Modular Inverses

- Additive inverse of x mod n, denoted $-x \bmod n$, is the number that must be added to x to get 0 mod n
  - $-2 \bmod 6 = 4$, since $2 + 4 = 0 \bmod 6$
- Multiplicative inverse of x mod n, denoted $x^{-1} \bmod n$, is the number that must be multiplied by x to get 1 mod n
  - $3^{-1} \bmod 7 = 5$, since $3 \cdot 5 = 1 \bmod 7$

Relative Primality

- x and y are **relatively prime** if they have no common factor other than 1
- x^{-1} mod y exists only when x and y are relatively prime
- If it exists, x^{-1} mod y is easy to compute using Euclidean Algorithm
  - We won't do the computation here
  - But, an efficient algorithm exists

Totient Function

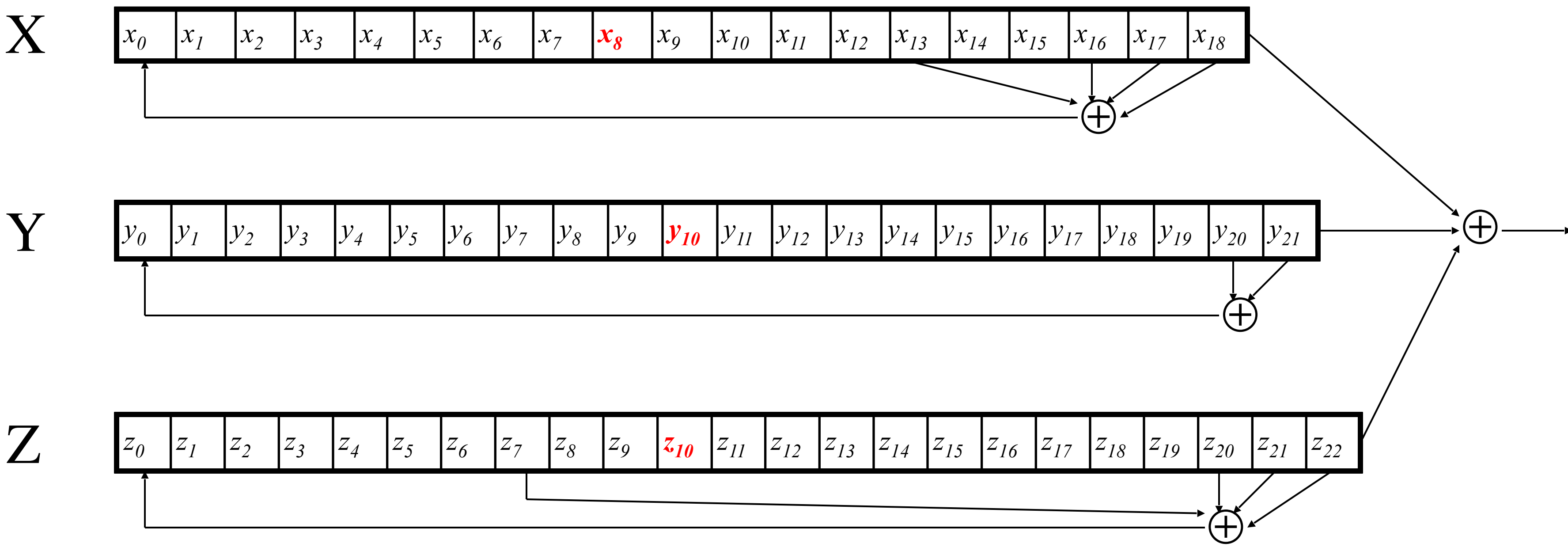
- $\varphi(n)$ is "the number of numbers less than n that are relatively prime to n"
  - Here, "numbers" are positive integers
- Examples
  - $\varphi(4) = 2$ since 4 is relatively prime to 3 and 1
  - $\varphi(5) = 4$ since 5 is relatively prime to 1,2,3,4
  - $\varphi(12) = 4$
  - $\varphi(p) = p-1$ if p is prime
  - $\varphi(pq) = (p-1)(q-1)$ if p and q prime

A5/1: Keystream

- At each iteration: $m = \text{maj}(x_8, y_{10}, z_{10})$
  - Examples: $\text{maj}(0,1,0) = 0$ and $\text{maj}(1,1,0) = 1$
- If $x_8 = m$ then X steps
  - $t = x_{13} \oplus x_{16} \oplus x_{17} \oplus x_{18}$
  - $x_i = x_{i-1}$ for $i = 18,17,\ldots,1$ and $x_0 = t$
- If $y_{10} = m$ then Y steps
  - $t = y_{20} \oplus y_{21}$
  - $y_i = y_{i-1}$ for $i = 21,20,\ldots,1$ and $y_0 = t$
- If $z_{10} = m$ then Z steps
  - $t = z_7 \oplus z_{20} \oplus z_{21} \oplus z_{22}$
  - $z_i = z_{i-1}$ for $i = 22,21,\ldots,1$ and $z_0 = t$
- Keystream **bit** is $x_{18} \oplus y_{21} \oplus z_{22}$

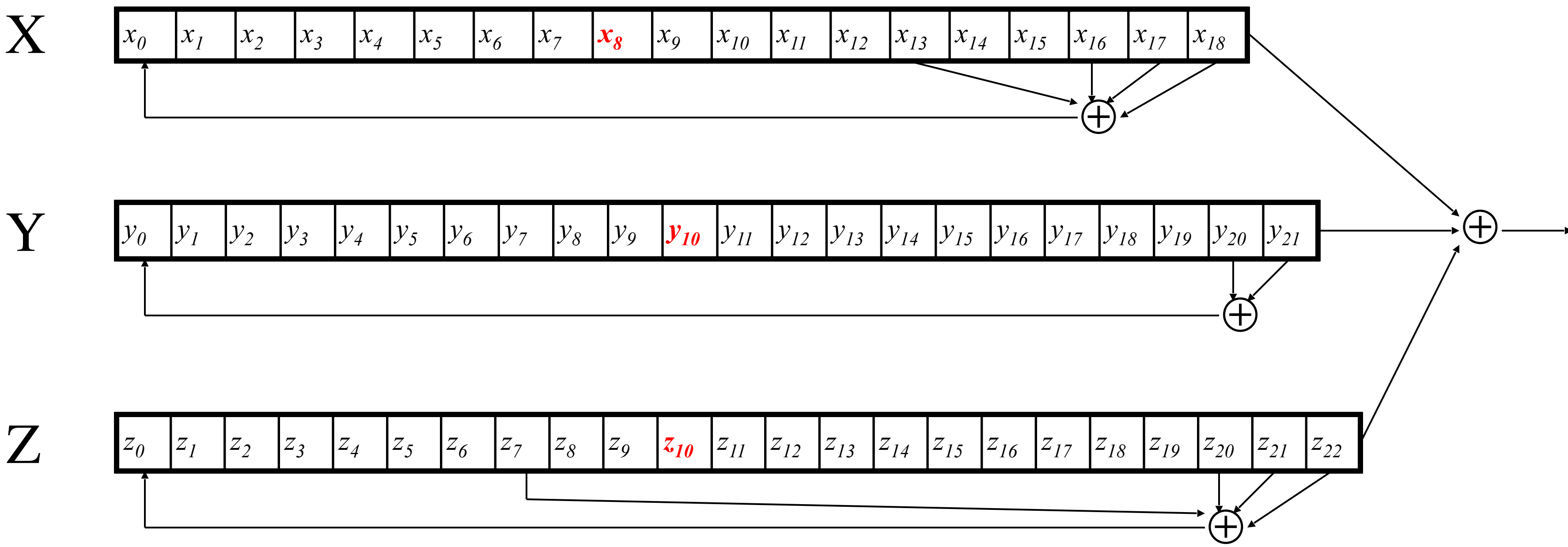
When register steps:
1. Computes new first bit
2. THEN, shifts

A5/1



- Each variable here is a single bit
- Key is used as **initial fill** of registers
- Each register steps (or not) based on $\mathrm{maj}(x_8, y_{10}, z_{10})$
- Keystream bit is XOR of rightmost bits of registers

A5/1

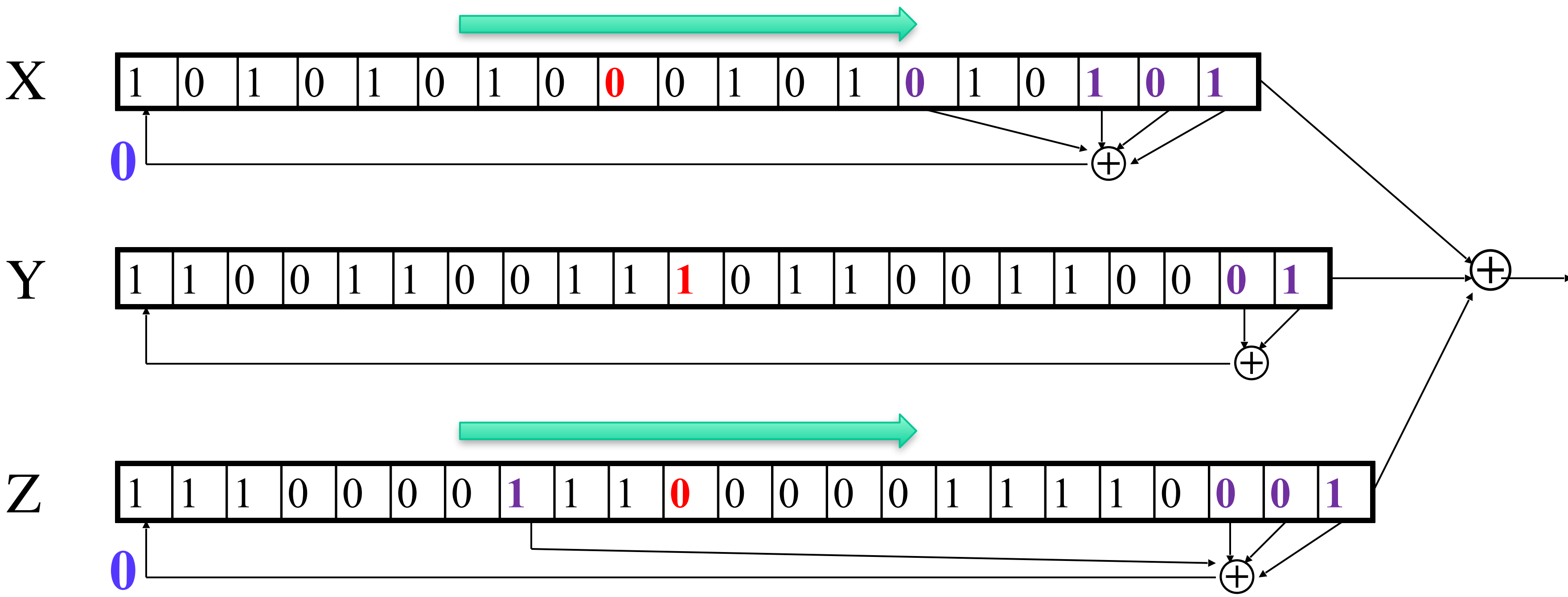


- Example - Key (64bits):

1010101000101010101110011001110110011000111100001110000011110001

  - 1010101000101010101 (first 19 bits) -> X
  - 1100110011101100110001 (middle 22 bits) -> Y
  - 11100001110000011110001 (last 23 bits) -> Z

A5/1

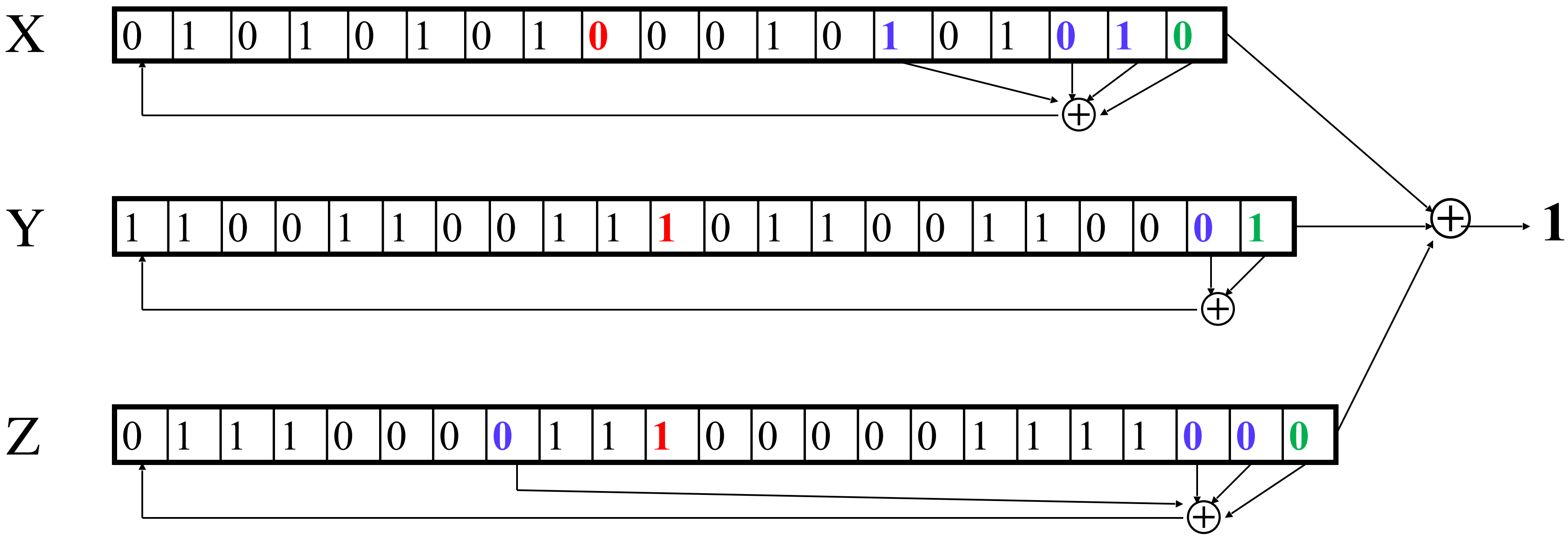


1. Majority vote: $m = \text{maj}(x_8, y_{10}, z_{10}) = \text{maj}(0,1,0) = \mathbf{0}$
2. Compute new first bits:

   X: $0 \oplus 1 \oplus 0 \oplus 1 = \mathbf{0}$

   Z: $1 \oplus 0 \oplus 0 \oplus 1 = \mathbf{0}$
3. Shift!

A5/1



1. Majority vote: $m = \text{maj}(x_8, y_{10}, z_{10}) = \text{maj}(0,1,0) = \mathbf{0}$
2. Compute new first bits:

   X: $0 \oplus 1 \oplus 0 \oplus 1 = \mathbf{0}$

   Z: $1 \oplus 0 \oplus 0 \oplus 1 = \mathbf{0}$
3. Shift!

Block Cipher Modes

ECB Mode

- Notation: $C = E(P,K)$
- Given plaintext $P_0, P_1, \ldots, P_m, \ldots$
- Most obvious way to use a block cipher:

EncryptDecrypt

$C_0 = E(P_0, K)$	$P_0 = D(C_0, K)$
$C_1 = E(P_1, K)$	$P_1 = D(C_1, K)$
$C_2 = E(P_2, K)\ \ldots$	$P_2 = D(C_2, K)\ \ldots$

- For fixed key K, this is "electronic" version of a codebook cipher (without additive)
  - With a different codebook for each key

Cipher Block Chaining (CBC) Mode

- Blocks are "chained" together
- A random initialization vector, or IV, is required to initialize CBC mode
- IV is random, but not secret

EncryptionDecryption

$C_0 = E(IV \oplus P_0, K)$,	$P_0 = IV \oplus D(C_0, K)$,
$C_1 = E(C_0 \oplus P_1, K)$,	$P_1 = C_0 \oplus D(C_1, K)$,
$C_2 = E(C_1 \oplus P_2, K),\ldots$	$P_2 = C_1 \oplus D(C_2, K),\ldots$

- Analogous to classic codebook with additive

Counter Mode (CTR)

- CTR is popular for random access
- Use block cipher like a stream cipher

EncryptionDecryption

$$C_0 = P_0 \oplus E(IV, K),$$
$$C_1 = P_1 \oplus E(IV+1, K),$$
$$C_2 = P_2 \oplus E(IV+2, K), \ldots$$

$$P_0 = C_0 \oplus E(IV, K),$$
$$P_1 = C_1 \oplus E(IV+1, K),$$
$$P_2 = C_2 \oplus E(IV+2, K), \ldots$$

- Note: CBC also works for random access
  - But there is a significant limitation…

MAC Computation

- MAC computation (assuming N blocks)

$$C_0 = E(IV \oplus P_0, K),$$

$$C_1 = E(C_0 \oplus P_1, K),$$

$$C_2 = E(C_1 \oplus P_2, K), \ldots$$

$$C_{N-1} = E(C_{N-2} \oplus P_{N-1}, K) = MAC$$

- Send $IV, P_0, P_1, \ldots, P_{N-1}$ and MAC
- Receiver does same computation and verifies that result agrees with MAC
- Both sender and receiver must know K

RSA

- Message M is treated as a number
- To encrypt M we compute
  $C = M^e \bmod N$
- To decrypt ciphertext C compute
  $M = C^d \bmod N$
- Recall that e and N are public
- If Trudy can factor $N = pq$, she can use e to easily find d since $ed = 1 \bmod (p-1)(q-1)$
- So, **factoring the modulus breaks RSA**
  - Is factoring the only way to break RSA?

Diffie-Hellman

- **Public:** g and p
- **Private:** Alice's exponent a, Bob's exponent b

$g^a \bmod p$ →

← $g^b \bmod p$

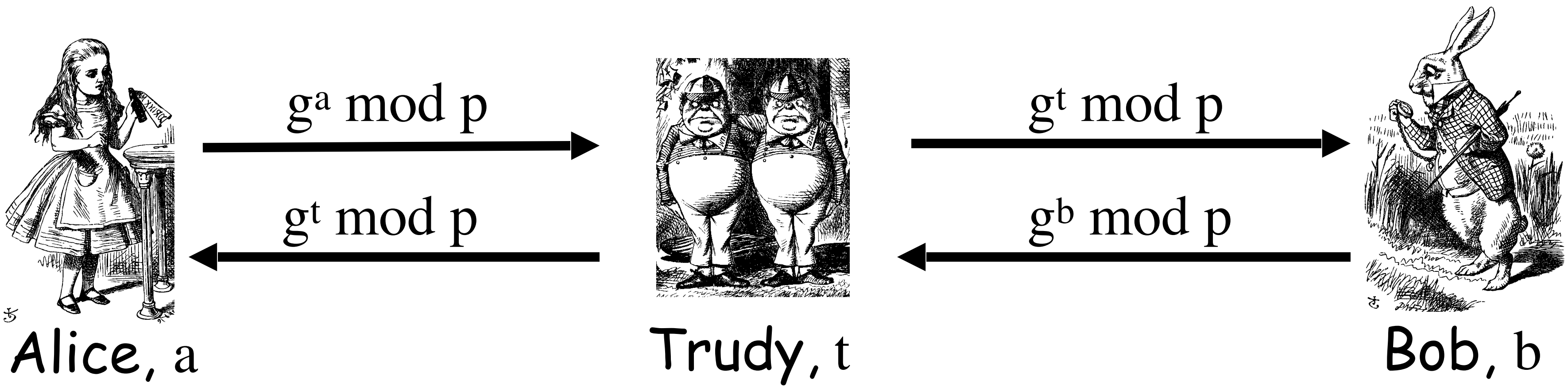
Alice, a

Bob, b

- Alice computes $(g^b)^a = g^{ba} = g^{ab} \bmod p$
- Bob computes $(g^a)^b = g^{ab} \bmod p$
- They can use $K = g^{ab} \bmod p$ as symmetric key

Diffie-Hellman

- Subject to man-in-the-middle (MiM) attack



- Trudy shares secret g^{at} mod p with Alice
- Trudy shares secret g^{bt} mod p with Bob
- Alice and Bob don't know Trudy is MiM

Diffie-Hellman

- How to prevent MiM attack?
  - Encrypt DH exchange with symmetric key
  - Encrypt DH exchange with public key
  - Sign DH values with private key
  - Other?
- At this point, DH may look pointless…
  - …but it's not (more on this later)
- You **MUST** be aware of MiM attack on Diffie-Hellman

Public Key Certificate

- Digital **certificate** contains name of user and user's public key (possibly other info too)
- It is **signed** by the issuer, a **Certificate Authority** (CA), such as VeriSign

  M = (Alice, Alice's public key), $S = [M]_{CA}$

  Alice's Certificate = (M, S)

- Signature on certificate is verified using CA's public key

  Must verify that $M = \{S\}_{CA}$

Non-crypto Hash (1)

- Data $X = (X_1, X_2, X_3, \ldots, X_n)$, each X_i is a byte
- Define $h(X) = (X_1 + X_2 + X_3 + \ldots + X_n) \bmod 256$
- Is this a secure cryptographic hash?
- Example: $X = (10101010, 00001111)$
- Hash is $h(X) = 10111001$
- If $Y = (00001111, 10101010)$ then $h(X) = h(Y)$
- Easy to find collisions, so **not** secure…

Non-crypto Hash (2)

- Data $X = (X_0, X_1, X_2, \ldots, X_{n-1})$
- Suppose hash is defined as

$$h(X) = (nX_1 + (n-1)X_2 + (n-2)X_3 + \ldots + 2 \cdot X_{n-1} + X_n) \bmod 256$$

- Is this a secure cryptographic hash?
- Note that

$$h(10101010, 00001111) \neq h(00001111, 10101010)$$

- But hash of $(00000001, 00001111)$ is same as hash of $(00000000, 00010001)$
- Not "secure", but this hash is used in the (non-crypto) application rsync