ENEE 459-C
Computer Security

Message authentication and PKI

UNIVERSITY OF
MARYLAND

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

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 $1 \leq j \leq n$
  - 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:
  $x_i = a x_{i-1} + b \mod m$ $i \geq 1$

  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 $x_n = 3 x_{n-1} + 5 \bmod 31$ $n \geq 1$, 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 $|\langle g \rangle| = p-1$, 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

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

$y_i = 1$ if $x_i \geq (p-1)/2$

$y_i = 0$ 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)

Review of Secret Key (Symmetric) Cryptography

- Confidentiality
  - block ciphers with encryption modes
- Integrity
  - Message authentication code (keyed hash functions)
- Limitation: sender and receiver must share the same key
  - Needs secure channel for key distribution
  - Impossible for two parties having no prior relationship
  - Needs many keys for n parties to communicate

Concept of Public Key Encryption

- Each party has a pair (K, K^{-1}) of keys:
  - K is the **public** key, and used for encryption
  - K^{-1} is the **private** key, and used for decryption
  - Satisfies $\mathbf{D}_{K^{-1}}[\mathbf{E}_K[M]] = M$
- Knowing the public-key K, it is computationally infeasible to compute the private key K^{-1}
  - Easy to check K,K^{-1} is a pair
- The public-key K may be made publicly available, e.g., in a publicly available directory
  - Many can encrypt, only one can decrypt
- Public-key systems aka *asymmetric* crypto systems

Public Key Cryptography Early History

- Proposed by Diffie and Hellman, documented in "New Directions in Cryptography" (1976)
  1. Public-key encryption schemes
  2. Key distribution systems
     - Diffie-Hellman key agreement protocol
  3. Digital signature

- Public-key encryption was proposed in 1970 in a classified paper by James Ellis
  - paper made public in 1997 by the British Governmental Communications Headquarters

- Concept of digital signature is still originally due to Diffie & Hellman

Public Key Encryption Algorithms

- Almost all public-key encryption algorithms use either number theory and modular arithmetic, or elliptic curves
- RSA
  - based on the hardness of factoring large numbers
- El Gamal
  - Based on the hardness of solving discrete logarithm
  - Use the same idea as Diffie-Hellman key agreement