ENEE 459-C
Computer Security

Message authentication
(continue from previous lecture)

UNIVERSITY OF
MARYLAND

Last lecture

- Hash function
- Cryptographic hash function
- Message authentication
  - with hash function (attack?)
  - with cryptographic hash function (attack?)

Find collisions for crypto-hashes?

- The brute-force birthday attack aims at finding a collision for a cryptographic function h
  - Randomly generate a sequence of plaintexts $X_1, X_2, X_3, \ldots$
  - For each X_i compute $y_i = h(X_i)$ and test whether $y_i = y_j$ for some $j < i$
  - Stop as soon as a collision has been found
- If there are m possible hash values, the probability that the i-th plaintext does not collide with any of the previous $i - 1$ plaintexts is $1 - (i - 1)/m$
- The probability F_k that the attack fails (no collisions) after k plaintexts is

$$F_k = (1 - 1/m)\,(1 - 2/m)\,(1 - 3/m)\ \ldots\ (1 - (k - 1)/m)$$

- Using the standard approximation $1 - x \approx e^{-x}$

$$F_k \approx e^{-(1/m\ +\ 2/m\ +\ 3/m\ +\ \ldots\ +\ (k-1)/m)} = e^{-k(k-1)/2m}$$

- The attack succeeds with probability p when $F_k = 1 - p$, that is,

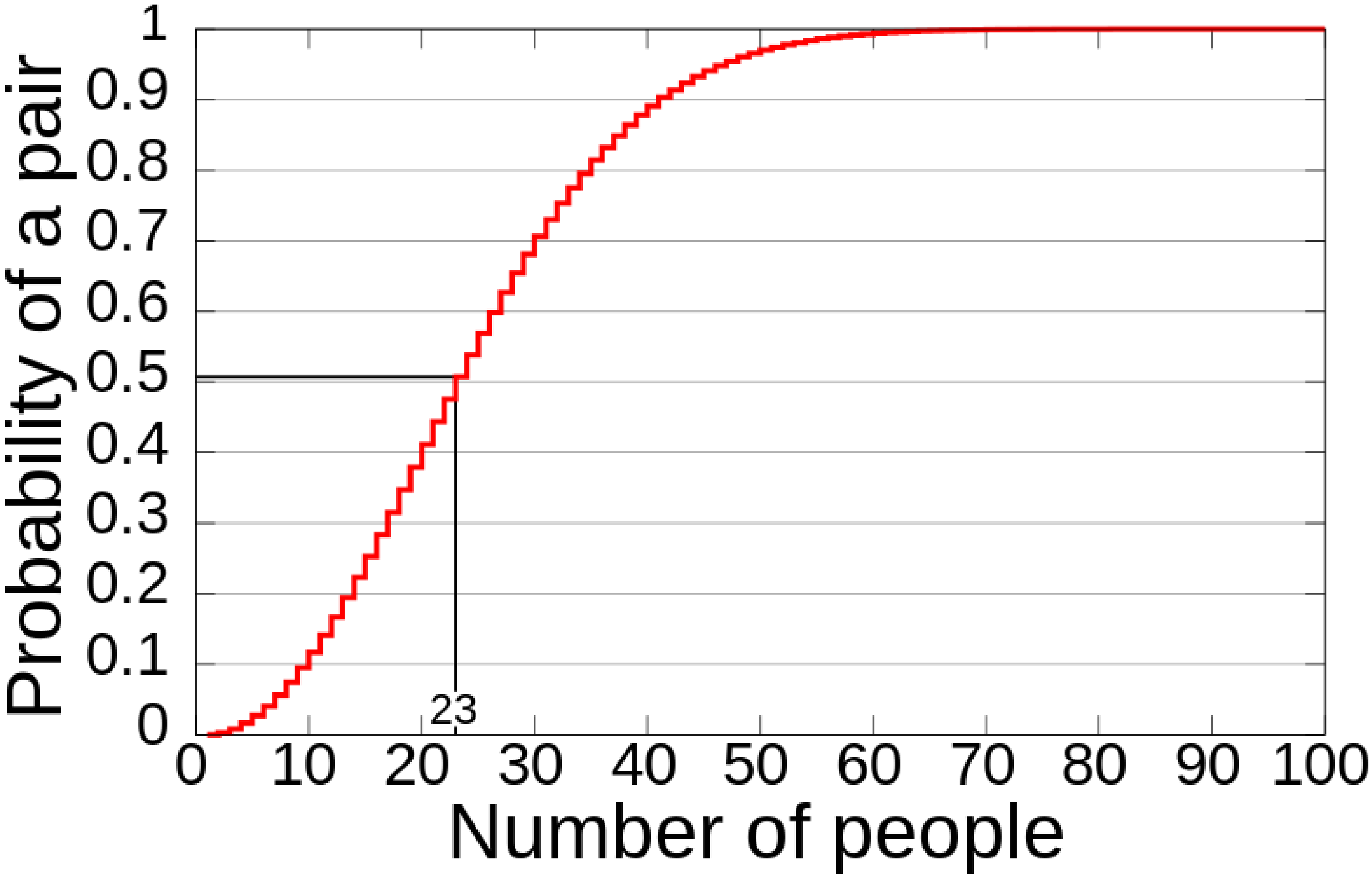
$$e^{-k(k-1)/2m} = 1 - p$$

- For p=1/2

$$k \approx 1.17\ m^{1/2}$$

- For m = 365, p=1/2, k is around 24

Birthday attack



Applications: Online Bid Example

- Suppose Alice, Bob, Charlie are bidders
- Alice plans to bid A, Bob B and Charlie C
  - They do not trust that bids will be secret
  - Nobody willing to submit their bid
- Solution?
  - Alice, Bob, Charlie submit **hashes** h(A),h(B),h(C)
  - All hashes received and posted online
  - Then bids A, B and C revealed
- Hashes do not reveal bids (which property?)
- Cannot change bid after hash sent (which property?)

Online Bid

- This protocol is not secure!
- A forward search attack is possible
  - Bob computes h(A) for likely bids A
- How to prevent this?
- Alice computes h(A,R), R is random
  - Then Alice must reveal A and R
  - Bob cannot try all A and R

Applications: Securing storage

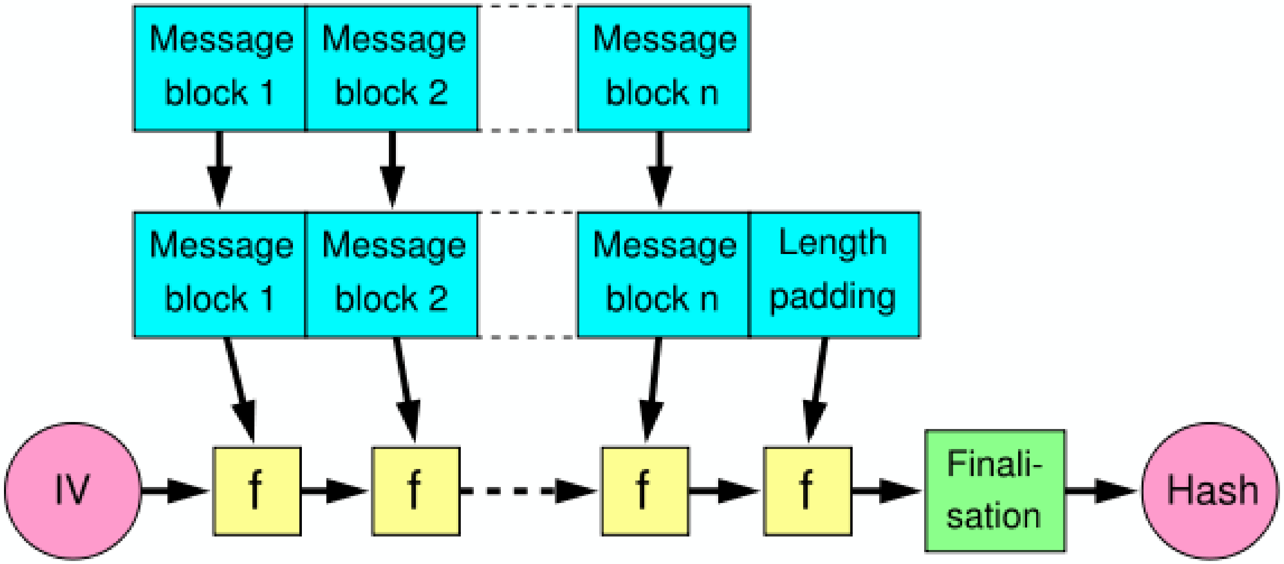
- Bob has files f1,f2,…,fn
- Bob sends to Amazon S3 the hashes
  - h(r||f1),h(r||f2),…,h(r||fn)
  - The files f1,f2,...,fn
- Bob stores randomness r (and keeps it secret)
- Every time Bob **reads** a file f1, he also reads h(r||fi) and verifies
- Any problems with **writes**?

Well Known Hash Functions

- MD5
  - output 128 bits
  - collision resistance completely broken by researchers in China in 2004
- SHA1
  - output 160 bits
  - considered insecure for collision resistance
- SHA2 (SHA-224, SHA-256, SHA-384, SHA-512)
  - outputs 224, 256, 384, and 512 bits, respectively
  - No real security concerns yet
- SHA3
  - Recently proposed
  - Not meant to replace SHA2

Merkle-Damgard Construction for Hash Functions

- Message is divided into fixed-size blocks and padded
- Uses a compression function f, which takes a chaining variable (of size of hash output) and a message block, and outputs the next chaining variable
- Final chaining variable is the hash value



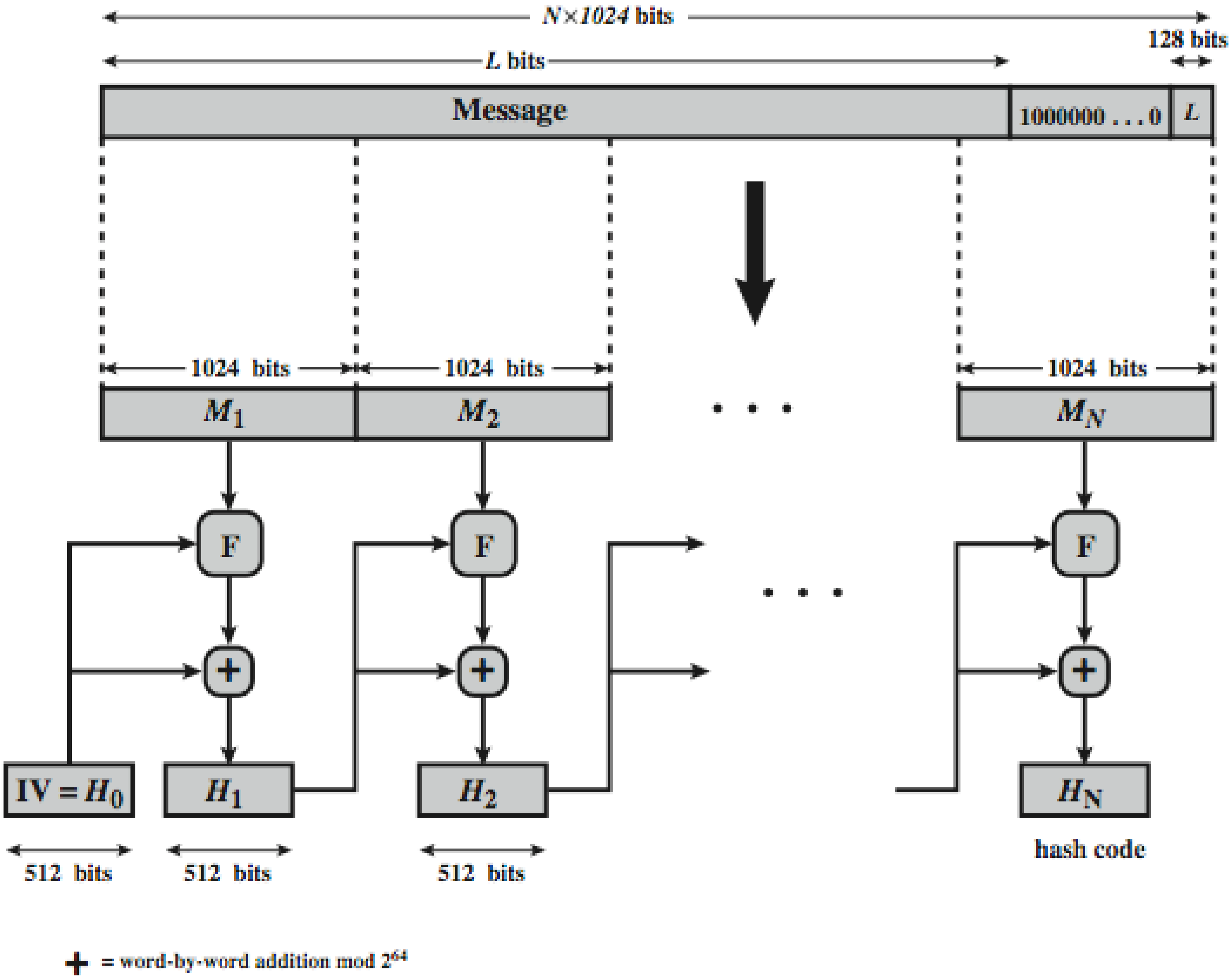
Merkle's meta-method

- any collision resistant compression function f can be extended to a CRHF
- Merkle's meta-method provides an efficient way to construct CRHF from f
  - n bit output, r bit chain variable
  - collision for h would imply collision for f for some stage i

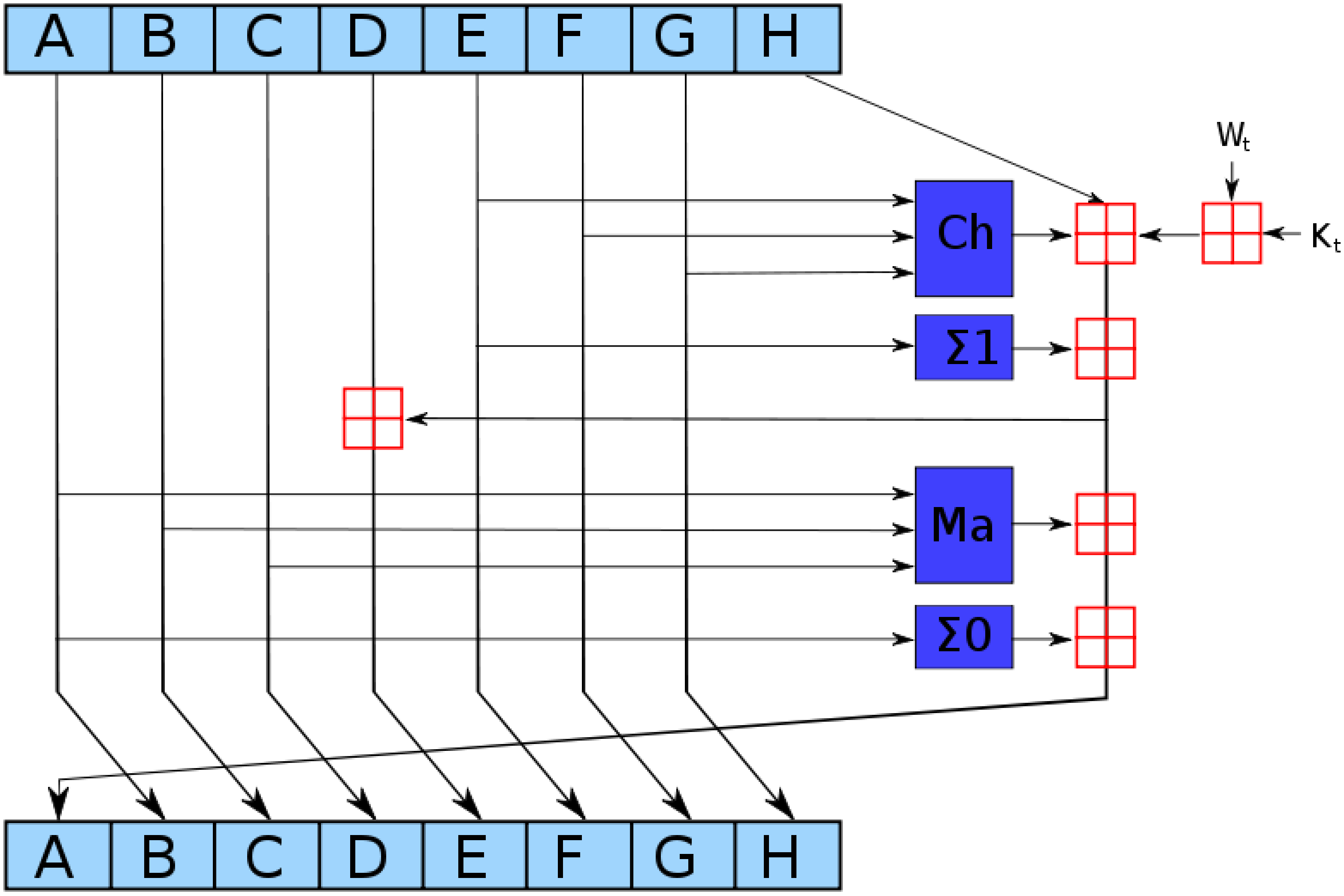
Message-Digest Algorithm 5 (MD5)

- Developed by Ron Rivest in 1991
- Uses 128-bit hash values
- Still widely used in legacy applications although considered insecure
- Various severe vulnerabilities discovered
- Collisions found by Marc Stevens, Arjen Lenstra and Benne de Weger

SHA-2 overview



SHA-2 internals



Limitation of Using Hash Functions for Authentication

- Require an authentic channel to transmit the hash of a message
  - Without such a channel, it is insecure, because anyone can compute the hash value of any message, as the hash function is public
  - Such a channel may not always exist
- How to address this?
  - use more than one hash functions
  - use a key to select which one to use

Hash Family

- A hash family is a four-tuple $(\mathcal{X},\mathcal{Y},\mathcal{K},\mathcal{H})$, where
  - $\mathcal{X}$ is a set of possible messages
  - $\mathcal{Y}$ is a finite set of possible message digests
  - $\mathcal{K}$ is the keyspace
  - For each $K \in \mathcal{K}$, there is a hash function $h_K \in \mathcal{H}$. Each h_K: $\mathcal{X} \to \mathcal{Y}$
- Alternatively, one can think of $\mathcal{H}$ as a function $\mathcal{K} \times \mathcal{X} \to \mathcal{Y}$

Message Authentication Code

- A MAC scheme is a hash family, used for message authentication
- $MAC(K,M) = H_K(M)$
- The sender and the receiver share secret K
- The sender sends $(M, H_k(M))$
- The receiver receives (X,Y) and verifies that $H_K(X)=Y$, if so, then accepts the message as from the sender
- To be secure, an adversary shouldn't be able to come up with (X',Y') such that $H_K(X')=Y'$.

Security Requirements for MAC

- Resist the Existential Forgery under Chosen Plaintext Attack
  - Challenger chooses a random key K
  - Adversary chooses a number of messages M_1, M_2, .., M_n, and obtains $t_j=MAC(K,M_j)$ for $1 \le j \le n$
  - Adversary outputs M' and t'
  - Adversary wins if $\forall j$ $M' \neq M_j$, and $t'=MAC(K,M')$

Constructing MAC from Hash Functions

- Let h be a one-way hash function

- MAC(K,M) = h(K || M), where || denote concatenation
  - Insecure as MAC
  - Because of the Merkle-Damgard construction for hash functions, given M and t=h(K || M), adversary can compute M'=M||… and t', such that h(K||M') = t'

HMAC: Constructing MAC from Cryptographic Hash Functions

$$HMAC_K[M] = Hash[(K^+ \oplus opad) \,||\, Hash[(K^+ \oplus ipad) || M)]]$$

- K^+ is the key padded (with 0) to B bytes, the input block size of the hash function
- ipad = the byte 0x36 repeated B times
- opad = the byte 0x5C repeated B times.

At high level, $HMAC_K[M] = H(K \,||\, H(K \,||\, M))$

HMAC Security

- If used with a secure hash functions (e.g., SHA-256) and according to the specification (key size, and use correct output), no known practical attacks against HMAC

Randomness is important!

- The keystream in the one-time pad
- The secret key used in ciphers
- The initialization vectors (IVs) used in ciphers

Pseudo-random Number Generator

- Pseudo-random number generator:
  - A polynomial-time computable function f (x) that expands a short random string x into a long string f (x) that appears random
- Not truly random in that:
  - Deterministic algorithm
  - Dependent on initial values
- Anyone who considers arithmetical methods of producing random digits is, of course, in a state of sin.” – John von Neumann
- Objectives
  - Fast
  - Secure

Pseudo-random Number Generator

- Classical PRNGs
  - Linear Congruential Generator
- Cryptographically Secure PRNGs
  - Blum-Micali Generator

Linear Congruential Generator - Algorithm

- Based on the linear recurrence:
  $x_i = a\,x_{i-1} + b \mod m \qquad i \geq 1$

  Where

  x_0 is the seed or start value

  a is the multiplier

  b is the increment

  m is the modulus

  Output

  $(x_1, x_2, \ldots, x_k)$

  $y_i = x_i \mod 2$

  $Y = (y_1 y_2 \ldots y_k)$ ← pseudo-random sequence of K bits

Linear Congruential Generator - Example

- Let $x_n = 3\,x_{n-1} + 5 \bmod 31 \quad n \geq 1$, and $x_0 = 2$
  - 3 and 31 are relatively prime, one-to-one (affine cipher)
  - 31 is prime, order is 30

- Then we have the 30 residues in a cycle:
  - 2, 11, 7, 26, 21, 6, 23, 12, 10, 4, 17, 25, 18, 28, 27, 24, 15, 19, 0, 5, 20, 3, 14, 16, 22, 9, 1, 8, 29, 30

- Pseudo-random sequences of 10 bits
  - when $x_0 = 2$
    01101010001
  - When $x_0 = 3$
    10001101001

Linear Congruential Generator - Security

- Fast, but insecure
  - Sensitive to the choice of parameters a, b, and m
  - Serial correlation between successive values
  - Short period, often $m=2^{32}$ or $m=2^{64}$

Linear Congruential Generator - Application

- Used commonly in compilers
  - Rand()
- Not suitable for high-quality randomness applications
- Not suitable for cryptographic applications
  - Use cryptographically secure pseudo-random number generators

Cryptographically Secure

- Passing the next-bit test
  - Given the first k bits of a string generated by PRBG, there is no polynomial-time algorithm that can correctly predict the next $(k+1)^{th}$ bit with probability significantly greater than ½
  - Next-bit unpredictable

Blum-Micali Generator - Concept

- Discrete logarithm
  - Let p be an odd prime, then $(Z_p^*, \cdot)$ is a cyclic group with order p-1
  - Let g be a generator of the group, then $|<g>| = p-1$, and for any element a in the group , we have $g^k = a \bmod p$ for some integer k
  - If we know k, it is easy to compute a
  - However, the inverse is hard to compute, that is, if we know a, it is hard to compute $k = \log_g a$

- Example
  - $(Z_{17}^*, \cdot)$ is a cyclic group with order 16, 3 is the generator of the group and $3^{16} = 1 \bmod 17$
  - Let k=4, $3^4 = 13 \bmod 17$, which is easy to compute
  - The inverse: $3^k = 13 \bmod 17$, what is k? what about large p?

Blum-Micali Generator - Algorithm

- Based on the discrete logarithm one-way function:
  - Let p be an odd prime, then $(Z_p^*, \cdot)$ is a cyclic group
  - Let g be a generator of the group, then for any element a, we have $g^k = a \bmod p$ for some k
  - Let x_0 be a seed

$$x_i = g^{x_{i-1}} \bmod p \qquad i \geq 1$$

Output

$(x_1, x_2, \ldots, x_k)$

$y_i = 1 \quad \text{if } x_i \geq (p-1)/2$

$y_i = 0 \quad \text{otherwise}$

$Y = (y_1 y_2 \ldots y_k)$ ← pseudo-random sequence of K bits

Blum-Micali Generator - Security

- Blum-Micali Generator is provably secure
  - It is difficult to predict the next bit in the sequence given the previous bits, assuming it is difficult to invert the discrete logarithm function (by reduction)