Fun with Linearity:

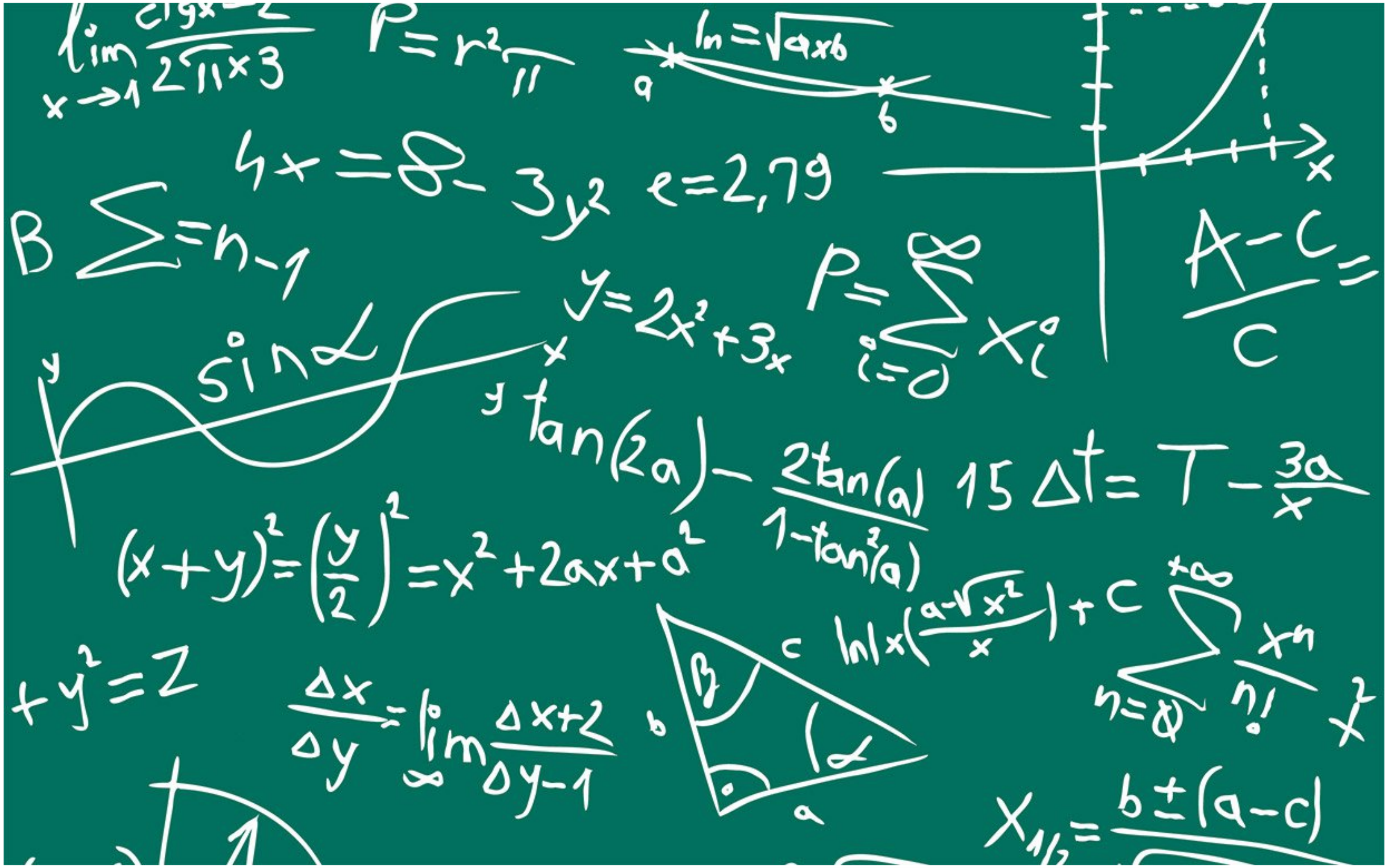
How encryption and erasure codes are intimately related

Jason Resch
IBM

Presentation Overview

- Linear Functions
- Combining Linear Functions
- Linearity of Erasure Codes
- Erasure Codes and Encryption
- Erasure Codes and Integrity Checks
- Exploiting Linearity for New Applications

Linear Functions



Linear Functions Defined

- A function ***F*** is "*linear*" if it satisfies the following:
  - **F**(x) + **F**(y) = **F**(x + y)
- In short, a function is linear if an operation applied to the inputs yields the same result as performing that operation on the outputs.

Examples of Linear Functions

- Consider the function **MulFive**(X)
  - It multiplies any input by 5
  - Is this a linear function?
- Let’s consider two inputs, X=3 and Y=7:
  - MulFive(3) = 15
  - MulFive(7) = 35
  - MulFive(3 + 7) = MulFive(3) + MulFive(7) ✓

CRC as a Linear Function

- Many functions have this property, among them:
  - The Cyclic Redundancy Check (CRC)
  - CRC is commonly used for integrity checks

- CRC is a “polynomial division” in a finite field
  - Addition in this field is “$\oplus$” rather than “+”

Consequences of CRC's Linearity

- Given the definition of linearity, it follows that:
  - $CRC(x) \oplus CRC(y) = CRC(x \oplus y)$
- In actual implementations of CRC, there is a caveat, we must also add CRC(zeros)
  - Where 'zeros' is a binary string consisting of all zeros and is equal in length to 'x' and 'y'
  - $CRC(x) \oplus CRC(y) \oplus CRC(zeros) = CRC(x \oplus y \oplus zeros)$

Example of zlib CRC-32's linearity

```python
#!/usr/bin/python

from zlib import crc32

def xor(x, y):
  return ''.join(chr(ord(a) ^ ord(b)) for a,b in zip(x,y))

def calc_combined_crc(x, y):
  zeros = chr(0) * len(x)
  return crc32(x) ^ crc32(y) ^ crc32(zeros)

print calc_combined_crc('foo', 'bar')

foobar = xor('foo', 'bar')
print crc32(foobar)
```

source — bash — 85×29

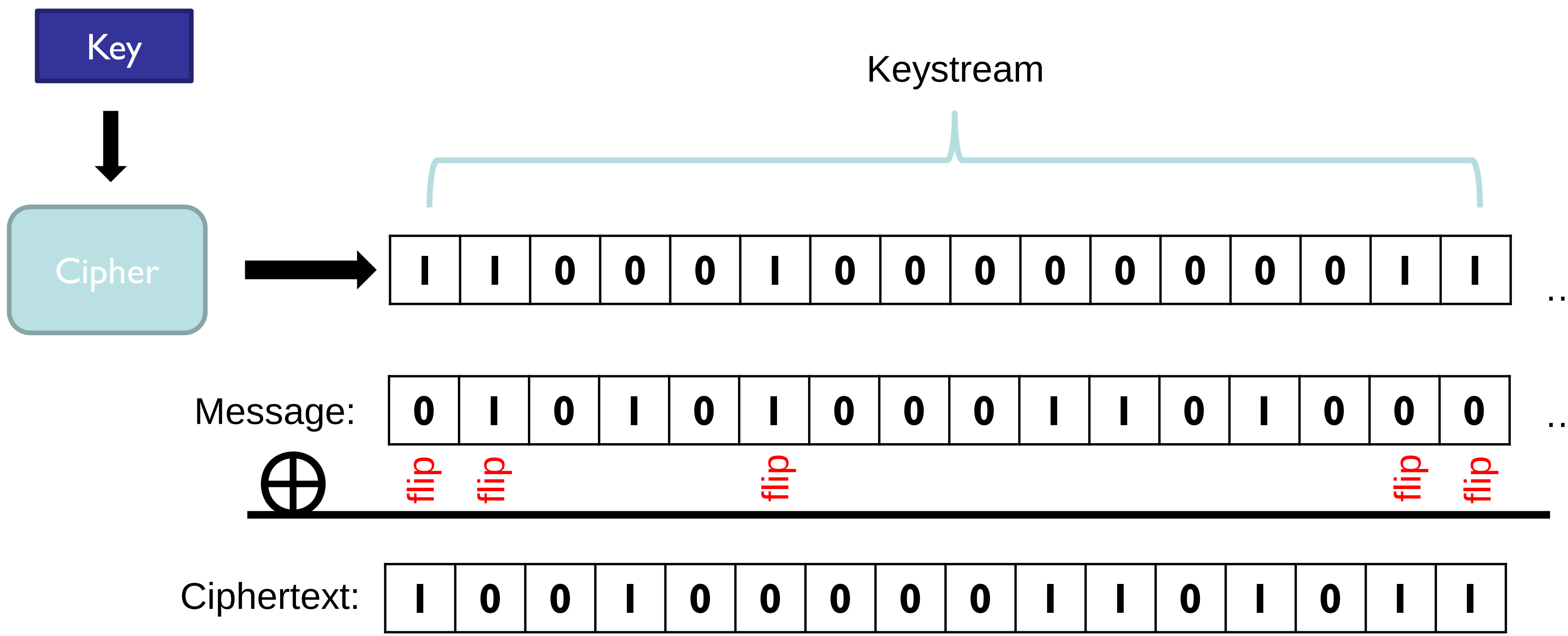
```
MacBook-Pro-L40:source jresch$ ./crc.py
97333401
97333401
MacBook-Pro-L40:source jresch$
```

Combining Linear Functions

Encryption Functions

- Encryption functions, in their most basic form, are simply random number generators
  - Key acts as a “seed” to generate an arbitrarily long set of random output–the “keystream”
  - Keystream is XORed with Plaintext to encrypt

Encryption Visualized



Encryption and CRC

- The XOR operation used in encryption is the same as addition in the finite field of CRC…
  - Ciphertext = Plaintext $\oplus$ Keystream

- Since the Ciphertext is equal to the Plaintext added to the Keystream it follows that:
  - CRC(Ciphertext) = CRC(Plaintext) $\oplus$ CRC(Keystream)

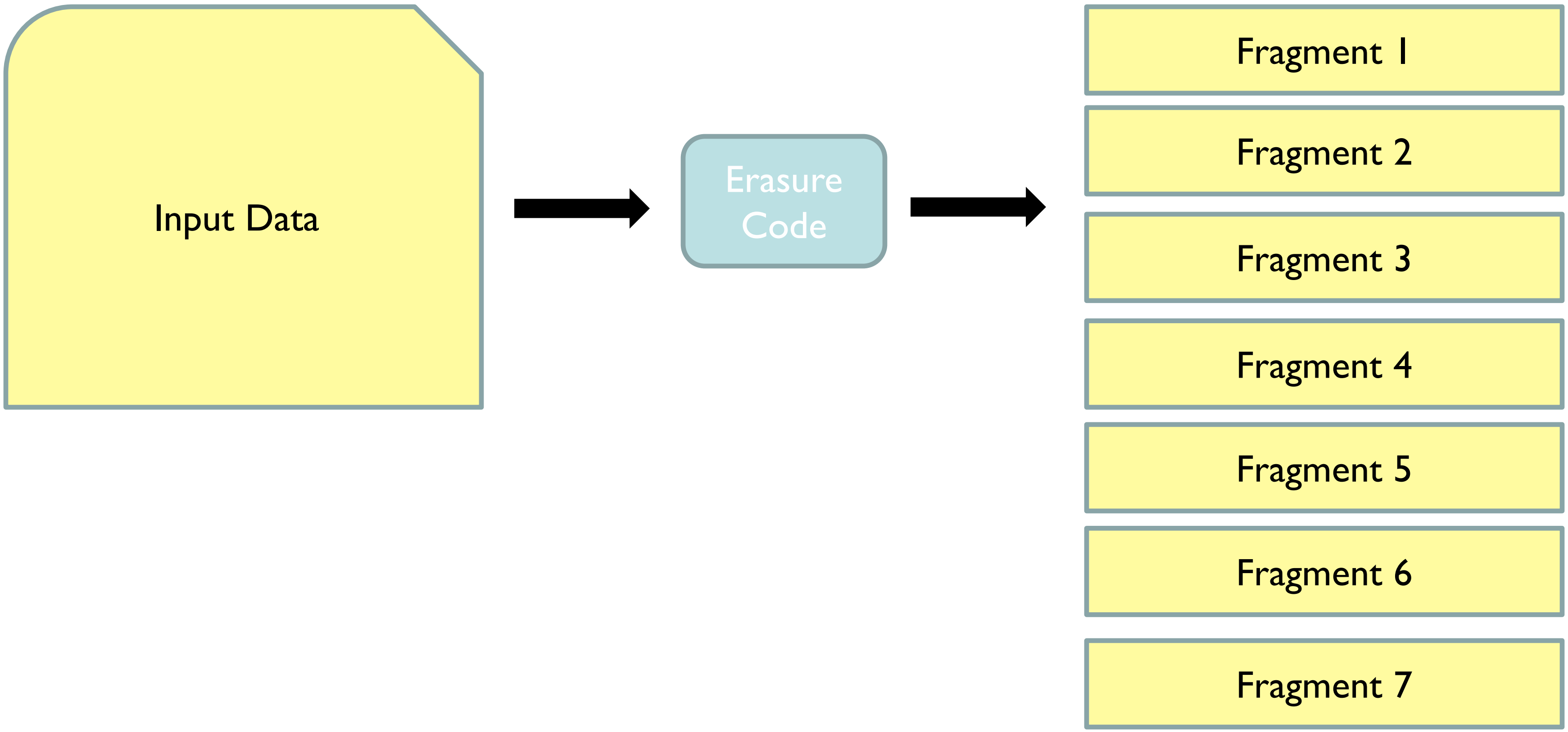
Linearity of Erasure Codes

Erasure Codes

- Erasure Codes encode ***T*** inputs into ***N*** outputs
- Can recover original input from any ***T*** of outputs

- Examples:
  - RAID 5, RAID 6, Reed-Solomon, Rabin's Information Dispersal, Shamir Secret Sharing

Visualizing Erasure Codes



Applications of Erasure Codes

- Erasure Codes are often used to achieve durability and availability in storage systems
  - Files split into ***T*** fragments, redundancy expands these into ***N*** fragments
  - Each fragment stored to different node/drive
    - Reliable: Data can survive (***N*** – ***T***) faults
    - Efficient: Overhead equal to (***N*** / ***T***)

Erasure Codes are Linear

- Like CRC and Encryption, Erasure Codes operate within a finite field
- Encoding and decoding are implemented via addition and multiplication in a field

- Erasure Codes are linear:
  - $EC(x) \oplus EC(y) = EC(x \oplus y)$

SDC 16

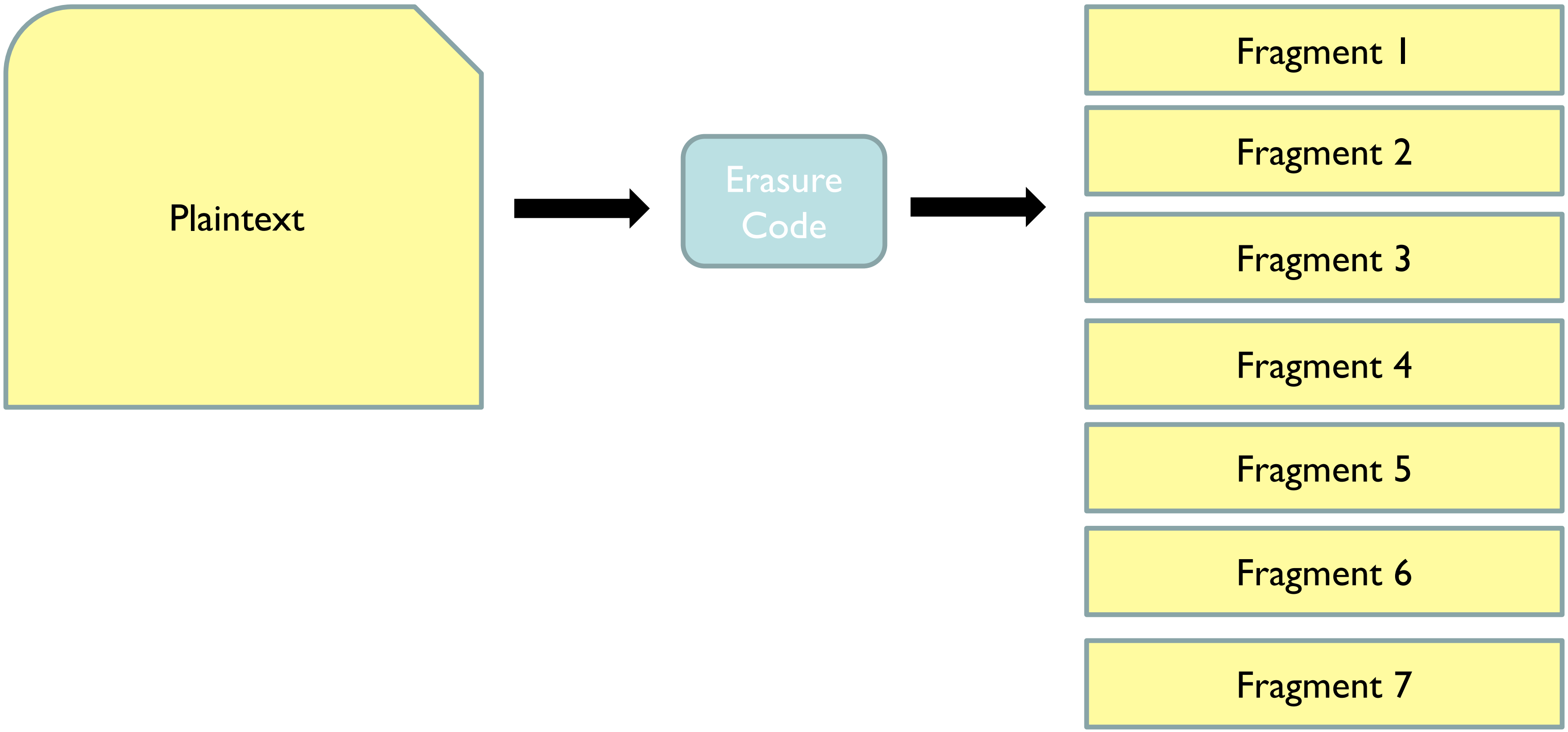
Erasure Codes and Encryption

Erasure Codes and Encryption

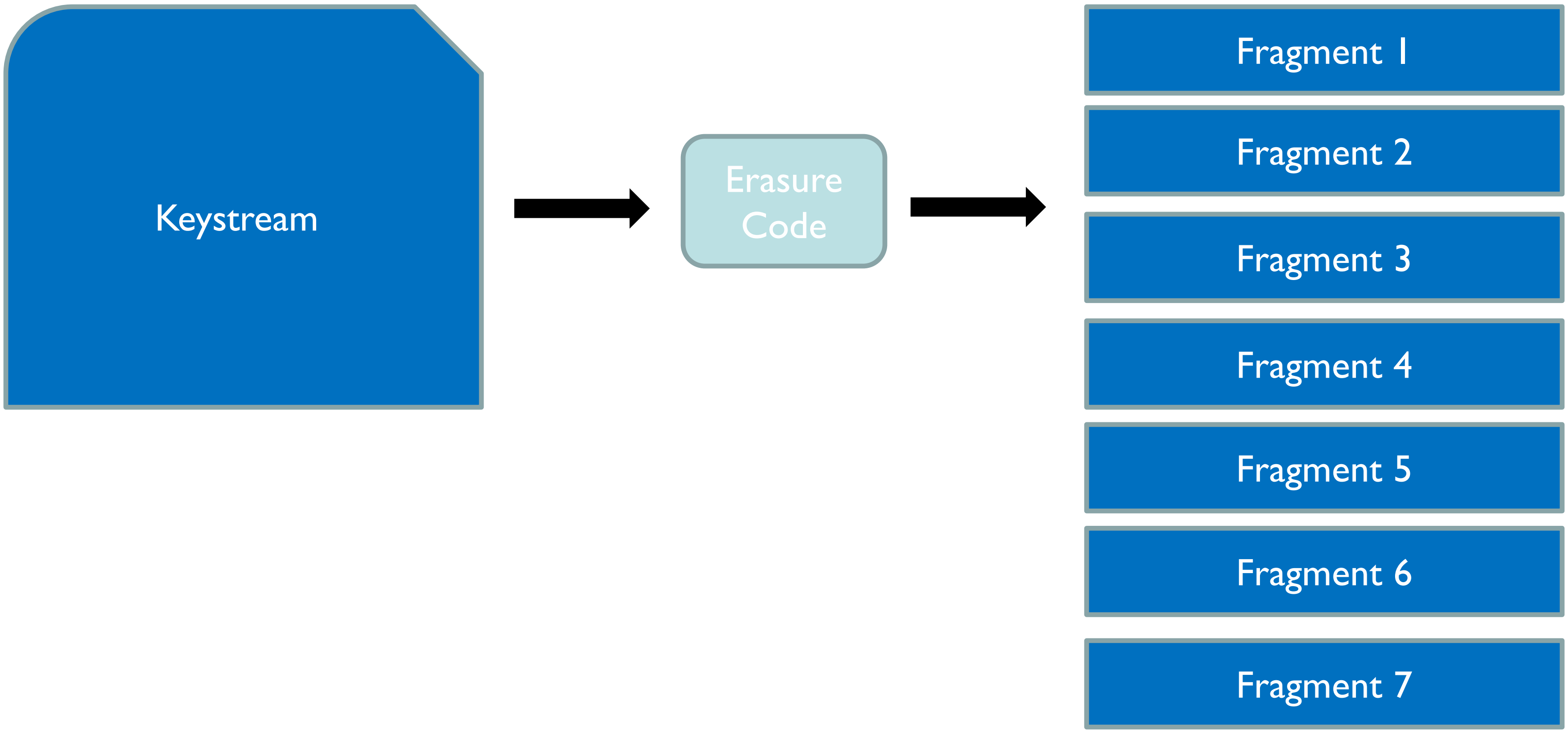
- Since encryption is addition, and because Erasure Codes are linear functions:
  - $EC(Ciphertext) = EC(Plaintext) \oplus EC(Keystream)$

- If we erasure code encrypted data, we will get output fragments that will be identical to if we erasure code the plaintext, and add those fragments to the fragments resulting from erasure coding the keystream.

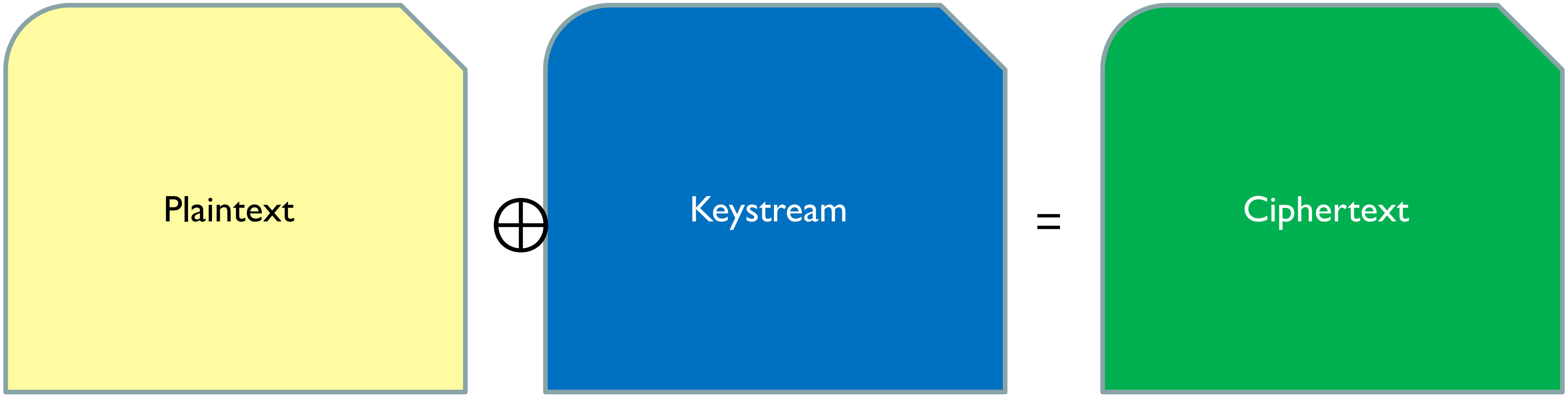
Erasure Coding Plaintext



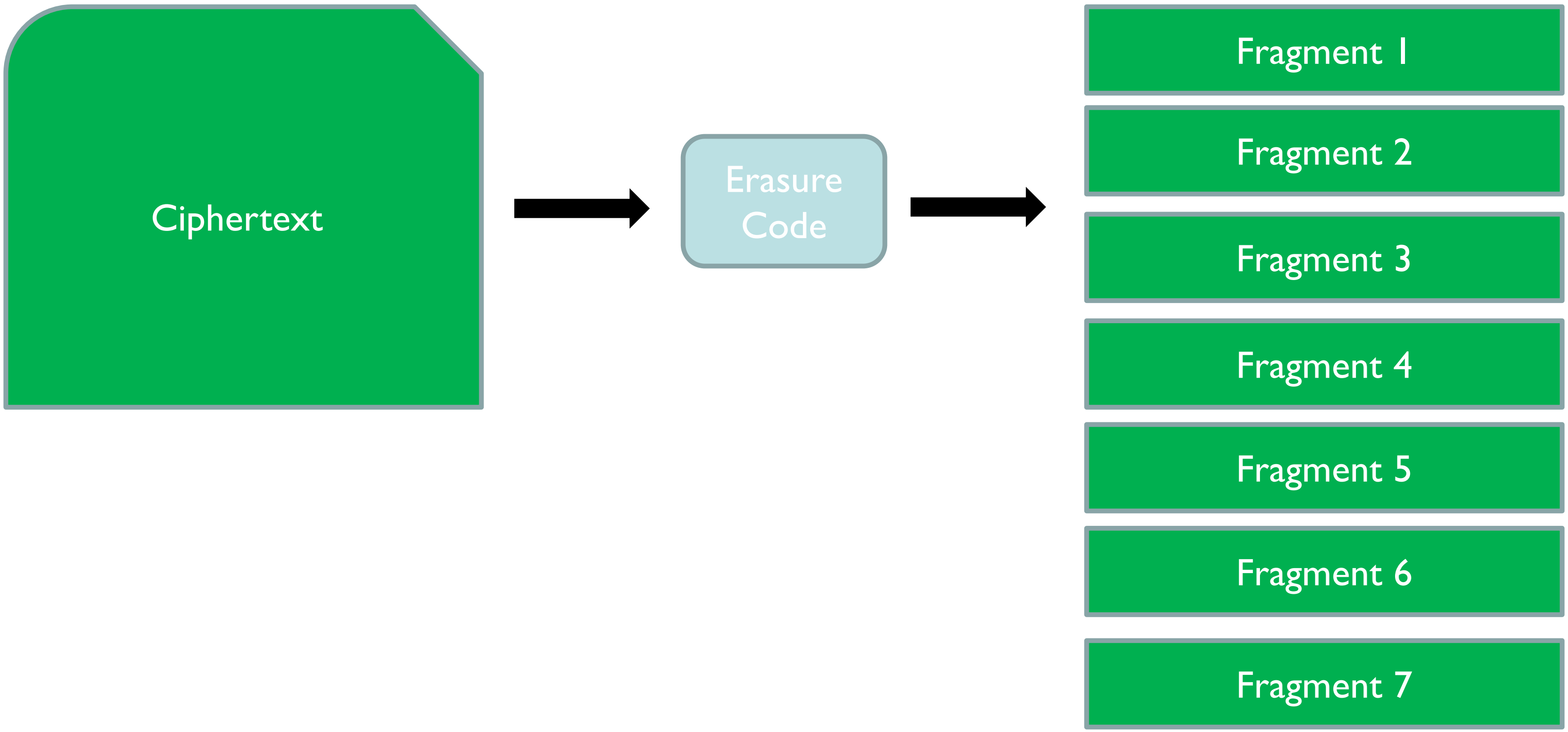
Erasure Coding Keystream



Relating the Erasure Code Inputs



Erasure Coding Ciphertext



Relating the Erasure Code Outputs

Plaintext Fragments		Keystream Fragments		Ciphertext Fragments
Fragment 1	$\oplus$	Fragment 1	=	Fragment 1
Fragment 2	$\oplus$	Fragment 2	=	Fragment 2
Fragment 3	$\oplus$	Fragment 3	=	Fragment 3
Fragment 4	$\oplus$	Fragment 4	=	Fragment 4
Fragment 5	$\oplus$	Fragment 5	=	Fragment 5
Fragment 6	$\oplus$	Fragment 6	=	Fragment 6
Fragment 7	$\oplus$	Fragment 7	=	Fragment 7

Using this property in practice

- In a distributed erasure coded system, we can:
  - Encrypt previously unencrypted fragments
  - Decrypt previously encrypted fragments
  - Rekey encrypted fragments

- But we can do so without any network transfer!
- We only need to send keys to storage nodes

Illustrating Rekeying of Fragments

What's currently stored		Fragment of Plaintext		Fragment of Keystream_1
$EC(\text{Plaintext} \oplus \text{Keystream}_1)$	=	$EC(\text{Plaintext})$	$\oplus$	$EC(\text{Keystream}_1)$

What we want stored		Fragment of Plaintext		Fragment of Keystream_2
$EC(\text{Plaintext} \oplus \text{Keystream}_2)$	=	$EC(\text{Plaintext})$	$\oplus$	$EC(\text{Keystream}_2)$

What we want stored		What's currently stored		Fragment of Keystreams
$EC(\text{Plaintext} \oplus \text{Keystream}_2)$	=	$EC(\text{Plaintext} \oplus \text{Keystream}_1)$	$\oplus$	$EC(\text{Keystream}_1 \oplus \text{Keystream}_2)$

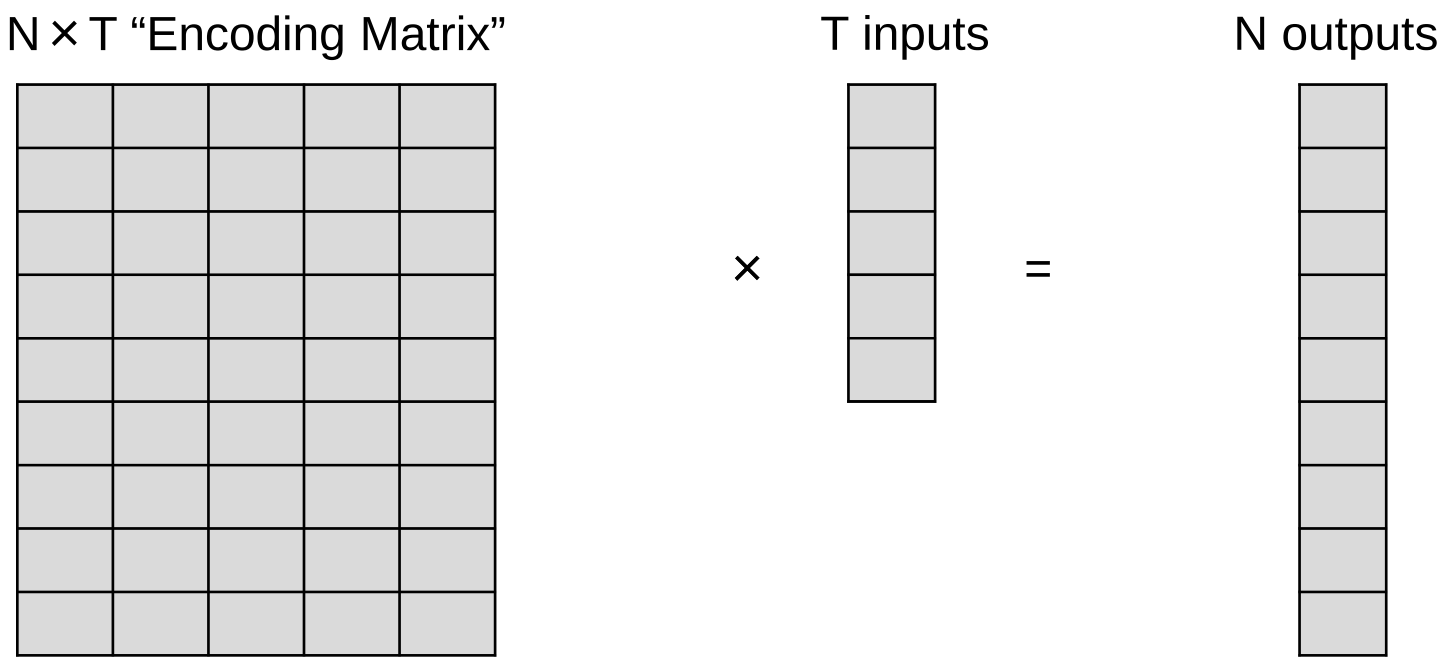
Erasure Codes and Integrity Checks

Erasure Codes and CRC

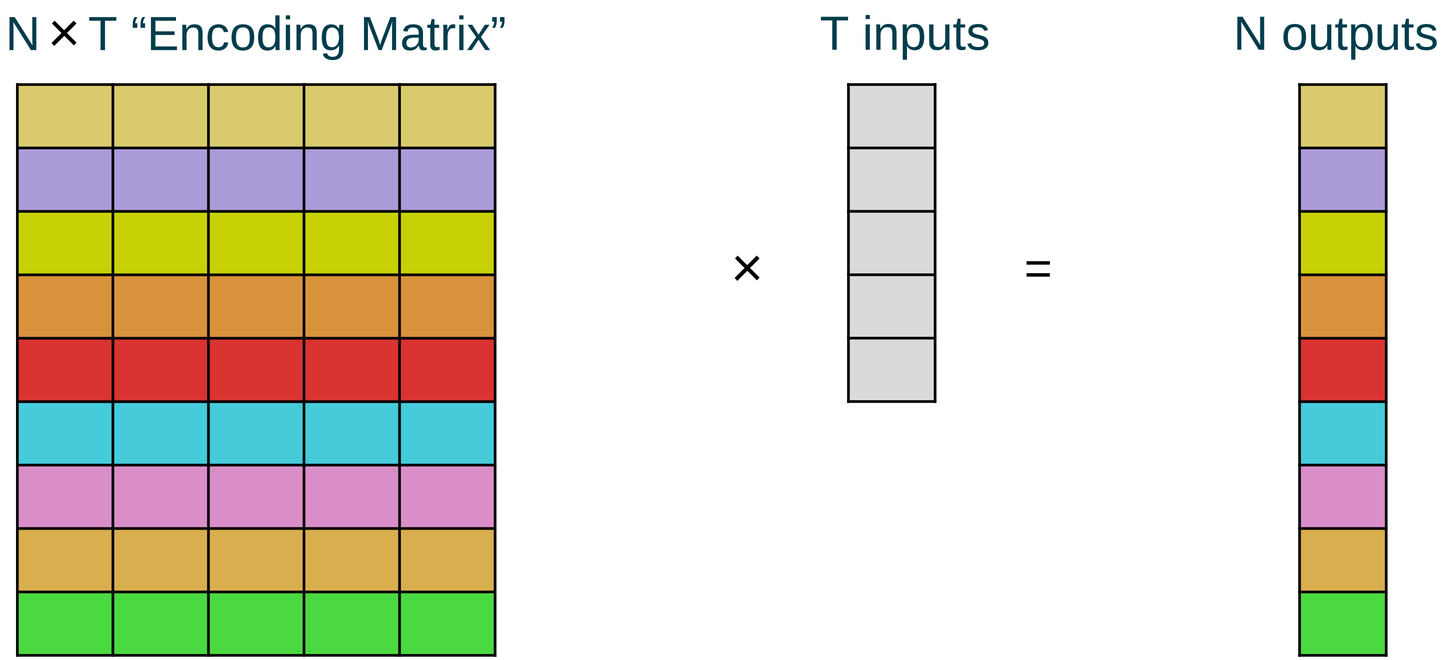
- We know Erasure Codes are linear
- We know Cyclic Redundancy Checks are linear
  - How can we combine them for practical uses?

- We will need to explore the internal workings of Erasure Codes to see how this is possible..

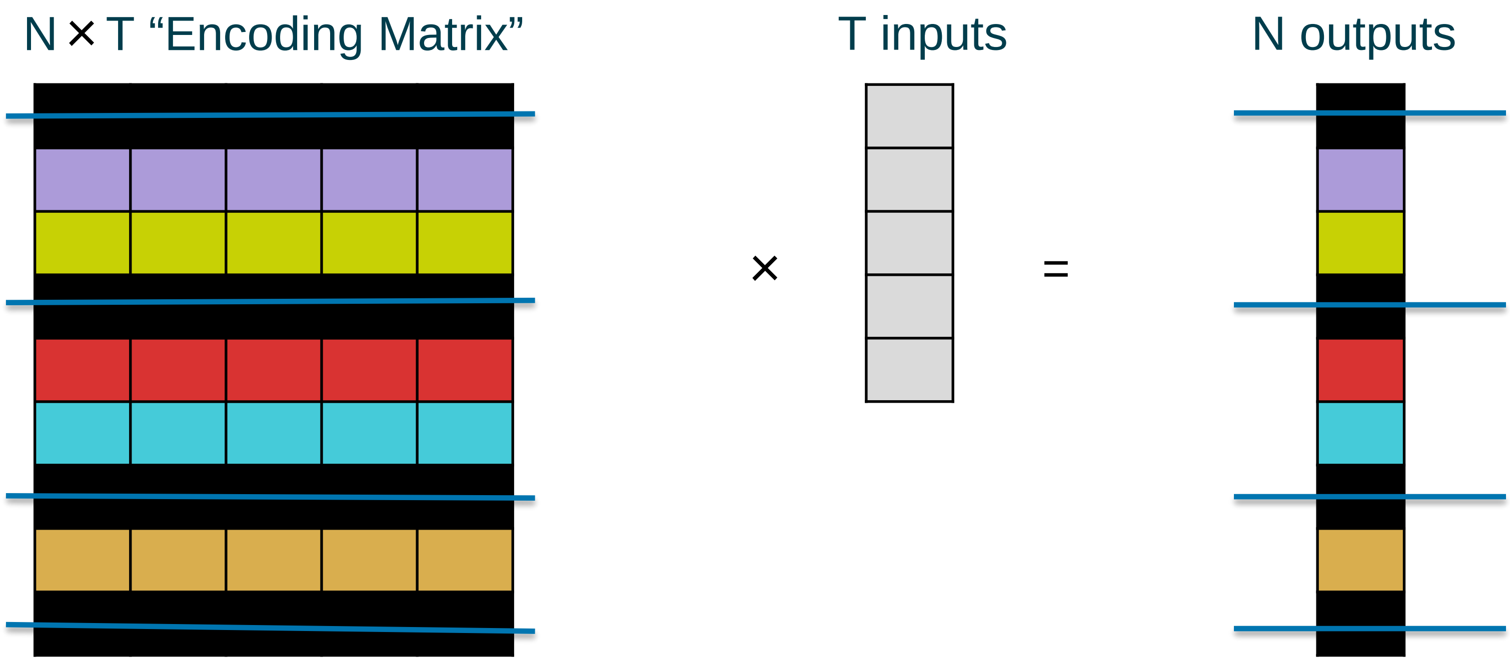
Erasure Codes: Encoding



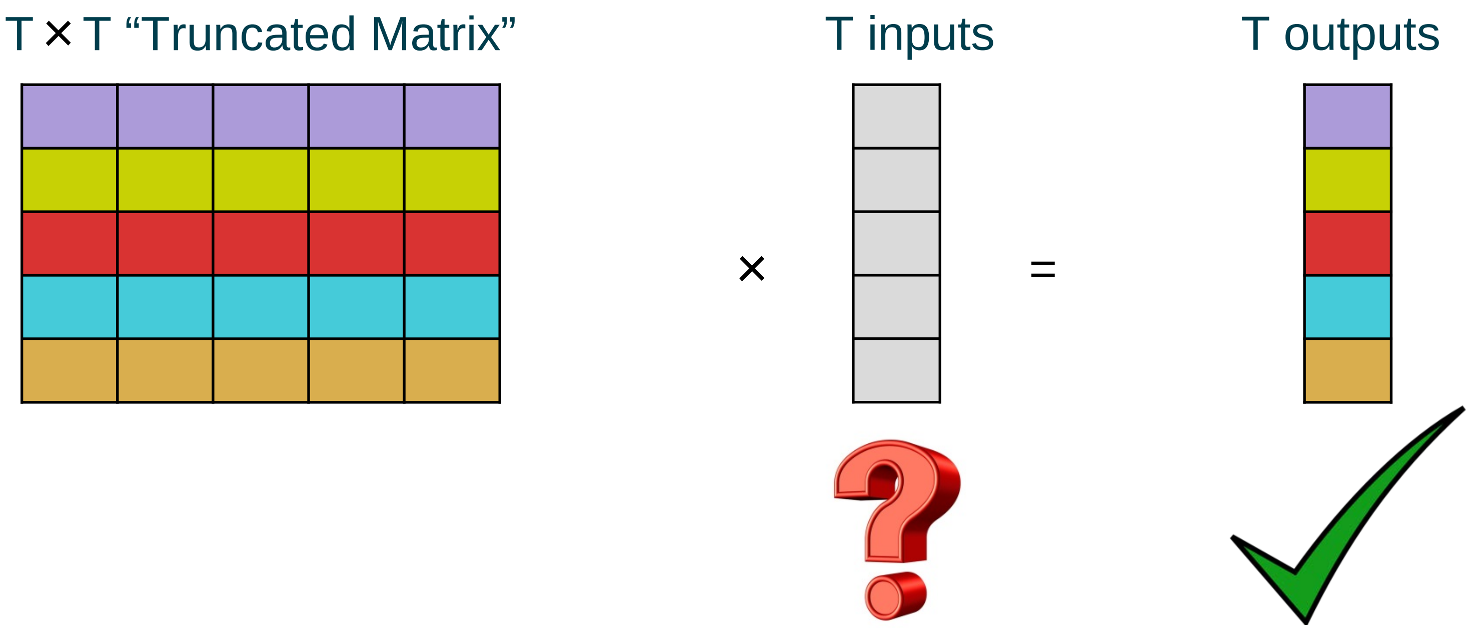
Erasure Codes: Encoding (Color Coded)



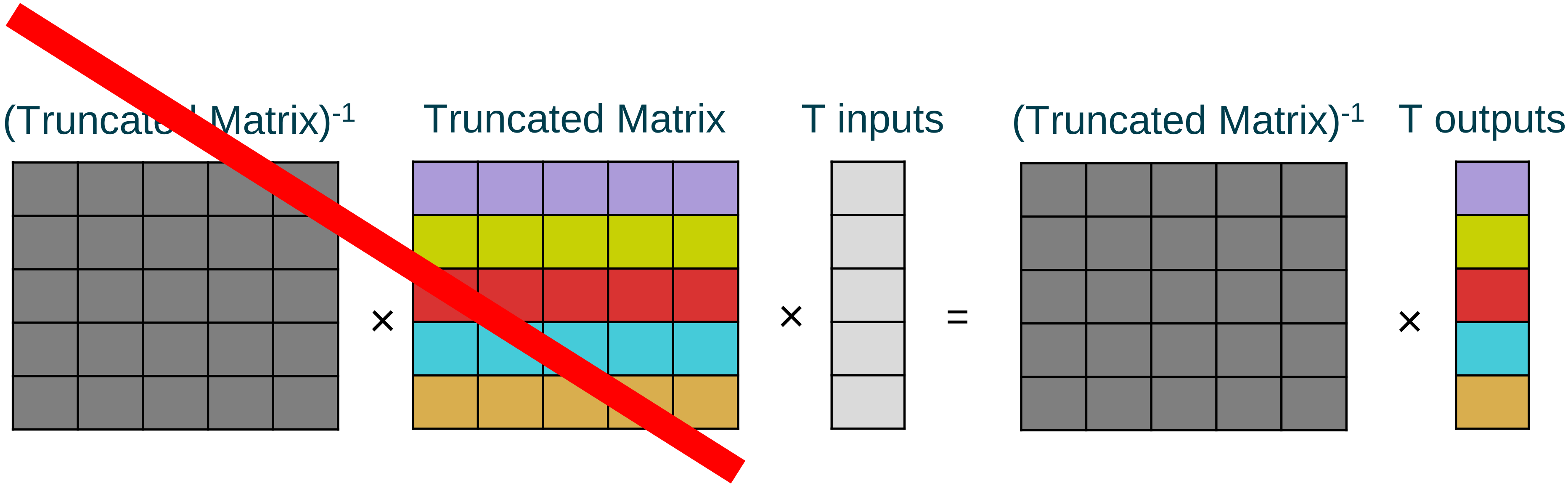
Erasure Codes: Delete Irrelevant Rows



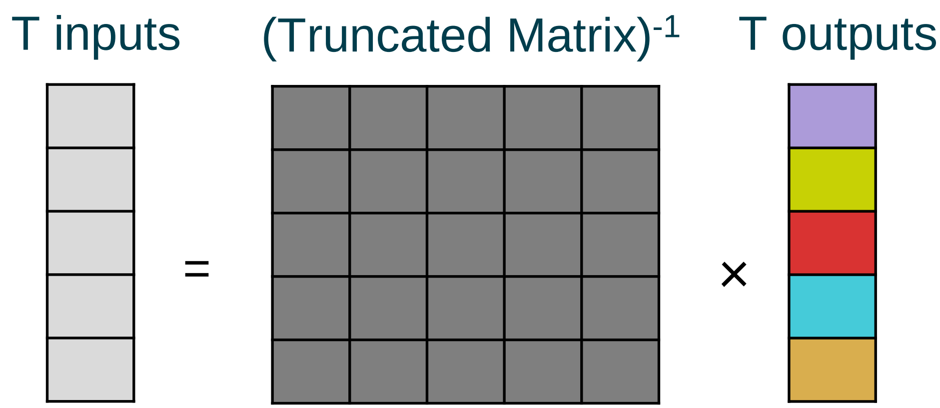
Erasure Codes: Getting the Inputs Back



Erasure Codes: Matrix Cancellation



Erasure Codes: Recovered Inputs

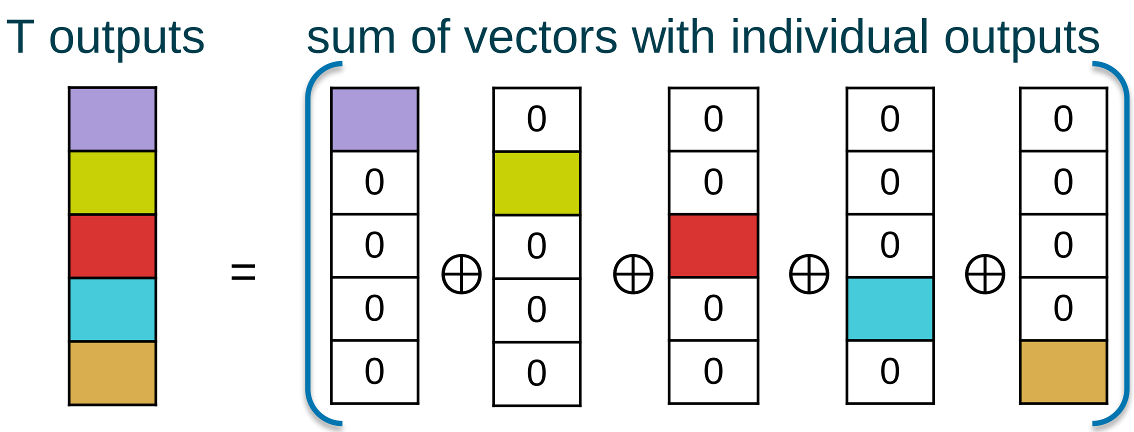


Combing CRCs with Erasure Codes

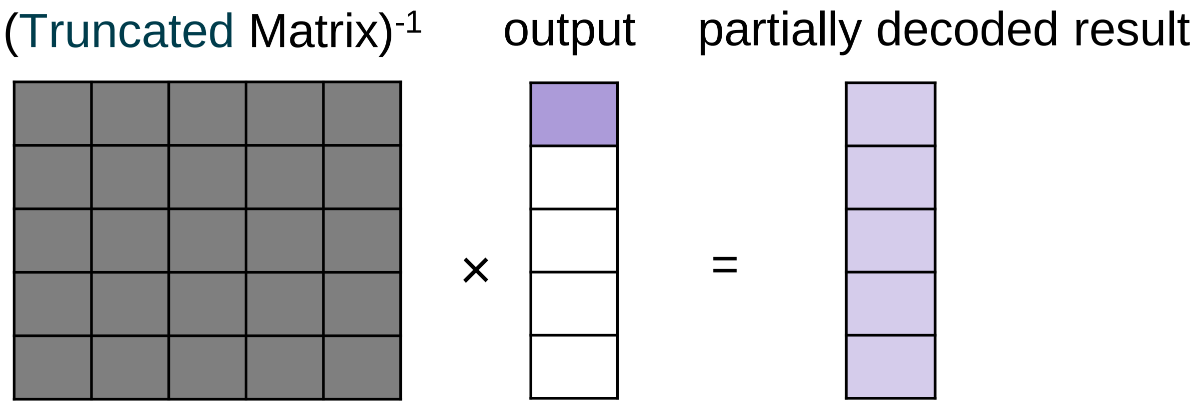
- With this background on how Erasure Codes encode and decode information, lets see what else we can do..
  - We observed decoding as happening all at once, given all ***T*** of the outputs
  - But it doesn't have to be like this, the decode operation can be done in ***T*** distinct steps

SDC 16

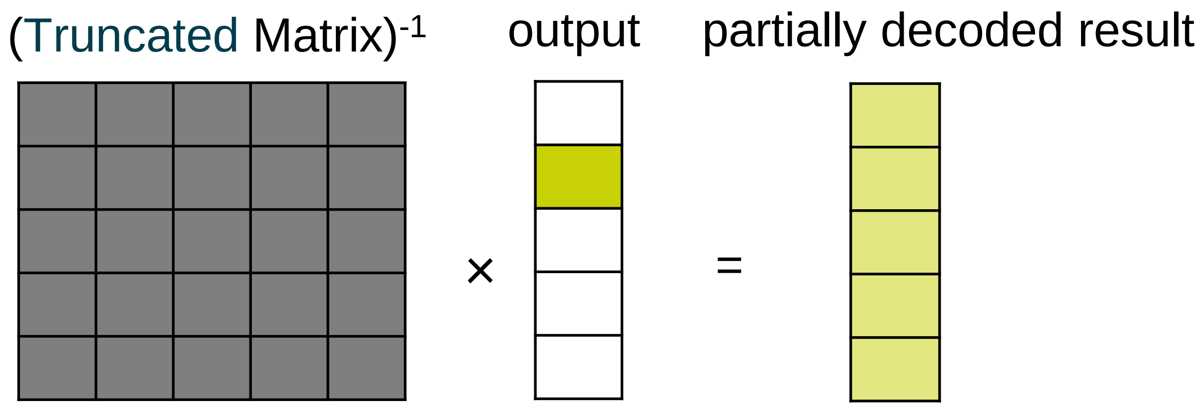
Decomposing the Output Vector



Decoding with one output at a time

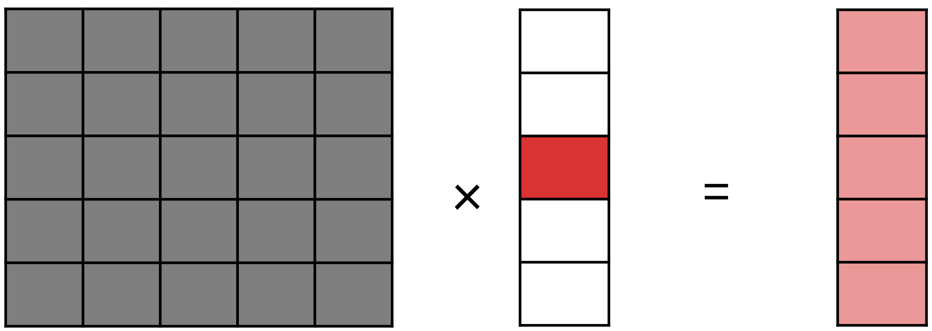


Decoding with one output at a time

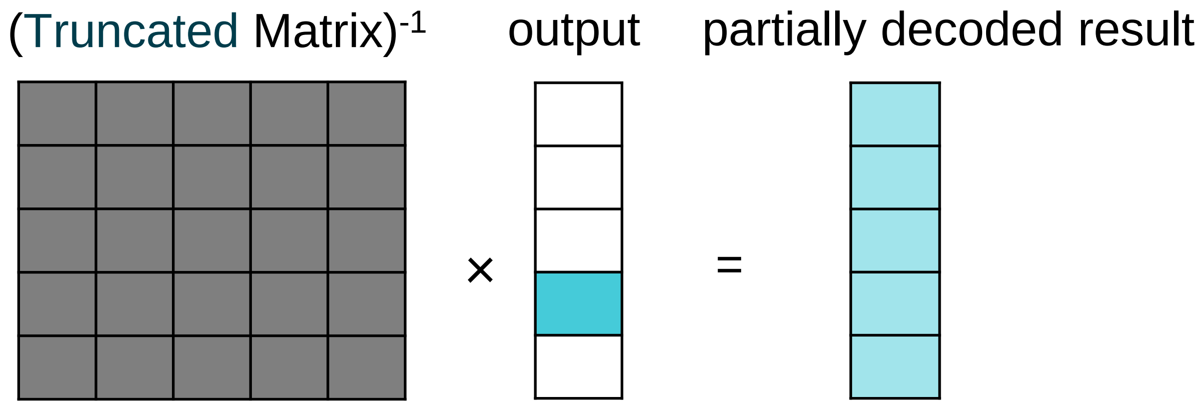


Decoding with one output at a time

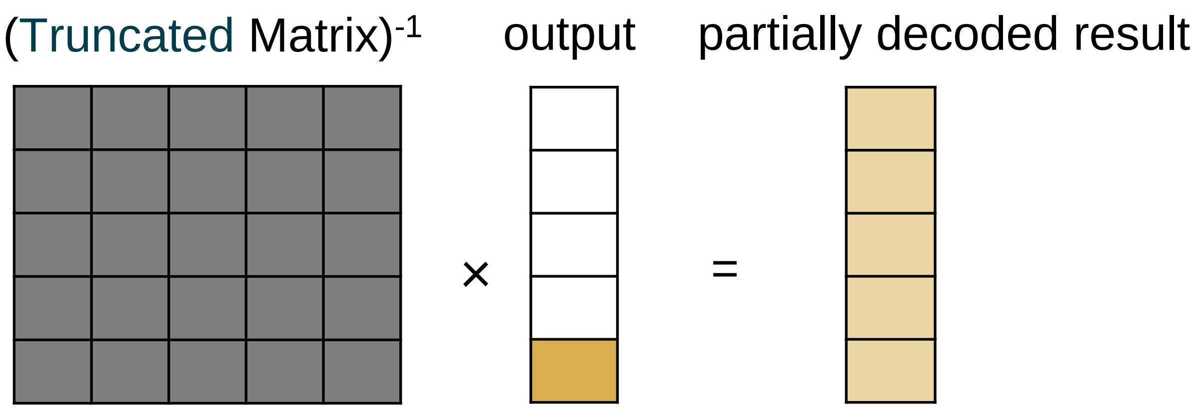
(Truncated Matrix)$^{-1}$ output partially decoded result



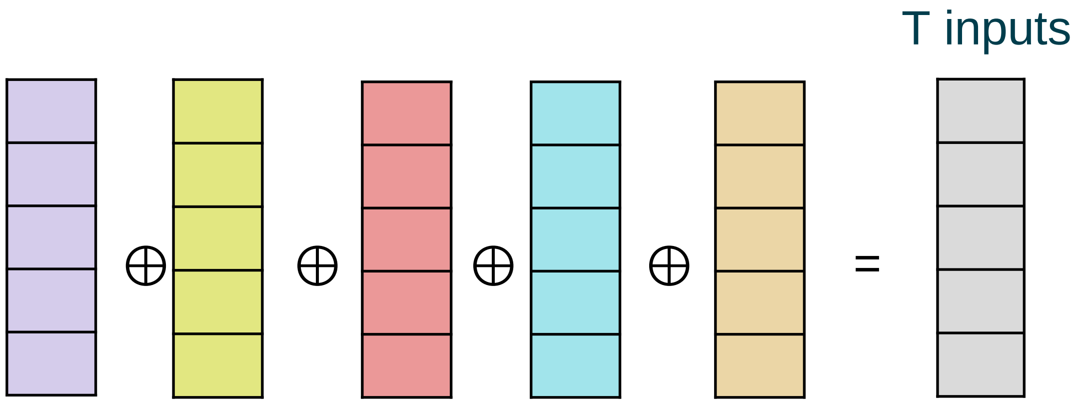
Decoding with one output at a time



Decoding with one output at a time



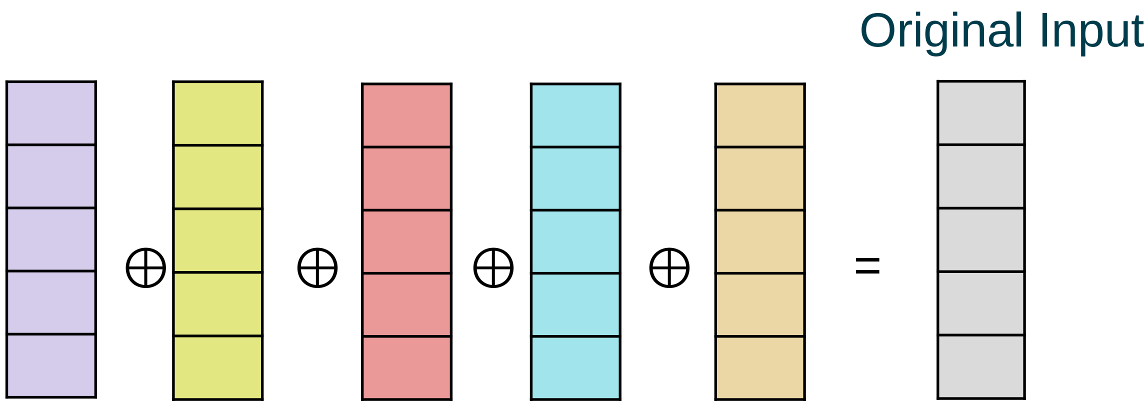
Putting them all together

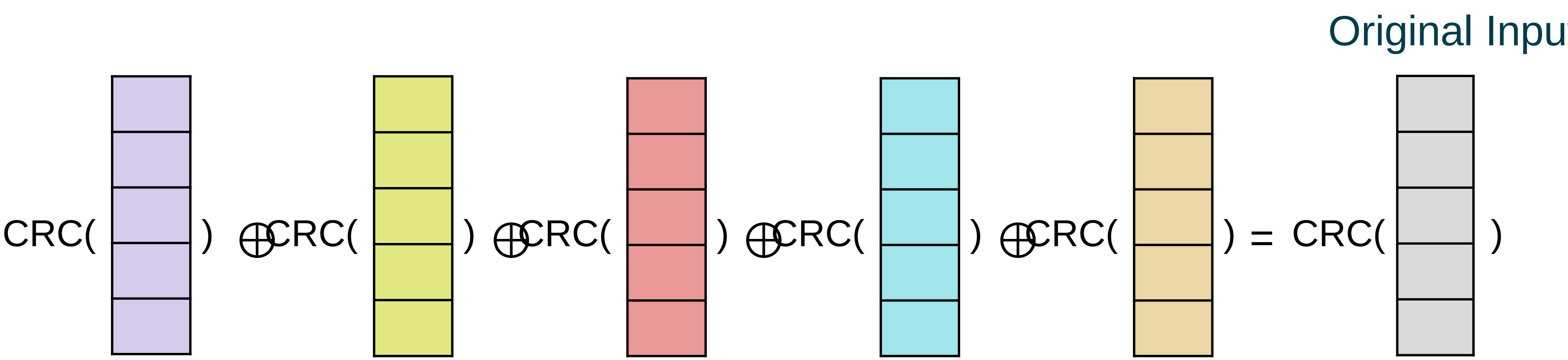


Observation

- Just as the ciphertext is the sum of a plaintext and a keystream, the original input is a sum of the ***T*** partially decoded results
  - This means the sum of the CRCs of the partially decoded results is the data’s CRC!

Computing CRCs of Data from fragments





Exploiting Linearity for New Applications

Secure Rebuilding

- Linearity also has applications for rebuilding:
  - It enables rebuilding of lost fragments, without having to decode the data first
    - Saves significant bandwidth in some topologies
  - It can be further extended to allow rebuilding without exposing any other fragment
    - Sounds paradoxical and impossible, but its not!
    - Very useful in cases of secret sharing

Conclusions

- Linear functions leave much room for innovation
- So far it has provided the ability to:
  - Encrypt, decrypt and rekey erasure coded data without any network transfer
  - Verify integrity of erasure coded data without having to download or retrieve it
  - Rebuild fragments with minimal network transfer and without having to decode data

Questions

Secure and Efficient Rebuilding:
http://dimacs.rutgers.edu/Workshops/SecureNetworking/Slides/resch.pptx