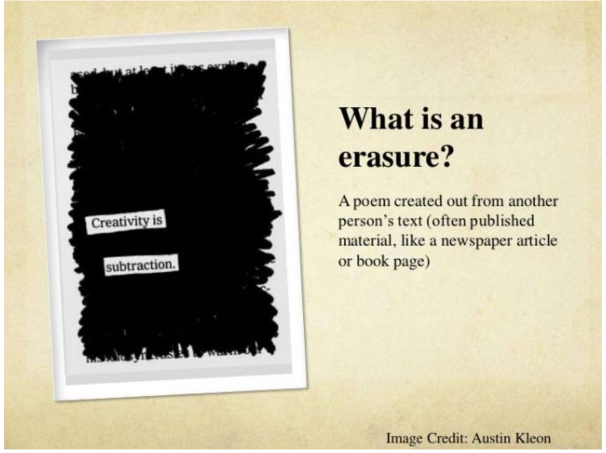


ERASURE CODES

INTRODUCTION

Erasures

- Byte errors where we ***know*** the **position** of the **dropped** or corrupted **bytes** is called an ***erasure***

- As opposed to byte errors where we don't know the position of the dropped or corrupted bytes

Types of Erasure Codes

- **Linear block code**
  – Reed Solomon Code
  – Can sustain lost bytes of known position
  – Distributed file systems

- **Fountain code**
  – LT Codes (Luby Transform)
  – Can sustain lost bytes of unknown position
  – P2P systems, torrents, video streaming …

Reed Solomon Code

- Error-correcting code
  - Used in QR codes
- Block encoding
  - **Data blocks**
  - Error-correcting blocks (**parity blocks**)
  - Read data blocks first
  - Else, decode with parity blocks

General Problem

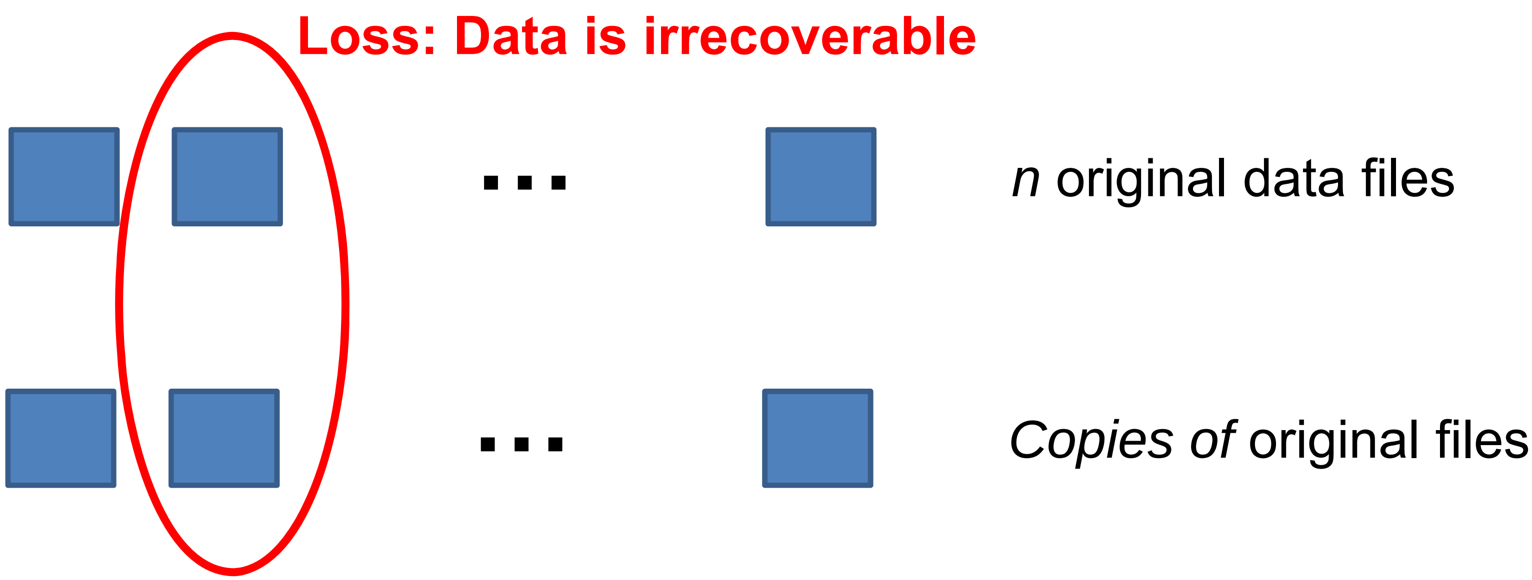
(*n*, *m*)-code

- We have ***n* data files**, guard against **losing *m*** of them
- Generate ***m* parity files**
- Lose up to *m* data files, can use equal number of parity files to recover data files
- Also works if some parity files are lost, as long as there are ***n* files left** (parity or data), can recover original data files

- Compare creating *n* parity files to making *a* copy of *n* data files (a.k.a. common backup strategy)

Recoverability

Via backup strategy



- With *n* original files and *n* copies (one each), if we loose two files (original and its copy), we can't recover loss
- Total storage requirement is 2*n*

Parity files take **same amount of space** but provide **superior (recovery capabilities**!

Recoverability

Via an (n, m)-code

Loss: Data remains recoverable

n original data files

n parity files

- With an (*n*, *m*)-code, we can protect *n* data files against the loss of *m* of them by generating *m* parity files
- Say, we use an (n, n)-code with total storage requirement 2*n*
- Could loose up to n files (any combination of data and parity files)

Optimal Erasure Code

- Above sketched (n, m)-code is an erasure code because it guards against byte erasures
- It does not guard against more general errors where we don't know which data bytes have been corrupted
- Called optimal because in general, we **need at least *n* known bytes to recover *n* data bytes**, bound achieved here

Erasure Code Computations

Parity & Data Reconstruction Computation

- Given a pair (n, m) and n data bytes
- **Compute parity data**: compute m parity bytes, given n data bytes
- **Reconstruct data**: given a partial list of at least n data and parity bytes
  - Return full list of data bytes, i.e., there are no more than m omitted data or parity bytes
  - Error otherwise
- Generally, operates on byte level

Erasure Code

n, m = 1

- For d_i a data byte, compute parity byte p
  - $p = d_0 + d_1 + \ldots + d_{n-1}$
- For m =1, guard against loss of a single d_i
- Reconstruct data
  - $\mathbf{d_i} = p - (d_0 + d_1 + \mathbf{d_{i-1}} + \mathbf{d_{i+1}} + \ldots + d_{n-1})$

Nota Bene

- In practise, above and below computations are done modulo 256
- Generally speaking, on finite fields, a.k.a. Galois fields
- For simplicity and illustration, we'll use decimal arithmetic
- Including in assignments, *etc*.
- Unless explicitly stated otherwise

Erasure Code

n = 3, m = 2

- (3, 2)-code
- Here, p_i **parity bytes**, d_i **data bytes** (i.e., numbers)
- Parity byte equations must be "*sufficiently different*"

$$p_0 = d_0 + d_1 + d_2$$
$$p_1 = 1 * d_0 + 2 * d_1 + 3 * d_2$$

- For example, the following is not "sufficiently different"

$$p_1 = 2 * d_0 + 2 * d_1 + 2 * d_2$$

- Here, $p_1 = 2\ p_0$ (**avoid linear combinations**)

For two **missing data bytes**, d_i, d_j $i < j$, and **given parity bytes** p_0, p_1, we rearrange parity equations to move unknown (i.e., missing data bytes) to left hand side:

Given parity equations:

$$p_0 = d_0 + d_1 + d_2$$

$$p_1 = 1 * d_0 + 2 * d_1 + 3 * d_2$$

Equations kept generic since we don't know upfront which bytes get lost

Rearranged (to solve for missing data bytes):

$$d_i + d_j = X = p_0 - d_k$$

$$(i + 1) * d_i + (j + 1) * d_j = Y = p_1 - (k + 1) * d_k$$

where $\boldsymbol{d_k}$ is the **known (not missing) data byte**

For two **missing data bytes**, d_i, d_j $i < j$, and **given parity bytes** p_0, p_1, we rearrange parity equations to move unknown (i.e., missing data bytes) to left hand side:

Given parity equations:

$$p_0 = d_0 + d_1 + d_2$$

$$p_1 = 1 * d_0 + 2 * d_1 + 3 * d_2$$

Rearranged (to solve for missing data bytes):

$$d_i + d_j = X = p_0 - d_k$$

$$(i + 1) * d_i + (j + 1) * d_j = Y = p_1 - (k + 1) * d_k$$

where $\boldsymbol{d_k}$ is the **known (not missing) data byte**

Multiply first equation by (i+1) and subtract it from second one to cancel the d_i term; then, use first equation to solve for d_i

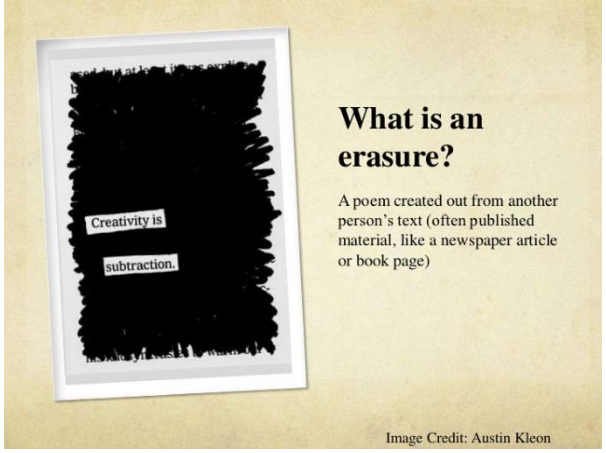
$$d_i + d_j = X = p_0 - d_k$$

$$(i+1) * d_i + (j+1) * d_j = Y = p_1 - (k+1) * d_k$$

$$d_j = (Y - (i-1) * X)/(j - i)$$

$$d_i = X - d_j = (((j+1) * X - Y)/(j - i)$$

We now have equations for d_i, d_j, $i < j$ to reconstruct the missing data from know parity bytes.



ERASURE CODES

BASIC LINEAR ALGEBRA

Basic Linear Algebra

Digression

Equations of parity numbers from above example are of the form:

$$p = a_0 * d_0 + a_1 * d_1 + a_2 * d_2$$

where a_i are constants

These are **linear combinations** of d_is and written as

$$p = (a_0 \quad a_1 \quad a_2).\begin{bmatrix} d_0 \\ d_1 \\ d_2 \end{bmatrix}$$

Correspondence

Linear Functions and Matrices: Taking *n* inputs to *m* outputs

Two parity numbers, each a linear combination of d_is:

$$p_0 = a_{00} * d_0 + a_{01} * d_1 + a_{02} * d_2$$

$$p_1 = a_{10} * d_0 + a_{11} * d_1 + a_{12} * d_2$$

$$\begin{bmatrix} p_0 \\ p_1 \end{bmatrix} = \begin{pmatrix} a_{00} & a_{01} & a_{02} \\ a_{10} & a_{11} & a_{12} \end{pmatrix} \cdot \begin{bmatrix} d_0 \\ d_1 \\ d_2 \end{bmatrix}$$

Linear Function vs. Matrix

- *Deleting a row* of a matrix corresponds to *deleting an output* of a linear function

Output

Input

$$\begin{bmatrix} p_0 \\ p_1 \end{bmatrix} = \begin{pmatrix} a_{00} & a_{01} & a_{02} \\ a_{10} & a_{11} & a_{12} \end{pmatrix} \cdot \begin{bmatrix} d_0 \\ d_1 \\ d_2 \end{bmatrix}$$

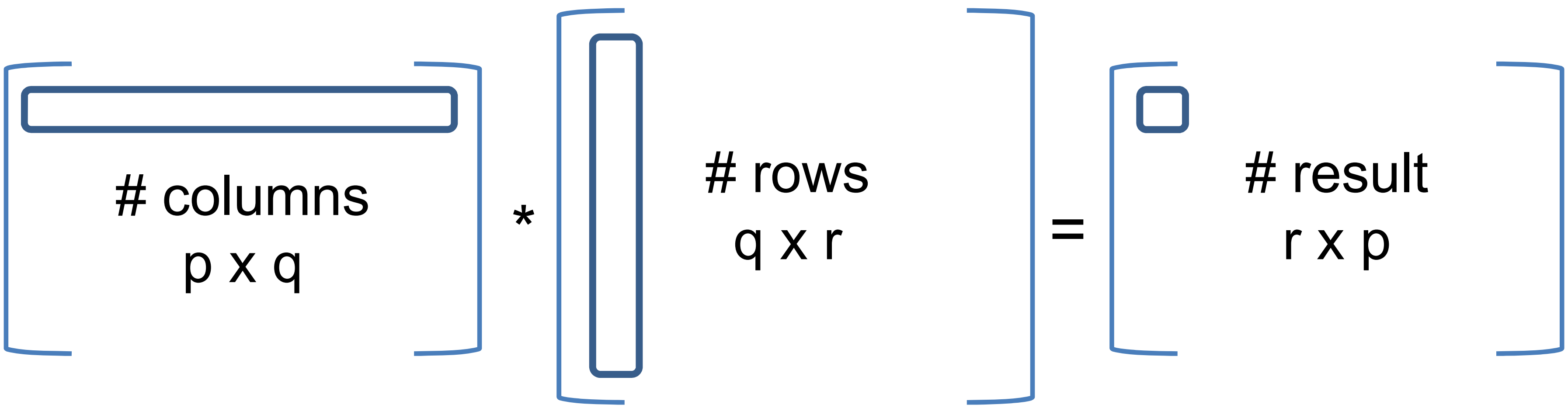
Linear Function vs. Matrix

- *Deleting a row* of a matrix corresponds to *deleting an output* of a linear function
- *Swapping two rows* of a matrix corresponds to *swapping two outputs* of a linear function
- *Appending a row* to a matrix corresponds to *adding an output* to a linear function

Matrix Multiplication

- Multiply matrices *A*, *B*, if they are **compatible**
- Number of columns of *A* must equal number of rows of *B*
  - *A* is ***p* x *q*** matrix (#rows x #columns)
  - *B* is a ***q* x *r*** matrix
  - Resulting *C* is an ***r* x *p*** matrix

Matrix Multiplication



Matrix Multiplication

```
MATRIX-MULTIPLY (A,B)
if A.columns ≠ B.rows
    error "incomplete dimensions"
else let C be a new A.rows × B.columns matrix
    for i = 1 to A.rows
        for j = 1 to B.columns
            c_ij = 0
            for k = 1 to A.columns
                c_ij = c_ij + a_ik * b_kj
return C
```

Identity Matrix

Identity function returns its *n* inputs as its outputs, corresponding matrix is the **identity matrix**

$$I_n = \begin{pmatrix} 1 & 0 & \cdots & 0 & 0 \\ 0 & 1 & \cdots & 0 & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & \cdots & 1 & 0 \\ 0 & 0 & \cdots & 0 & 1 \end{pmatrix}$$

The Inverse

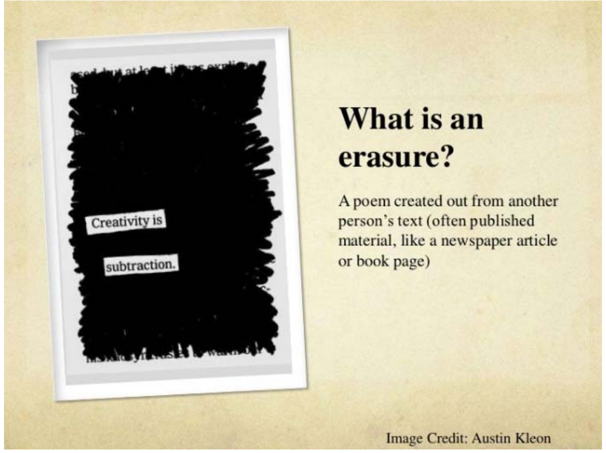
M a square matrix (*n* x n) and
M^{-1} its inverse, if it exists

$$M * M^{-1} = M^{-1} * M = I_n$$

M Invertible

$$M = \begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix} \qquad M^{-1} = \begin{pmatrix} -2 & 1 \\ 3/2 & -1/2 \end{pmatrix}$$

$$M * M^{-1} = M^{-1} * M = I_n$$



ERASURE CODES

(3, 2)-CODE EXAMPLE CONTINUED

Erasure Codes

n = 3, m = 2 from above example (3, 2)-code

$$p_0 = d_0 + d_1 + d_2$$
$$p_1 = 1 * d_0 + 2 * d_1 + 3 * d_2$$

$$P = \begin{pmatrix} 1 & 1 & 1 \\ 1 & 2 & 3 \end{pmatrix}$$

$$\begin{bmatrix} p_0 \\ p_1 \end{bmatrix} = \begin{pmatrix} 1 & 1 & 1 \\ 1 & 2 & 3 \end{pmatrix} \cdot \begin{bmatrix} d_0 \\ d_1 \\ d_2 \end{bmatrix}$$

Data Reconstruction Matrix

A.k.a. encoding matrix

$$\begin{bmatrix} d_0 \\ d_1 \\ d_2 \\ p_0 \\ p_1 \end{bmatrix} = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 1 & 1 \\ 1 & 2 & 3 \end{pmatrix} * \begin{bmatrix} d_0 \\ d_1 \\ d_2 \end{bmatrix}$$

Identity matrix

Parity matrix / encoding matrix fragment

Linear function that returns input data and parity bytes

Data Reconstruction Matrix

A.k.a. encoding matrix

$$\begin{bmatrix} d_0 \\ d_1 \\ d_2 \\ p_0 \\ p_1 \end{bmatrix} = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 1 & 1 \\ 1 & 2 & 3 \end{pmatrix} * \begin{bmatrix} d_0 \\ d_1 \\ d_2 \end{bmatrix}$$

Linear function that returns input data and parity bytes

Data Loss Example

$$\begin{bmatrix} \cancel{d_0} \\ d_1 \\ \cancel{d_2} \\ p_0 \\ p_1 \end{bmatrix} = \begin{pmatrix} \cancel{1} & \cancel{0} & \cancel{0} \\ 0 & 1 & 0 \\ \cancel{0} & \cancel{0} & \cancel{1} \\ 1 & 1 & 1 \\ 1 & 2 & 3 \end{pmatrix} * \begin{bmatrix} d_0 \\ d_1 \\ d_2 \end{bmatrix}$$

Linear function mapping data bytes to non-lost data and parity bytes:

$$\begin{bmatrix} d_1 \\ p_0 \\ p_1 \end{bmatrix} = \begin{pmatrix} 0 & 1 & 0 \\ 1 & 1 & 1 \\ 1 & 2 & 3 \end{pmatrix} . \begin{bmatrix} d_0 \\ d_1 \\ d_2 \end{bmatrix}$$

On the wrong side of equation ☹

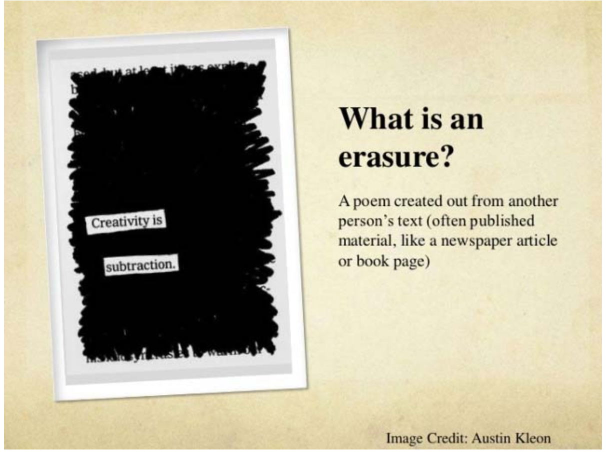
Solve for d_is

Need Inverse of Reconstruction Matrix

$$\begin{bmatrix} d_1 \\ p_0 \\ p_1 \end{bmatrix} = \begin{pmatrix} 0 & 1 & 0 \\ 1 & 1 & 1 \\ 1 & 2 & 3 \end{pmatrix} . \begin{bmatrix} d_0 \\ d_1 \\ d_2 \end{bmatrix} \quad M^{-1} = \begin{pmatrix} -1/2 & 3/2 & -1/2 \\ 1 & 0 & 0 \\ -1/2 & -1/2 & 1/2 \end{pmatrix}$$

Gives us **original data bytes** in terms of **known data** and **parity bytes**!

$$\begin{bmatrix} d_0 \\ d_1 \\ d_2 \end{bmatrix} = \begin{pmatrix} -1/2 & 3/2 & -1/2 \\ 1 & 0 & 0 \\ -1/2 & -1/2 & 1/2 \end{pmatrix} \begin{bmatrix} d_1 \\ p_0 \\ p_1 \end{bmatrix}$$



ERASURE CODES

GENERALIZING TO (N, M)-CODES

Generalizing

Arbitrary *n*, *m* - compute parity matrix

Generate an *m* x *n* **parity matrix *P*** (rows need to be "*sufficiently different*")

$$\begin{bmatrix} p_0 \\ \vdots \\ p_{m-1} \end{bmatrix} = \begin{pmatrix} \boldsymbol{p}_0 \\ \vdots \\ \boldsymbol{p}_{m-1} \end{pmatrix} . \begin{bmatrix} d_0 \\ \vdots \\ d_{n-1} \end{bmatrix}$$

$\boldsymbol{p}_i$ are rows of *P* – *an m* x *n* parity matrix

Compute Data Reconstruction Matrix

Append rows of P to I_n, denoted as e_i

$$\begin{bmatrix} d_0 \\ \vdots \\ d_{n-1} \\ p_0 \\ \vdots \\ p_{m-1} \end{bmatrix} = \begin{pmatrix} e_0 \\ \vdots \\ e_{n-1} \\ \boldsymbol{p}_0 \\ \vdots \\ \boldsymbol{p}_{m-1} \end{pmatrix} * \begin{bmatrix} d_0 \\ \vdots \\ d_{n-1} \end{bmatrix}$$

Data Loss with Resulting Matrix

- Indices $k \leq m$ of **missing data bytes** are $i_0, \ldots, i_{k-1}$
- Remove rows corresponding to missing data bytes
- Keep k **parity rows** $p_0, \ldots, p_{k-1}$
- $j_0, \ldots, j_{n-k-1}$ indices of present n-k data bytes

$$\begin{bmatrix} d_{j0} \\ \vdots \\ d_{j_{n-k-1}} \\ p_0 \\ \vdots \\ p_{k-1} \end{bmatrix} = \begin{pmatrix} e_{j_0} \\ \vdots \\ e_{j_{n-k-1}} \\ \boldsymbol{p}_0 \\ \vdots \\ \boldsymbol{p}_{k-1} \end{pmatrix} \cdot \begin{bmatrix} d_0 \\ \vdots \\ d_{n-1} \end{bmatrix}$$

Compute *M*'s Inverse M^{-1}

Reconstruct data by multiplying with M^{-1}

If P was chosen correctly, M^{-1} exists.

$$\begin{bmatrix} d_0 \\ \vdots \\ d_{n-1} \end{bmatrix} = M^{-1} * \begin{bmatrix} d_{j_0} \\ \vdots \\ d_{j_{n-k-1}} \\ \boldsymbol{p}_0 \\ \vdots \\ \boldsymbol{p}_{k-1} \end{bmatrix}$$

Loose Ends

- *How do we compute matrix inverses?*
- *How do we generate "optimal" parity matrices P s.t. M^{-1} always exists?*
- *How do we compute parity bytes instead of parity numbers* (informally referred to as bytes, above)?

Facts: Non-invertible Matrix

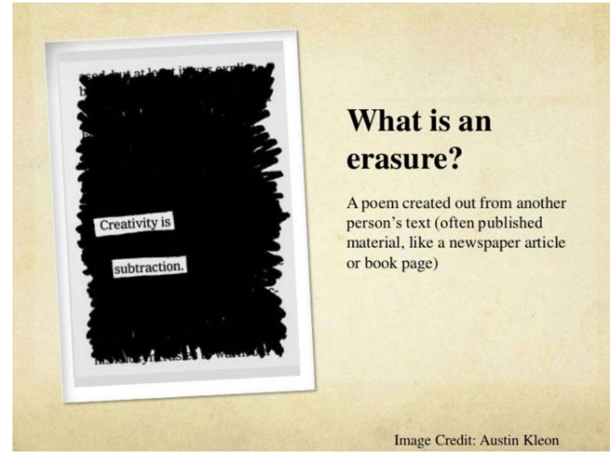
More a negative result

- M has a row that can be expressed as a linear combination of other rows of M then M is non-invertible

- Said differently:
  - Linear function corresponding to M has one of its outputs redundant with the other outputs, so it is essentially a linear function taking n inputs to fewer than m outputs

“Not Sufficiently Different”

- If one parity function is a linear combination of other parity functions then it is redundant, i.e., not sufficiently different
- Therefore, choose P wisely
- **No row of *P* should be expressed as linear combination of other rows of *P***
- ***…***



ERASURE CODES

ANOTHER EXAMPLE

Erasure Coding Example

- Storage: “My private key”
- Pad with spaces to obtain right length if needed
  - “My private key_ _” (added two spaces to obtain length of 16 characters)
- Build data matrix, here, a 4x4 matrix (ASCII code)

```
My p
riva
te k
ey__
```

Encoding Matrix

Protect against loss of 2 bytes

- Identity matrix appended with
- Parity matrix
  - Derived form parity equations, i.e., one per parity byte (a.k.a. encoding matrix fragment)
- Results in full encoding matrix

Encoding Matrix

(For a (3, 2)-code)

$$\begin{bmatrix} d_0 \\ d_1 \\ d_2 \\ p_0 \\ p_1 \end{bmatrix} = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 1 & 1 \\ 1 & 2 & 3 \end{pmatrix} \cdot \begin{bmatrix} d_0 \\ d_1 \\ d_2 \end{bmatrix}$$

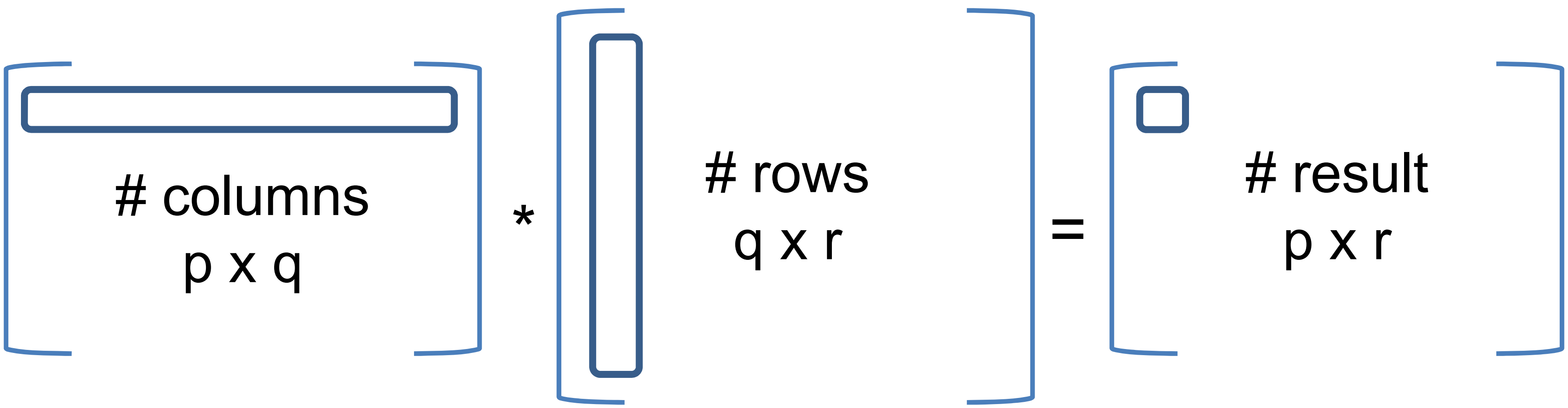
Identity matrix

Encoding matrix fragment – derived from parity equations

Parity equations need to be “sufficiently different,” e.g., not be linear combinations of each other.

Parity Matrix

- Results from **multiplying encoding matrix with original data matrix**



Data Loss Example

$$\begin{bmatrix} \cancel{d_0} \\ d_1 \\ \cancel{d_2} \\ p_0 \\ p_1 \end{bmatrix} = \begin{pmatrix} \cancel{1} & \cancel{0} & \cancel{0} \\ 0 & 1 & 0 \\ \cancel{0} & \cancel{0} & \cancel{1} \\ 1 & 1 & 1 \\ 1 & 2 & 3 \end{pmatrix} . \begin{bmatrix} d_0 \\ d_1 \\ d_2 \end{bmatrix}$$

Data reconstruction matrix:

$$\begin{bmatrix} d_1 \\ p_0 \\ p_1 \end{bmatrix} = \begin{pmatrix} 0 & 1 & 0 \\ 1 & 1 & 1 \\ 1 & 2 & 3 \end{pmatrix} . \begin{bmatrix} d_0 \\ d_1 \\ d_2 \end{bmatrix}$$

Recovering Lost Data

- Find **Inverse** to data reconstruction matrix (from previous slide; it is a **square matrix**)
- For large matrix, use online tools to determine matrix inverse
- Recover data by multiplying **Inverse with relevant rows of parity matrix** (rows not affected by data loss)

Solve for d_is

Need Inverse of Data Reconstruction Matrix

$$\begin{bmatrix} d_1 \\ p_0 \\ p_1 \end{bmatrix} = \begin{pmatrix} 0 & 1 & 0 \\ 1 & 1 & 1 \\ 1 & 2 & 3 \end{pmatrix} . \begin{bmatrix} d_0 \\ d_1 \\ d_2 \end{bmatrix} M^{-1} = \begin{pmatrix} -1/2 & 3/2 & -1/2 \\ 1 & 0 & 0 \\ -1/2 & -1/2 & 1/2 \end{pmatrix}$$

Gives us original data bytes in terms of known data and parity bytes!

$$\begin{bmatrix} d_0 \\ d_1 \\ d_2 \end{bmatrix} = \begin{pmatrix} -1/2 & 3/2 & -1/2 \\ 1 & 0 & 0 \\ -1/2 & -1/2 & 1/2 \end{pmatrix} \begin{bmatrix} d_1 \\ p_0 \\ p_1 \end{bmatrix}$$