Cryptography Review

CS461 / ECE422 – UIUC SPRING 2016

By Due Chuenchujit

Outline

- Message Integrity
- Message Confidentiality
- Key Exchange
- Public-key Cryptography

Message Integrity: Problem

- Setting: Alice wants to send message *m* to Bob
  - The messenger or the network is not trustworthy
  - Bob wants to make sure that the message he received is actually what Alice sent
- Threat Model:
  - Mallory can see, modify, forge messages
  - Mallory wants to trick Bob into accepting a message that Alice didn't send

Message Integrity: Solution

- Message Authentication Code (MAC)
  - Alice compute a value v = *f*(m) for some function *f*, and send it with the message
  - Bob verifies that v = *f*(m) and only accept the message if this is true
  - Mallory cannot compute v for arbitrary messages

  - Candidate functions for *f*
    - Random lookup table: provably secure but impractical
    - Pseudorandom functions: we don't know if these exists
    - Hash-based MAC

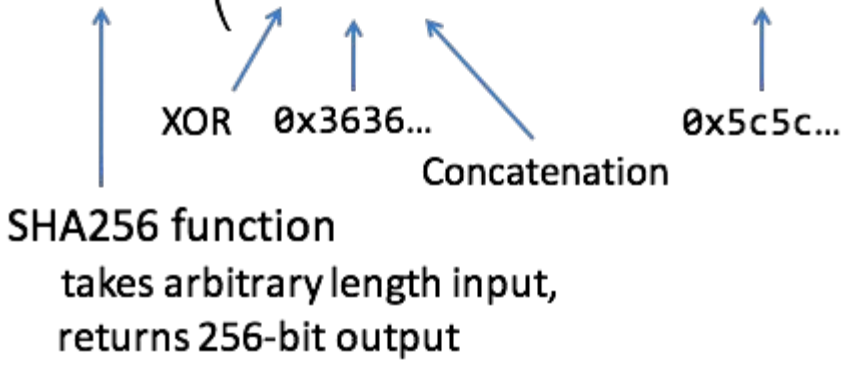
PseudoRandom Function

- The Game
  - Let $\mathbf{f_k}$ be a big family of functions all known to Mallory
  - We flip a coin secretly to get a bit b
  - If b = 0, let **g** be a random function in the family

    If b = 1, let **g** be $\mathbf{f_k}$ for a chosen **k**
  - Mallory chooses **x**; we announce **g(x)**
  - Repeat until mallory says stop
  - Mallory chooses **x**; we announce **g(x);** Mallory guess **b** and wins if he guesses correctly
- **f** is a secure PRF if Mallory cannot do better than guessing.

HMAC-SHA256

$\text{HMAC}_k(m) = \text{SHA256}\left(k \oplus c_1 \parallel \text{SHA256}\left(k \oplus c_2 \parallel m\right)\right)$



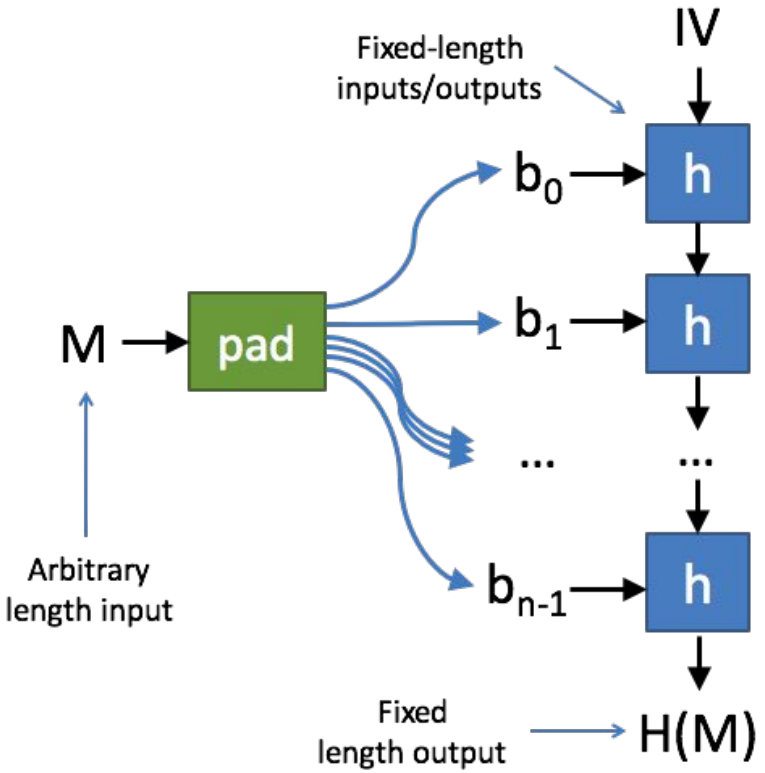
Kerckhoffs's Principle:

Don't rely on secret functions. Use a secret key to choose from a function family

SHA256

- A cryptographic hash function
- Input: arbitrary length data
- Output: 256 bits
- uses Merkle–Damgård Construction

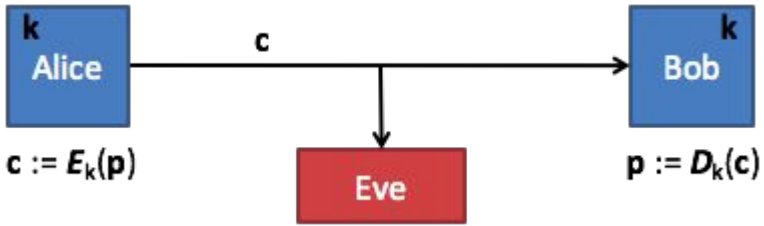
Merkle–Damgård Construction



- Arbitrary-length input
- Fixed-length output
- Built from fixed-size “compression function”
- Message is padded to multiple of the compression function’s size then processed in blocks.
- Suffers length-extension attack

Message Confidentiality: Problem

- Goal: keep message content secret from eavesdroppers



- Terminology:
  - p - plaintext
  - c - ciphertext
  - k - secret key
  - E - encryption function
  - D - decryption function

Classical Cryptography

- Caesar Cipher
  - Replace a character with one a fixed number of placed down the alphabet
- Vigenere Cipher
  - Encrypt successive letters using a sequence of Car ciphers determined by letters of a keyword
- Substitution Cipher
  - Replace each character with another character using a pre-determined mapping

One-time pad (OTP)

- A very long string of random bits (at least as long as the message) is generated to be use as a key. The pad should never be reused.
  - Encryption: $c_i = p_i \text{ xor } k_i$
  - Decryption: $p_i = c_i \text{ xor } k_i$
- Stream Cipher: use pseudorandom generator instead of a truly random pad
  - Pseudorandom Generator (PRG) takes a small seed and generate a long sequence of bits that are “as good as random”. Secured if the output is indistinguishable from random
  - Alice and Bob agreed on a shared secret *k*
  - Alice and Bob each use *k* to seed the PRG
  - PRG will generate the same pad for both party
- Why XOR?

PseudoRandom Generator

- The Game
  - We flip a coin secretly to get a bit b
  - If b = 0, let **s** be a truly random stream
    If b = 1, let **s** be $\mathbf{g_k}$ for a random secret **k**
  - Mallory can see as much of s as he wants
  - Mallory guess **b** and wins if he guesses correctly
- **g** is a secure PRG if Mallory cannot do better than guessing.

Block Ciphers

- Functions that encrypts fixed-size blocks with a reusable key
- Decrypt by using inverse function with the same key
- A block cipher is NOT a pseudorandom function
- Minimal properties of a good block cipher
  - Highly nonlinear (“confusion”)
  - Mixes input bits together (“diffusion”)
  - Depends on the key
- AES
  - variable key size and block size; message has to be padded to multiple of block size
  - operates in rounds
  - operate in different modes:
    - Encrypted codebook (ECB) - encrypt each block independently
    - Cipher Block Chaining (CBC) - xor each block with the intermediate result
    - Counter Mode - XOR i^{th} block of message with $\mathbf{E}_k$(message_id||i)

Key Management

- THIS is the hard part!
- Each key should only serves one purpose
  - Do NOT reuse old keys
- Vulnerability of a key increases with
  - The more you use it
  - The more copies you made of it
  - The longer you have been using it
- Protect yourself against compromised old keys
  - Forward secrecy - the key used to protect transmission of data must not be used to derive any additional keys, and if the key used to protect transmission of data is derived from some other keying material, then that material must not be used to derive any more keys

Diffie-Hellman Key Exchange

- Agrees on symmetric key in public channel
- Based on hard math problem
- Protocol:
  - Alice and Bob agrees on modulus g, p
  - Alice chooses a secret integer a, then sends Bob
    $A = g^a \bmod p$
  - Bob chooses a secret integer b, then sends Alice
    $B = g^b \bmod p$
  - Alice computes $s = B^a \bmod p$
    Bob computes $s = A^b \bmod p$
  - s is now the shared secret
- Vulnerability?

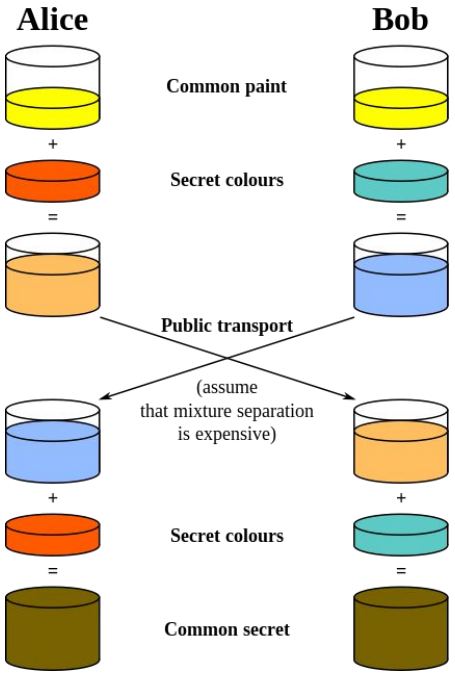


image from https://en.wikipedia.org/wiki/Diffie%E2%80%93Hellman_key_exchange#/media/File:Diffie-Hellman_Key_Exchange.svg

Man In The Middle Attack (MITM)

- D-H can give you secure communication, but you cannot verify who is on the other end!
- How can we defend against this?
  - ~~Hope there isn't an adversary~~
  - Integrate D-H with authentication
  - Use digital signatures

Public-key Cryptography

- Symmetric key cryptography: encryption key == decryption key
- Public-key cryptography: encryption key != decryption key
  - Almost always used by splitting key-pair: keep “private” key secret and publish “public” key
- Security requirement
  - Key-pair generation is computationally easy
  - Encryption and decryption are computationally easy
  - Computationally infeasible to compute private key of a public key
  - Computationally infeasible to recover message given public key and encrypted message

RSA - a re-introduction

- Components of RSA
  - n - the modulus of the keys, created as a product of two large prime numbers (p, q)
  - e - the public key
  - d - the private key
- Key generation
  - Pick two large random primes p, q
  - Compute n = pq
  - pick e = 1 mod (p-1)*(q-1)
  - compute d such that 1 = ed mod (p-1)*(q-1)
- Encryption with public key
  - $\text{encrypted\_text} = \text{plaintext}^{e} \bmod n$
- Decryption with private key
  - $\text{plaintext} = \text{encrypted\_text}^{d} \bmod n$

RSA - a re-introduction

- Is RSA secure?
  - Best known way to compute d from e, n is to factor n into p and q
  - Best known algorithm (General number field sieve) takes more than polynomial time but less than exponential time. Still way too long if we pick large enough primes
- RSA can be use for either confidentiality or integrity
  - Encrypt with public key for confidentiality (only owner of private key can decrypt this message)
  - Encrypt with private key for integrity (only the owner of private key can encrypt this message)
    - also called a digital signature
- RSA Drawback: performance
  - Factor of 1000 more more slower than AES
  - In practice, use for symmetric key exchange or digital signature

Confidentiality + Integrity Exercise

Assume that Alice and Bob have securely distributed their public keys K_A and K_B to each other. Suppose K_a and K_b are the private key of Alice and Bob respectively. They are sending a message m.

1. Using public key cryptography, design a message that enables Bob to verify that the message's integrity has not been violated and that it is from Alice.
    - $m||\{h(m)\}K_a$
2. Using public key cryptography, design a message that protects the confidentiality of the request and ensures that Bob can verify the message's integrity and source.
    - $\{m||\{h(m)\}K_a\}K_B$

Attacking RSA

- Brute force: trying all possible private keys
- Mathematical attacks: factoring
  - “weak” RSA keys can be factor with a reasonable amount of time (MP3.2)
- Timing attacks: using the running time of decryption
- Hardware-based fault attack: induce faults in hardware to generate digital signature
- Chosen ciphertext attack
  - Given access to a decryption oracle, recover m without feeding c to the oracle directly