt the secrets to compute their outputs



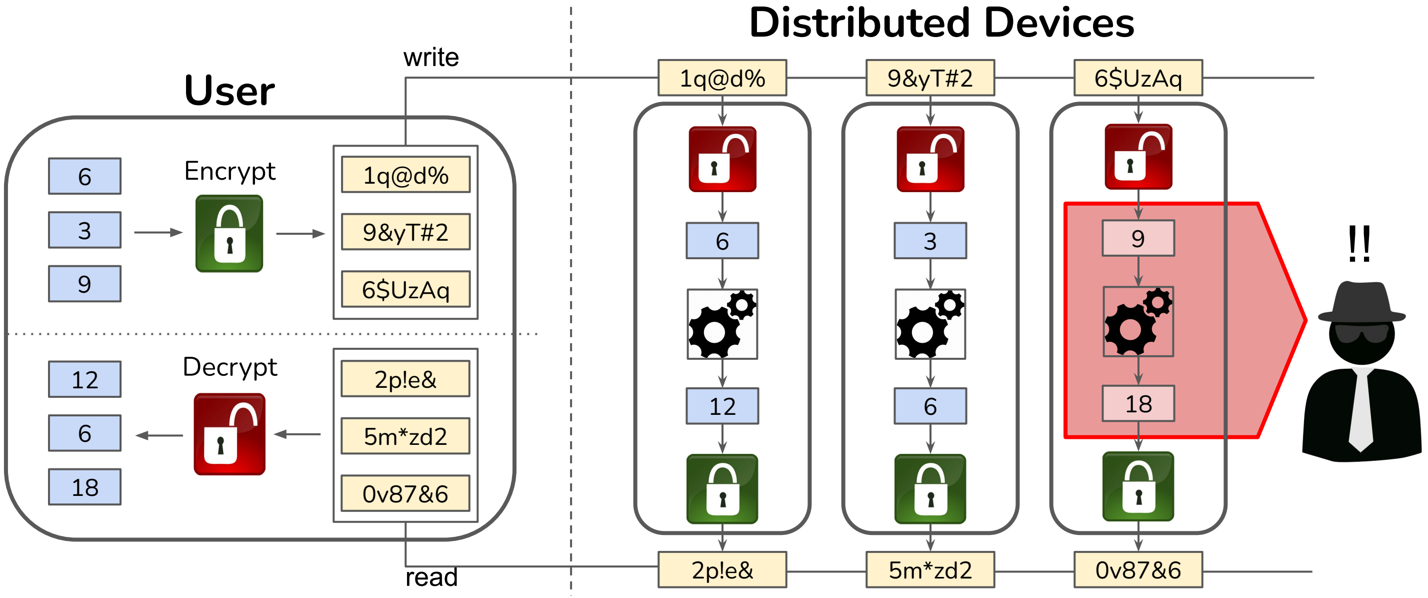
Computing on the Cloud

- Servers transmit the encrypted output back to the user



Computing on the Cloud

- Since data is decrypted during computations, could lead to potential leakages



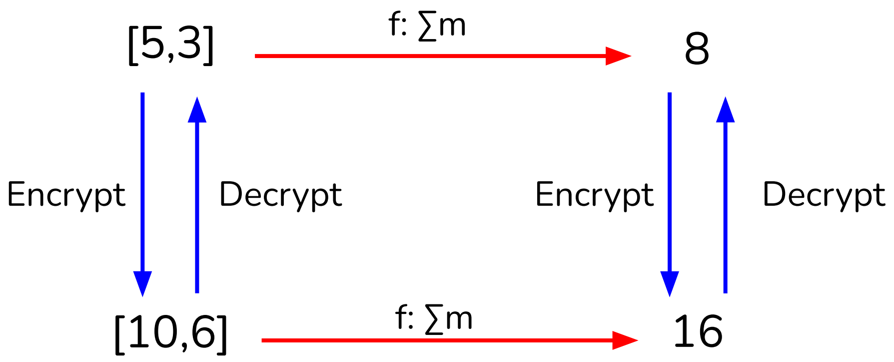
A solution: Homomorphic Encryption (HE)

- Produces ciphertexts that can undergo computations without decryption

- Protect data while in-storage, in-transit, and ***in-use***

Toy Example:

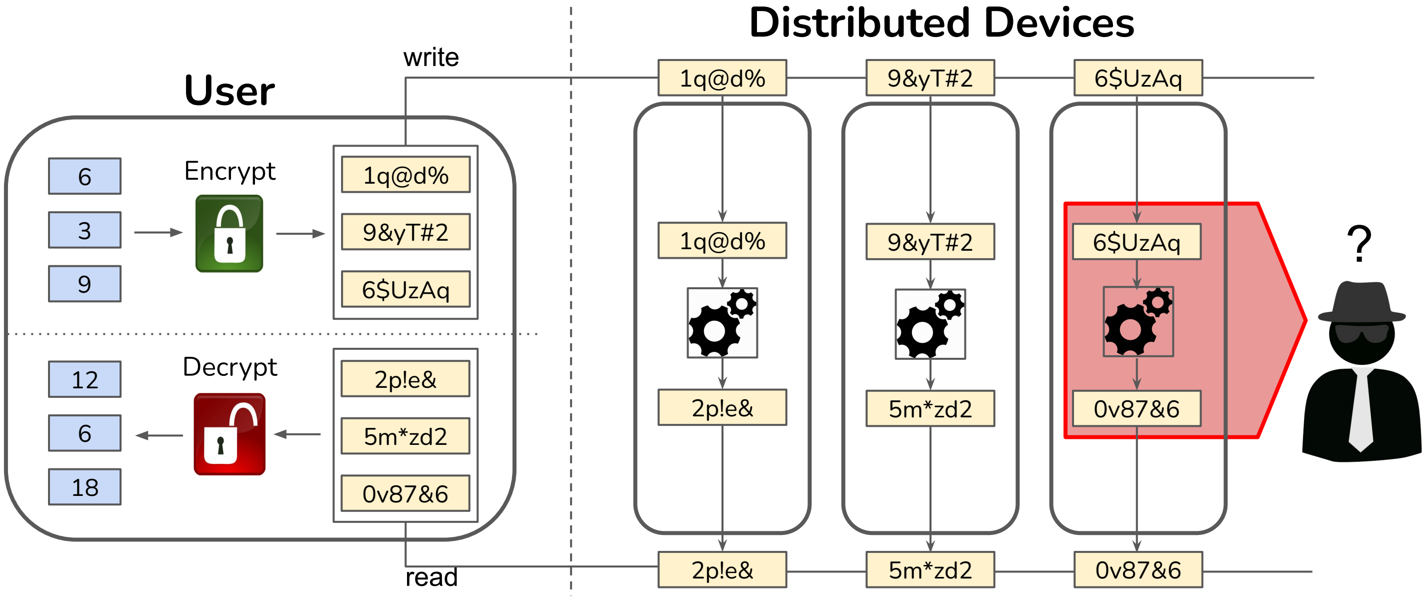
Encrypt(m) = 2 * m   Decrypt(c) = c / 2



- Partially (add. *or* mult.) or Fully (add. *and* mult.) homomorphic

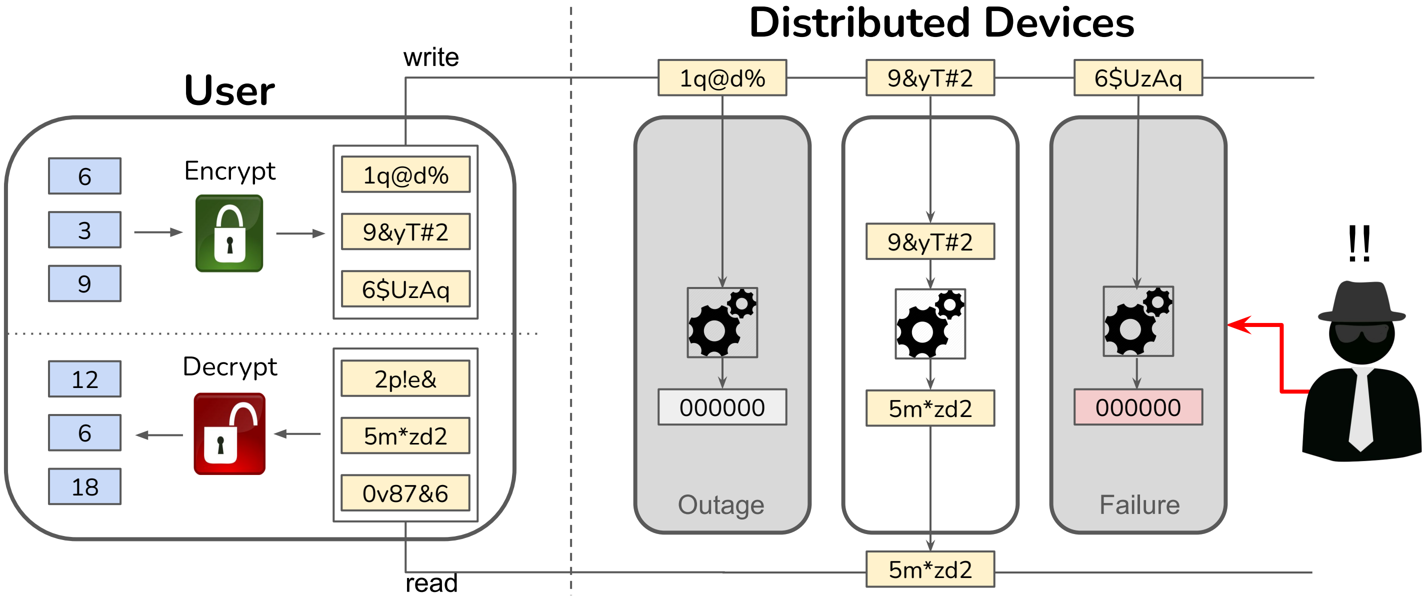
Computing on the Cloud

- With support for HE, computations can take place without ever revealing the secrets



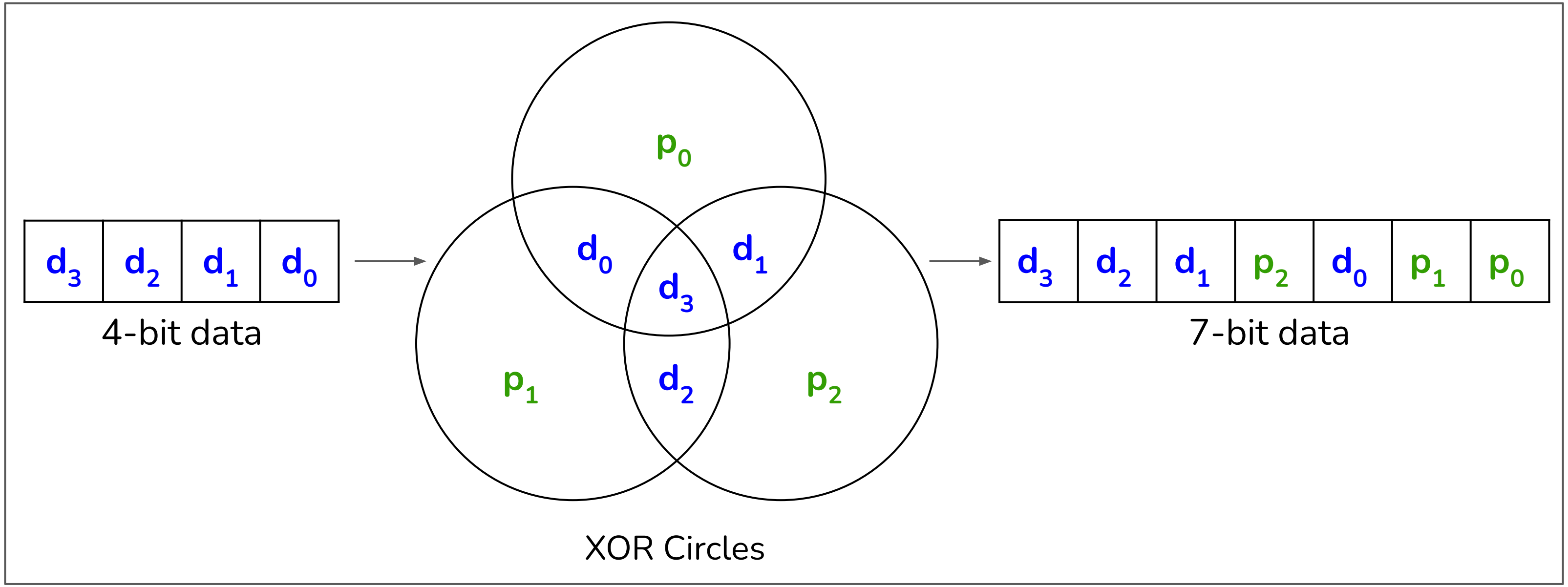
Challenge: Making HE ciphertexts resilient

- How can we handle data losses in the cloud?



Common approach: Erasure codes

- Compute erasure codes over plaintext to enable recovery



- Distribute redundancy codewords over servers

The problem with erasure codes and HE ciphertexts

How can it be applied to ciphertexts?

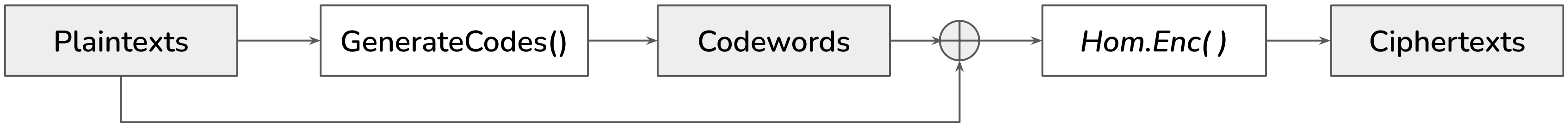
- If trivially applied, would be computed on the ciphertext level

- Problem:
  - Storage impact of codewords is proportional to input
  - HE ciphertexts can be LARGE → Ciphertexts can be large polynomials (Ring-LWE based FHE)
    - ~6000% size increase from plaintext

Alternative: Codewords first, then encrypt

A version *Encrypting-with-Redundancy*

- Not well studied for Homomorphic Encryption



- Problem:
  - Assume the ciphertexts are not operated on, or only supported for partial homomorphic operations
  - Not applicable for fully homomorphic schemes
  - Not applicable for variety of complex cloud computing operations

To bridge this gap: X-Cipher

- Enables recovery of fully homomorphic ciphertexts without decryption
- Leverages encoding and packing techniques for optimized storage
- Maintains privacy and recoverability across fully-homomorphic computations

Fully Homomorphic Encryption (FHE)

- Enables additions and multiplications without decryption

- This work uses schemes based on Ring Learning-with-Errors (Ring-LWE)
  - Elements are based on polynomial ring $R_q = Z_q[x]/\Phi[x]$
  - Plaintext values are encoded into polynomials

Example:

$$A \rightarrow A(x) = a_0 + a_1x + a_2x^2 + \ldots + a_nx^n$$

- Compared to standard encryption, FHE has large storage requirement

Optimized polynomial encoding: Ciphertext Packing

Subfield packing: Packing values into subfields using Chinese Remainder Theorem (CRT)

Polynomial modulus: $\Phi(x) = x^4 + 1 = (x - 2)(x - 2^3)(x - 2^5)(x - 2^7) \pmod{17}$

Each vector element corresponds to a 0 degree polynomial:

v = [8, 5, 16, 9]

$1 + x + 7x^2 + 12x^3 \equiv \mathbf{8} \mod (17, x-2)$

$1 + x + 7x^2 + 12x^3 \equiv \mathbf{5} \mod (17, x-2^3)$

$1 + x + 7x^2 + 12x^3 \equiv \mathbf{16} \mod (17, x-2^5)$

$1 + x + 7x^2 + 12x^3 \equiv \mathbf{9} \mod (17, x-2^7)$

Utilized in CKKS and **BGV** — X-Cipher leverages BGV scheme

Enables SIMD-like homomorphic operations

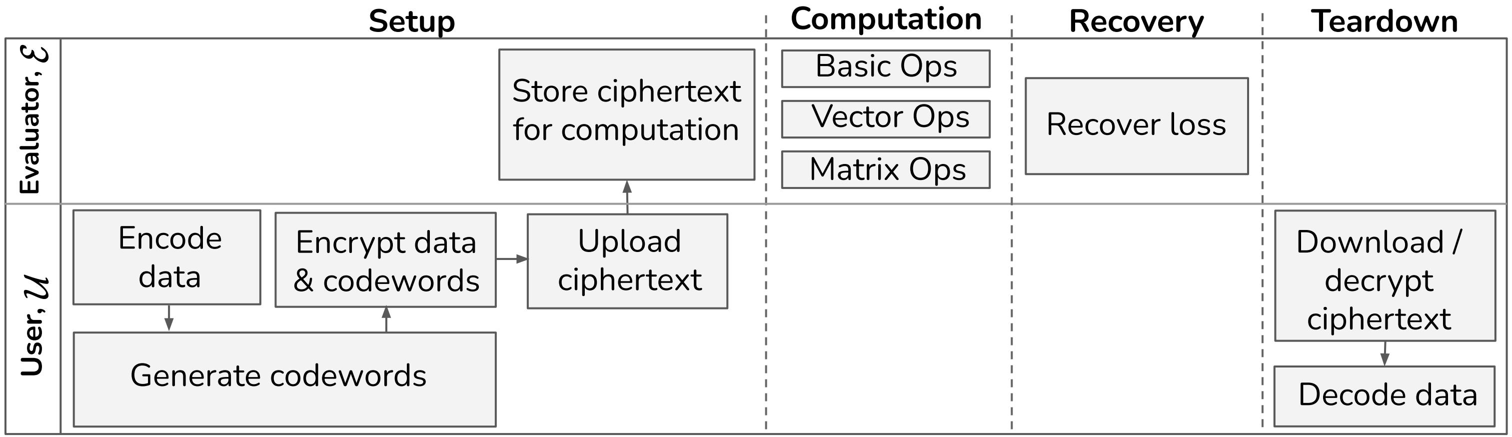
X-Cipher key ideas:

- Generate code words for plaintext vectors and pack them alongside each other for optimized storage and recovery capability

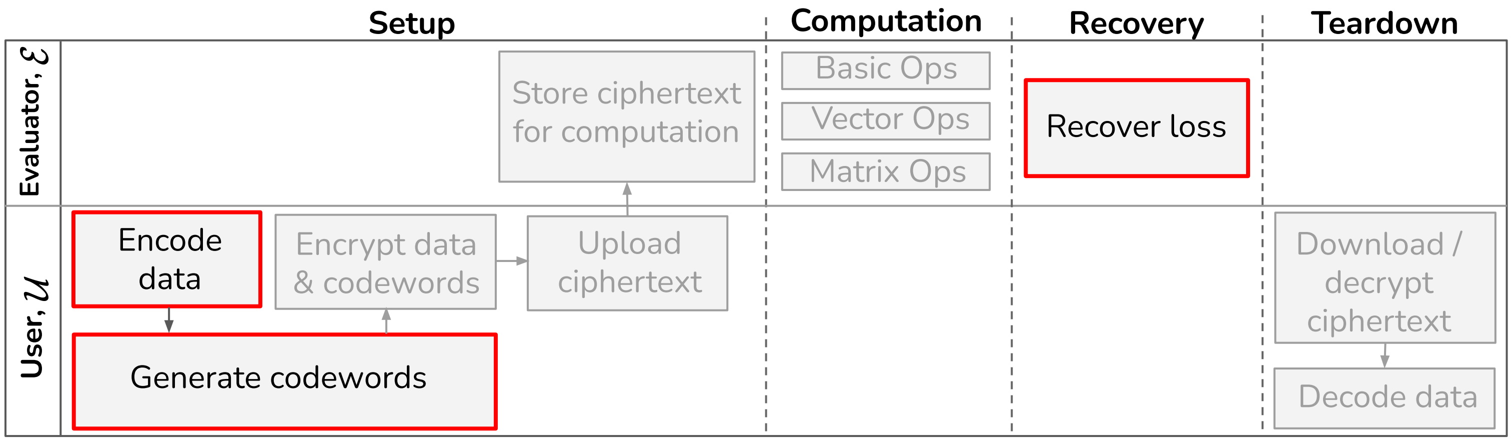
- Provide homomorphic recovery algorithms to enable data recovery without requiring decryption

- Enable computations over homomorphic ciphertexts that maintain the ability to recover intermediate or final results

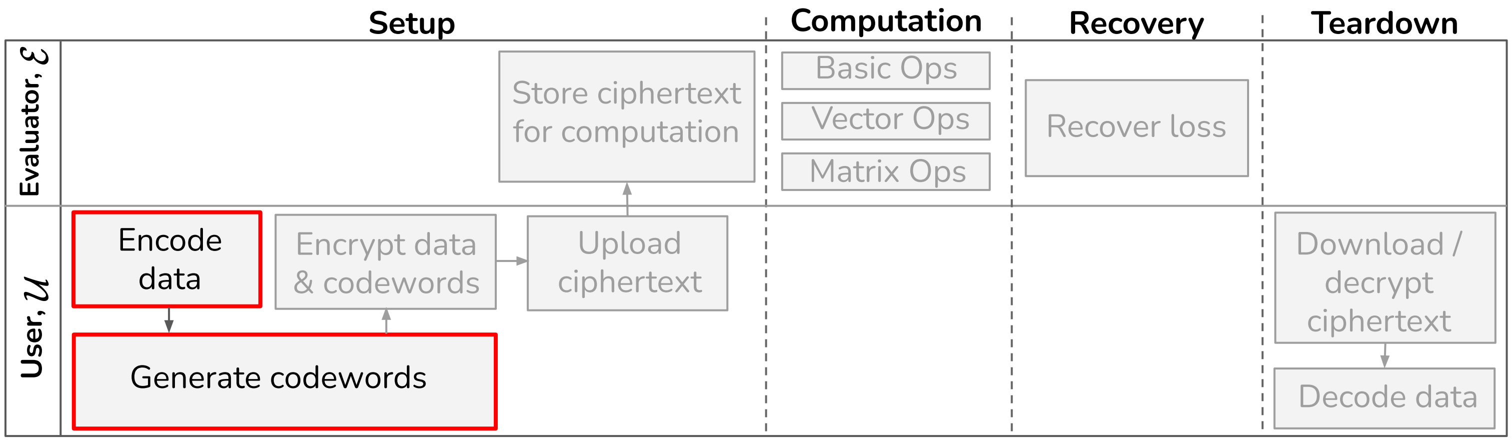
X-Cipher Workflow



X-Cipher Workflow

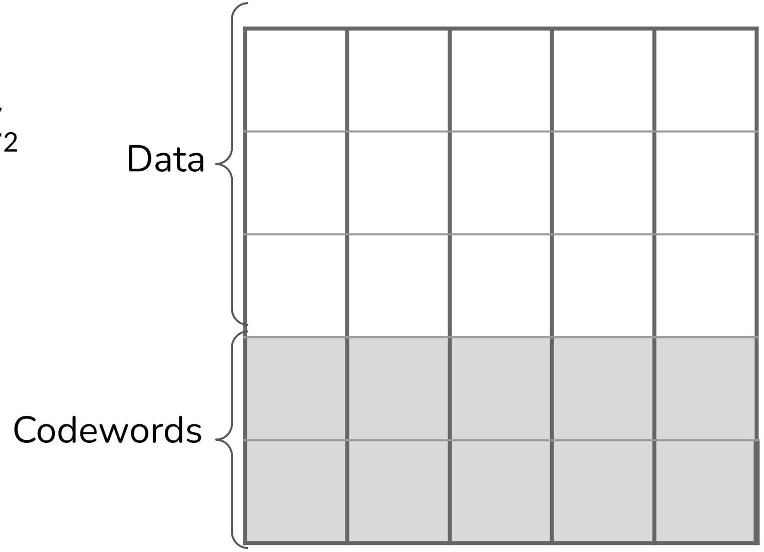


X-Cipher Workflow



X-Code for codewords

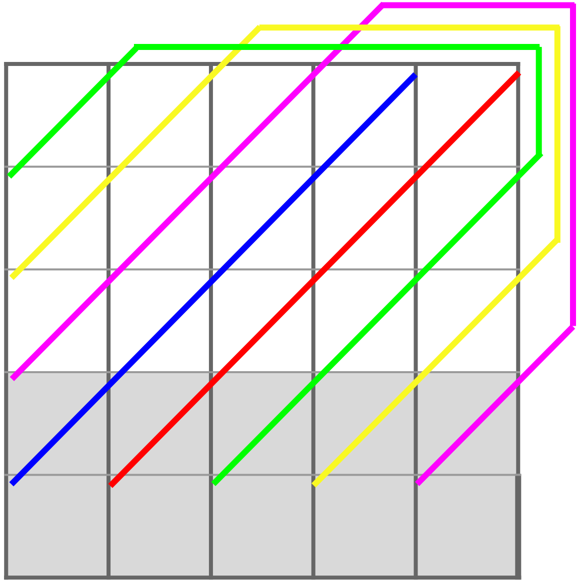
- Utilize X-Code erasure codes
  - Dual parity → two column recovery
  - Only requires xor operations → Addition in Z_2
- Construct an *n x n* grid
  - First *n-2* rows are **data**
  - Last 2 rows with **codewords**



Example: n=5

How to generate codewords?

- First row of codewords is computed by addition along slope +1 diagonals

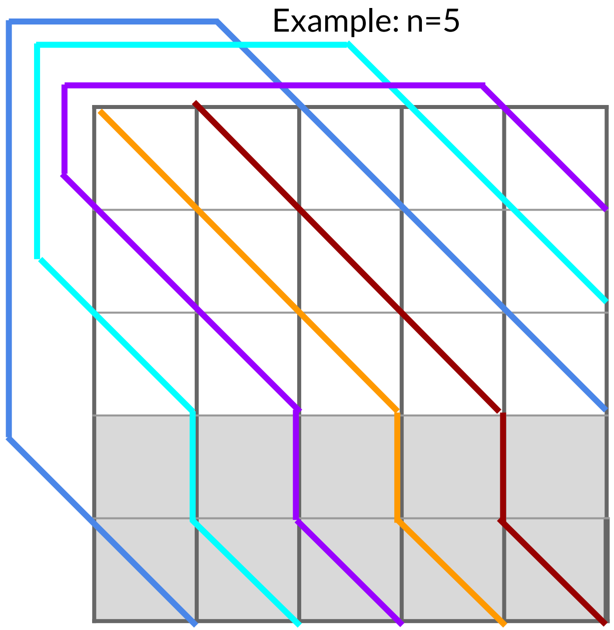


Example: n=5

How to generate codewords

- First row of codewords is computed by addition along +1 slope diagonals

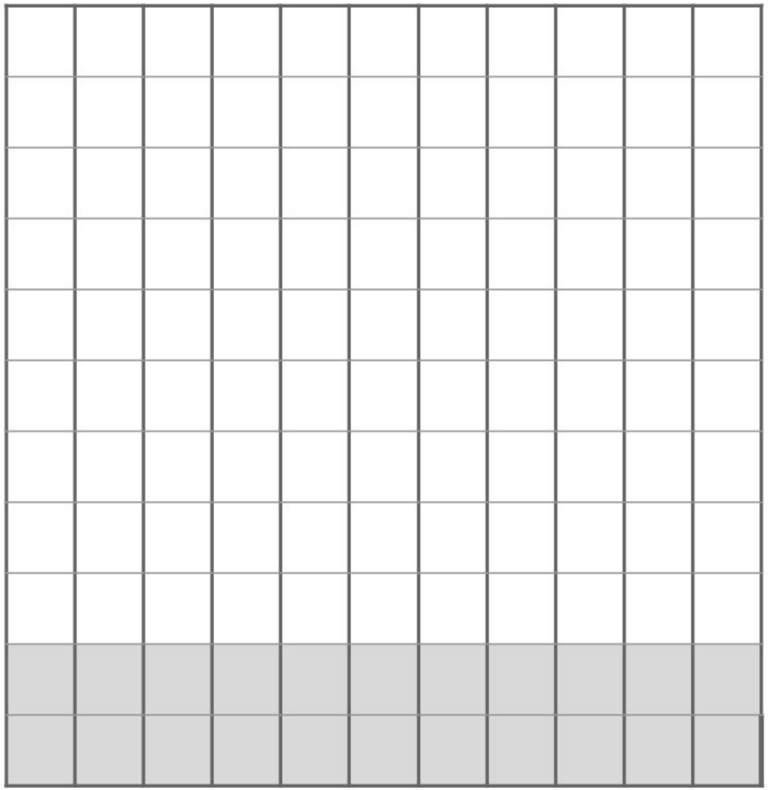
- Second row of codewords is computed by addition along -1 slope diagonals



X-Code structure for large data

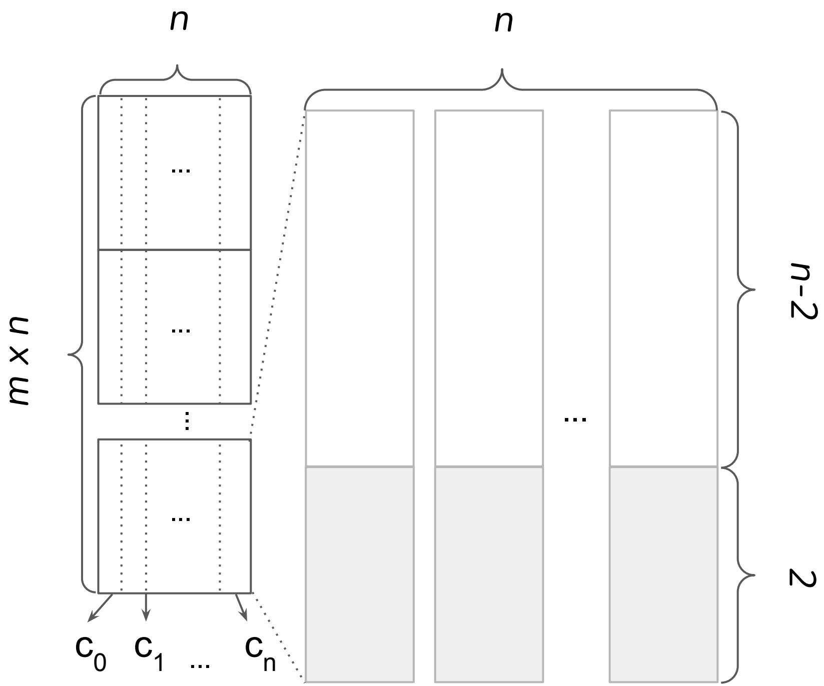
- Problem: proportion of recoverable data decreases as grid becomes larger

- Additionally, might assume we have more available servers

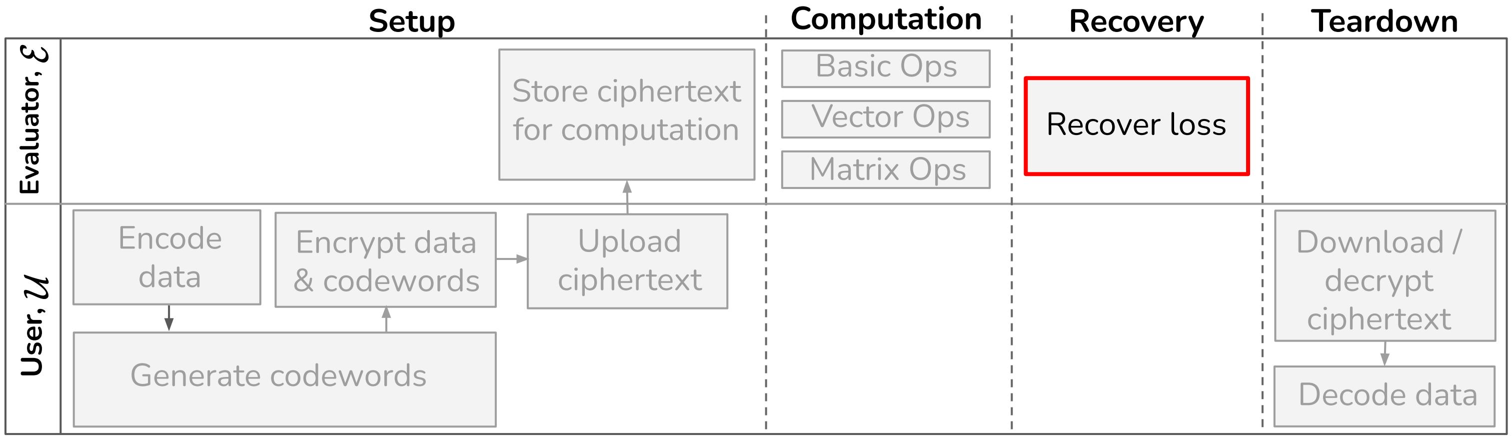


X-Cipher Structure

- Stack *m* of the *n x n* grids vertically
- Use encoding and ciphertext packing to encrypt each column into a single ciphertext
- Perform operations on each column
- Maximizes the “utilization” of ciphertext packing

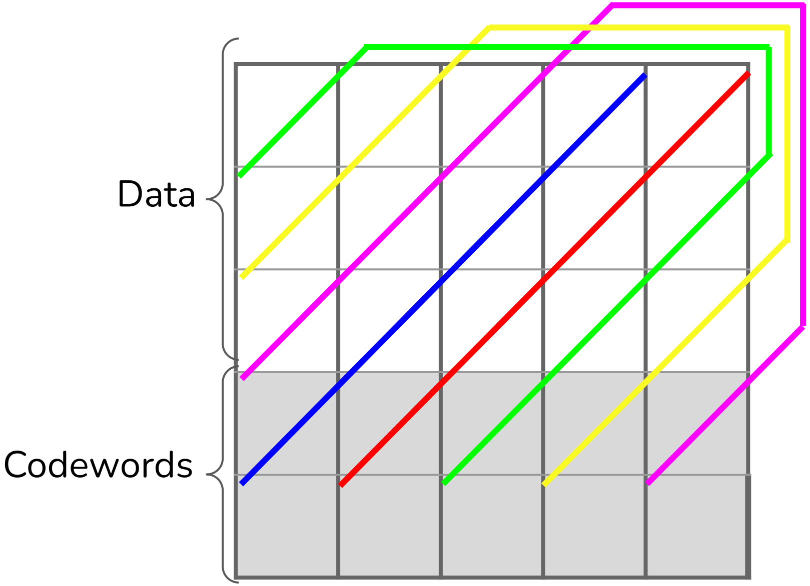


X-Cipher Workflow



Recovery follows algorithm of X-Code

- Summing along the diagonals to recover data



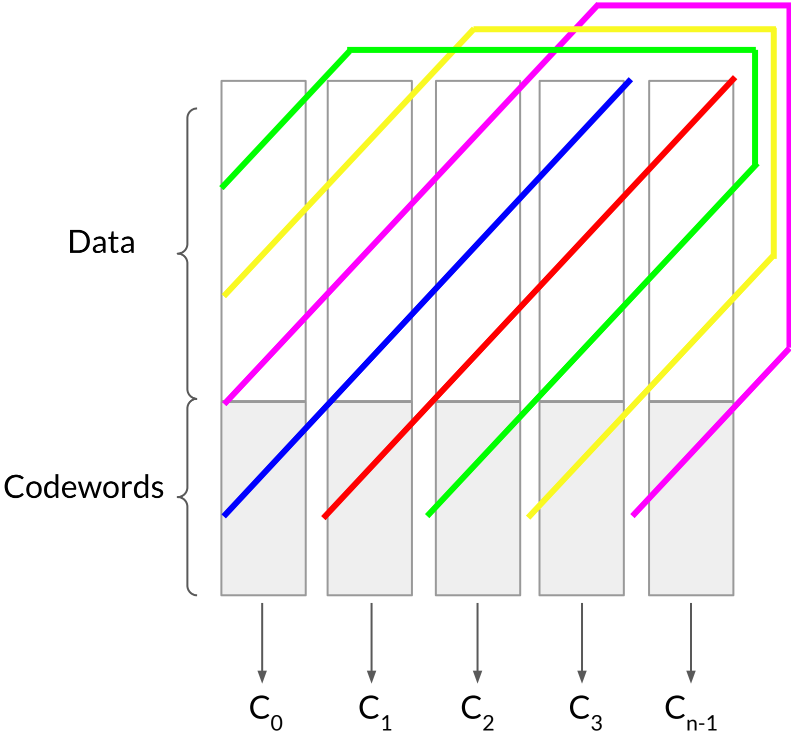
Example: n=5

Recovery follows algorithm of X-Code

- Summing along the diagonals to recover data

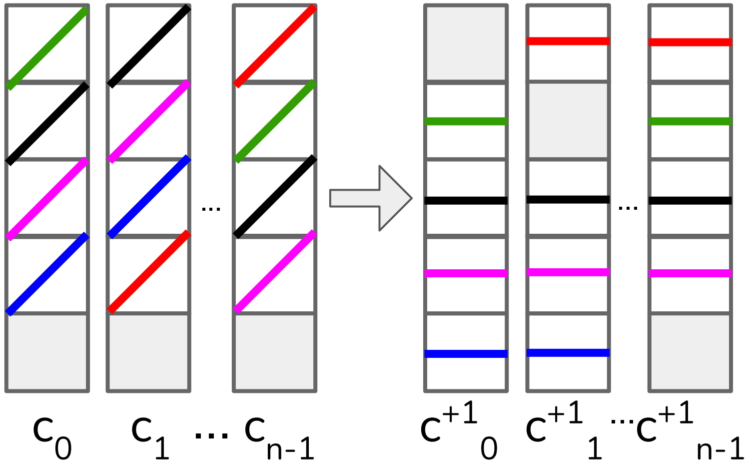
- However, ciphertexts correspond to a single column

- We must *rotate* the columns



Ciphertext Rotation

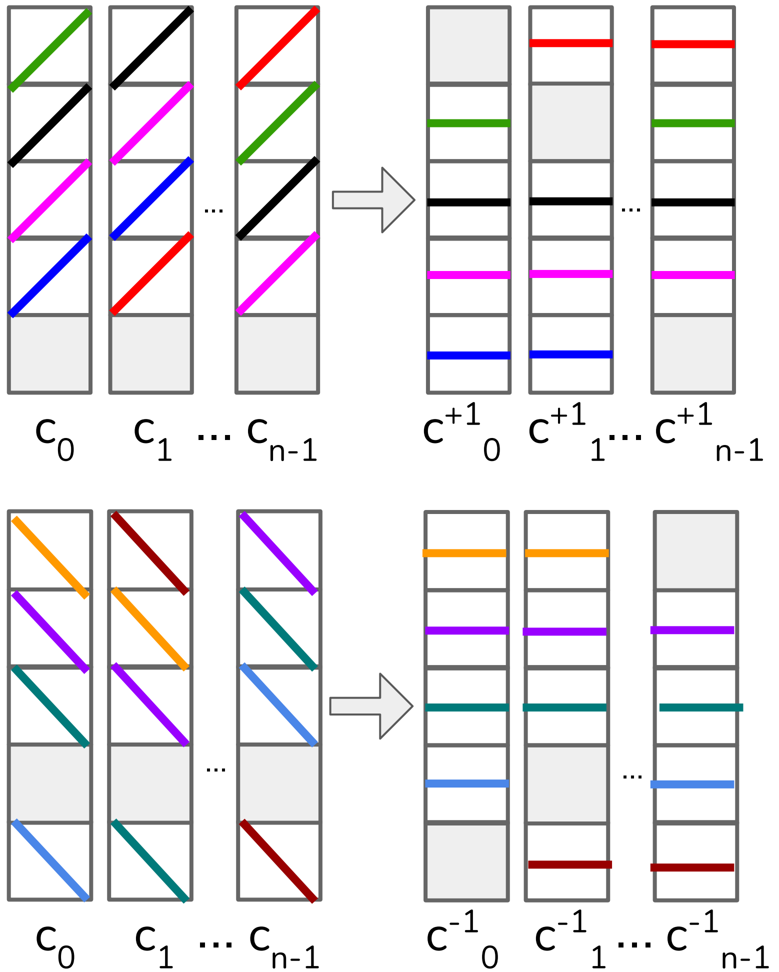
- For the +1 sloped diagonals, the column c_i must be rotated by *n-i-1*



Ciphertext Rotation

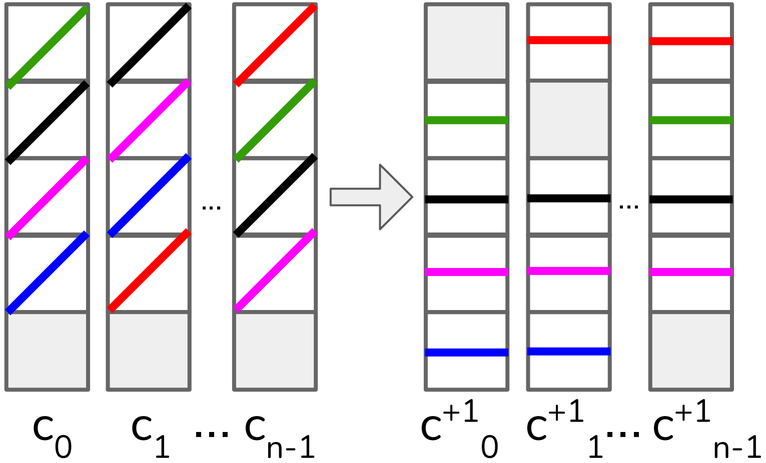
- For the +1 sloped diagonals, the column c_i must be rotated by $n\text{-}i\text{-}1$

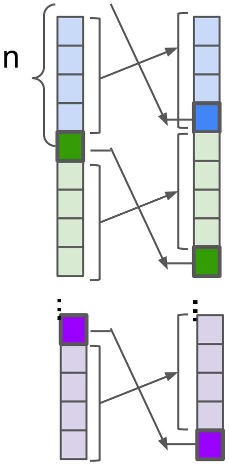
- For the -1 sloped diagonals, the column c_i must be rotated by i



Ciphertext Rotation

- For the +1 sloped diagonals, the column c_i must be rotated by *n-i-1*
- For the -1 sloped diagonals, the column c_i must be rotated by *i*
- Recall: X-Cipher structure is multiple square grids

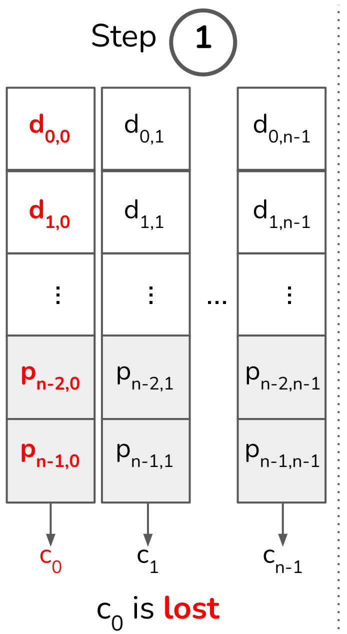




X-Cipher rotation algorithm applies to all internal square grids simultaneously

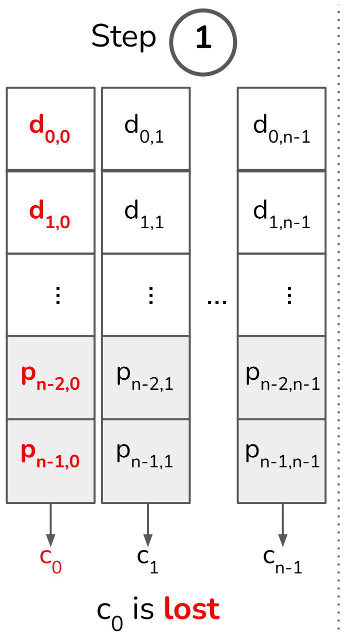
Rotation for one column recovery

Consider a case in which Adversary has caused failure in Server 0 (c_0 is lost)



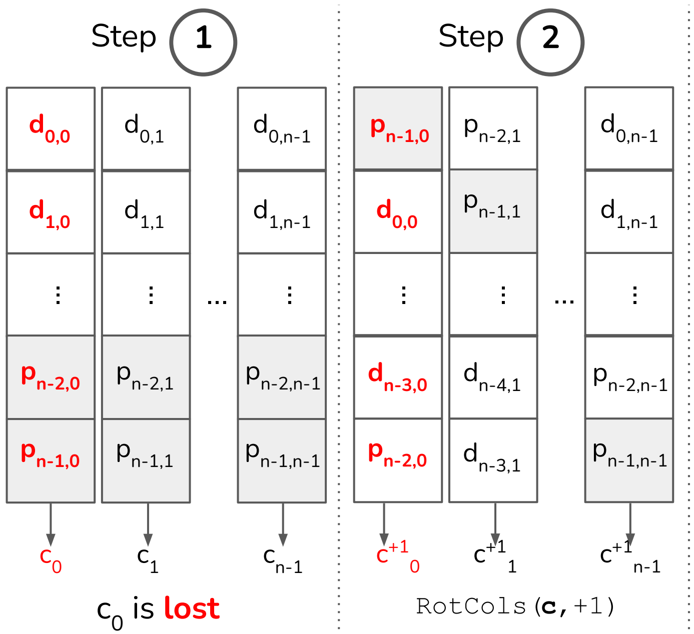
Rotation for one column recovery

First we, detect c_0 is lost due to a lack of response from Server 0



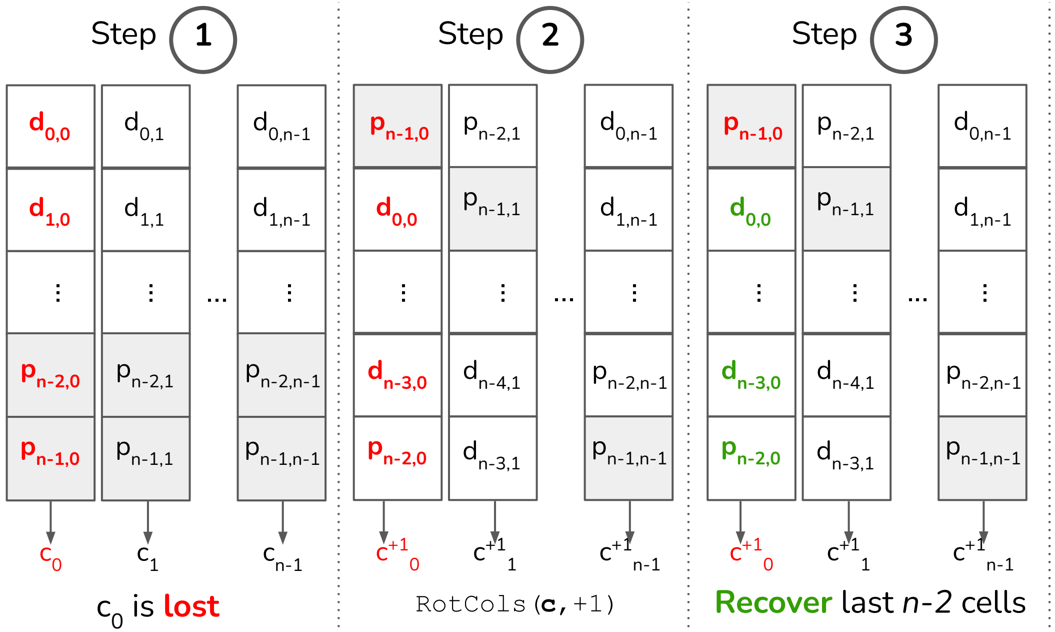
Rotation for one column recovery

Second, we begin rotating and summing to recover via +1 sloped codewords



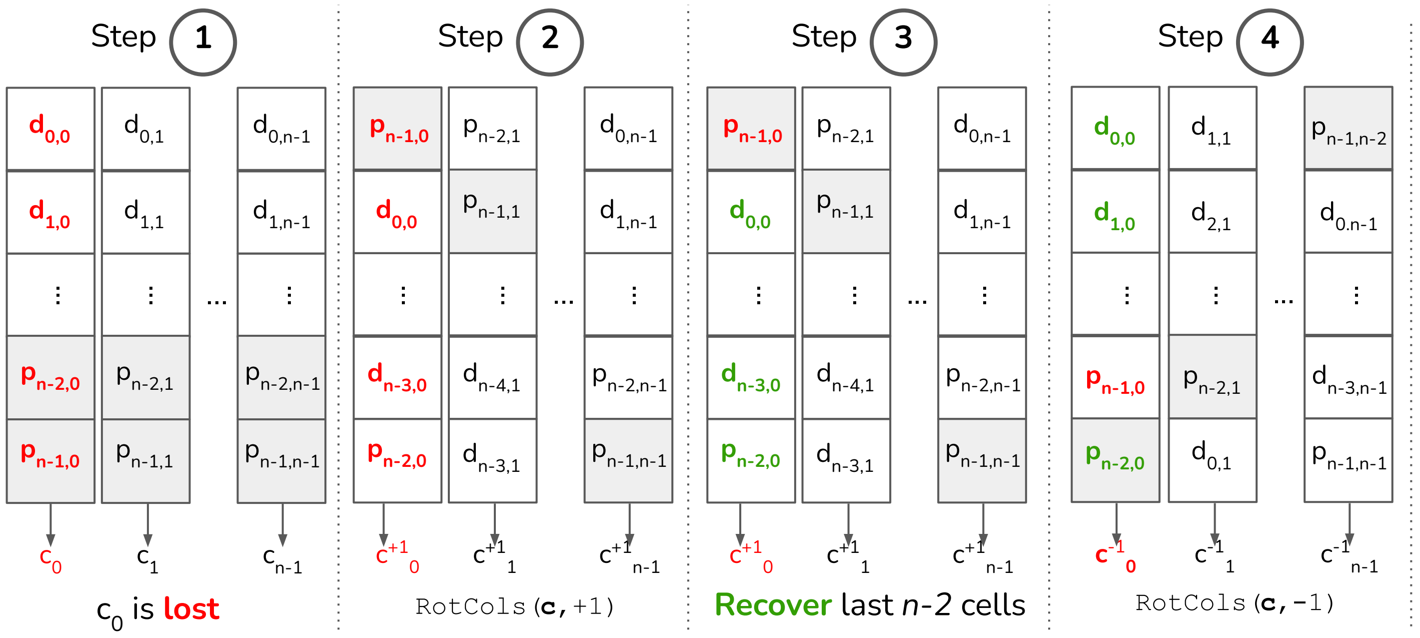
Rotation for one column recovery

We **recover** all data within c_0 except for the final codeword $p_{n-1,0}$



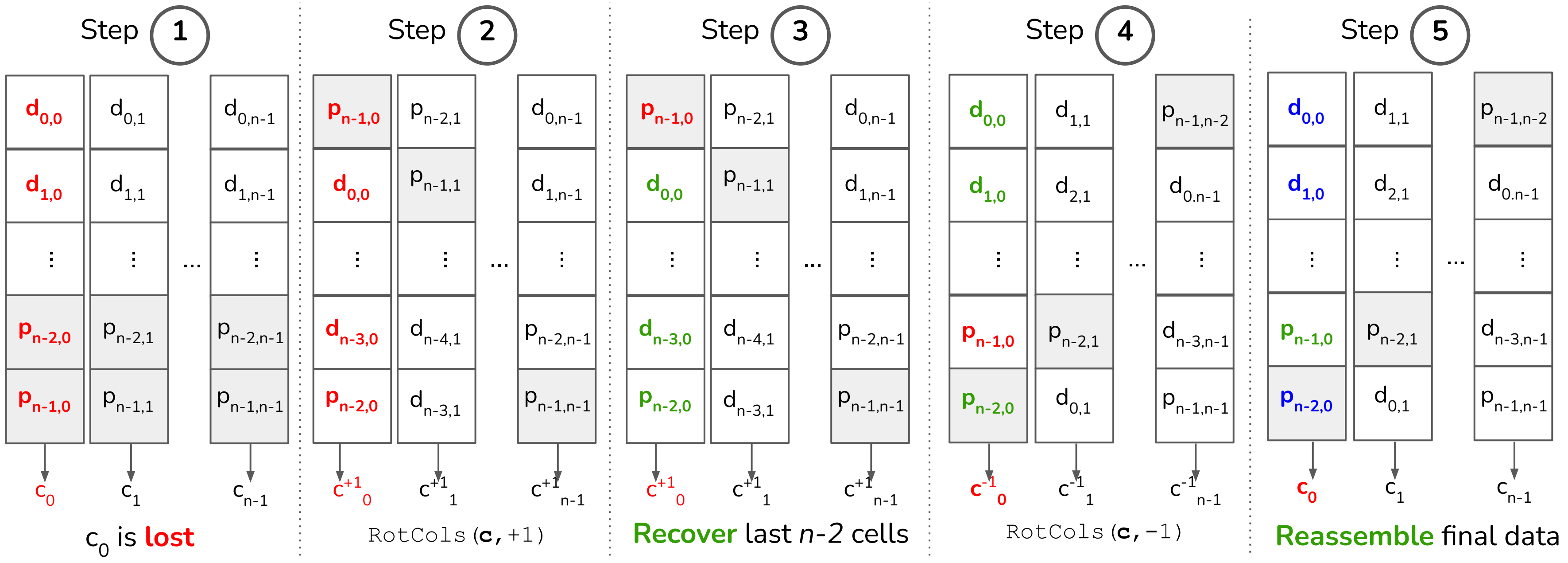
Rotation for one column recovery

Perform another rotation and summing to recover the final data



Rotation for one column recovery

c_0 has now been fully recovered. It can be redistributed to continue operations



Other features (see paper for details!)

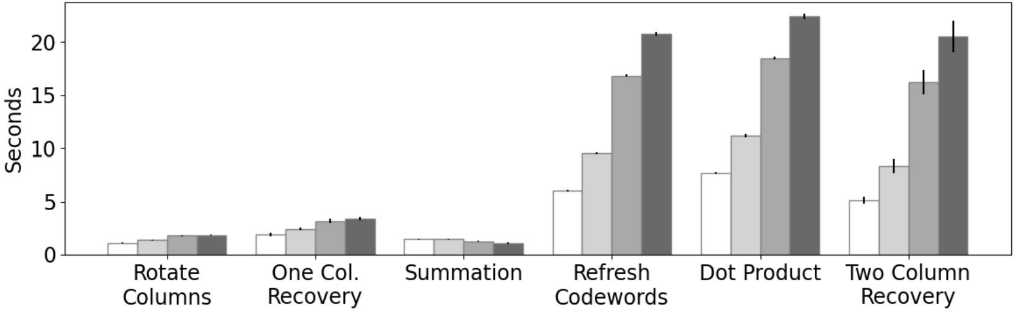
- Two column recovery → multiple iterations of the one-column recovery

- Basic operations / Primitive operations:
    - Homomorphic arithmetic, refresh codewords, dot-product, summation

- Demonstrate construction of complex algorithms:
    - Private set intersection (PSI)
    - Matrix multiplication

Results

- Evaluated using CloudLab
- Primitive function timing
- Ciphertext size impact



Parameters			Size (KB)		
Dimension (n)	Multiples (m)	Data Cells	Plaintext	Ciphertext (X-Cipher)	Ciphertext (without)
5	7	180	0.72	0.93	55.8
7	9	315	1.26	1.30	82.1
11	5	495	1.98	2.05	112.5
13	4	572	2.28	2.42	125.7

Thank you!

Information:

Paper (via ICICS):

Code (via Github):

Contact:

My email (ac7717@rit.edu)