Security, part 1

The tools

Announcements

- HW2 due
- HW3, Project 3 both coming soon
- Max Krohn guest lecture next Tuesday

Last time

- DNS: the Domain Name System
  - A global distributed map
    - names → IP addresses
    - IP addresses → names
    - other information (e.g., domain → mail server)
- Scalability through
  - Hierarchy of servers
  - Caching, reduced consistency

Today: Security, part 1

- General background
- Cryptography
  - public-key and private-key cryptography
  - DES
- Cryptographic hashing
- Digital signatures

Distributed systems and security

- Distributed systems provide access to objects, data, and functions to authorized users and processes
- Security goals:
  - Authenticate users/processes
    - Do not provide services to unauthenticated users
  - Privacy
    - Keep interactions with the system private
  - Availability
    - Do not allow unauthorized users to prevent access by authorized users

Security models

- What might an enemy/threat do to attack the system?
  - Send messages to server, trying to emulate a client
  - Send messages to client, trying to emulate the server
  - Copy, inject, or otherwise alter messages as part of a communication channel
    - Man-in-the-middle attack
    - Replay attack
    - Denial-of-service attacks

The network

- Provides only simple message services
  - Messages are unreliable
  - Data is public, no privacy
  - Sender IP address is forgeable

Cryptography

- A tool to provide authentication and privacy
- Typically:

  The "ciphertext"

  The "plaintext" message to encrypt

  – An encryption function: $c = E(m)$
  – A decryption function: $m = D(c)$
- Two basic types of cryptosystems:
  – Public-key cryptography
  – Private-key cryptography

Public-key cryptography

- Two keys:
  - A public key K_{pub}
    - You give your public key to everyone you might want to communicate with
  - A private key K_{priv}
    - You keep your private key as a secret

Public-key cryptography

- Typically:
  - Public key needed to encrypt a message:

$$c = E(K_{pub}, m)$$

  - Private key needed to decrypt a message:

$$m = D(K_{priv}, c)$$

Public-key cryptography

- The keys are large
  - Typically 1024 or 2048 bits
- The algorithms are slow compared to private-key crypto

Public-key cryptography

- Security depends on hardness of determining the private key from the public key
  - E.g., for RSA, can determine the private key from the public key only if we can factor large numbers (product of two large primes)
    - Thus, breaking RSA should be as hard as factoring
    - We haven’t proven that factoring is hard, but a thousand years worth of mathematicians haven’t solved the problem yet!

Private-key cryptography

- One shared key, *K*
  - Anyone with *K* can read messages encrypted with *K*
- Typically:

$$c = E(K, m)$$

$$m = D(K, c)$$

e.g., A one-time pad

- Choose K uniformly at random, with $|K| \geq |m|$

XOR

$$c = E(K, m) = m \oplus K$$

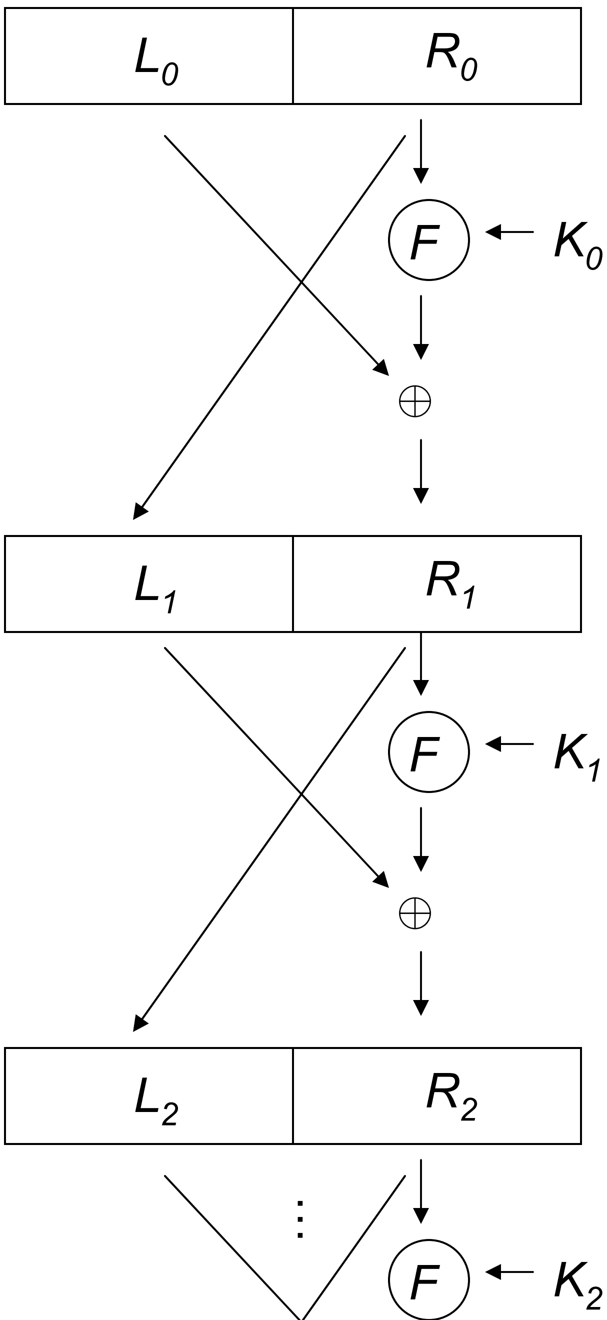
$$m = D(K, c) = c \oplus K$$

- Ciphertext c gives no information about m, if K used only once
- Impractical since $|K|$ must be $\geq |m|$

Feistel block ciphers

- The basis of several popular private-key cryptosystems
- Divide *m* into left half L_0 and right half R_0
- Given L_i and R_i, apply Feistel cipher to get L_{i+1} and R_{i+1}

Feistel block ciphers



- For some function F and secret key K_i:

$$L_{i+1} = R_i$$
$$R_{i+1} = L_i \oplus F(R_i, K_i)$$

- Repeated rounds "confuse and diffuse" bits of original message
  - Not provably secure, but seems(!) hard to invert without knowing each K_i

Inverting a Feistel block cipher

- Easy if you know each K_i:

$$R_i = L_{i+1}$$

Can compute each R_i and L_i using R_{i+1} and L_{i+1}

$$R_{i+1} = L_i \oplus F(R_i, K_i)$$

$$R_{i+1} \oplus F(R_i, K_i) = L_i \oplus F(R_i, K_i) \oplus F(R_i, K_i)$$

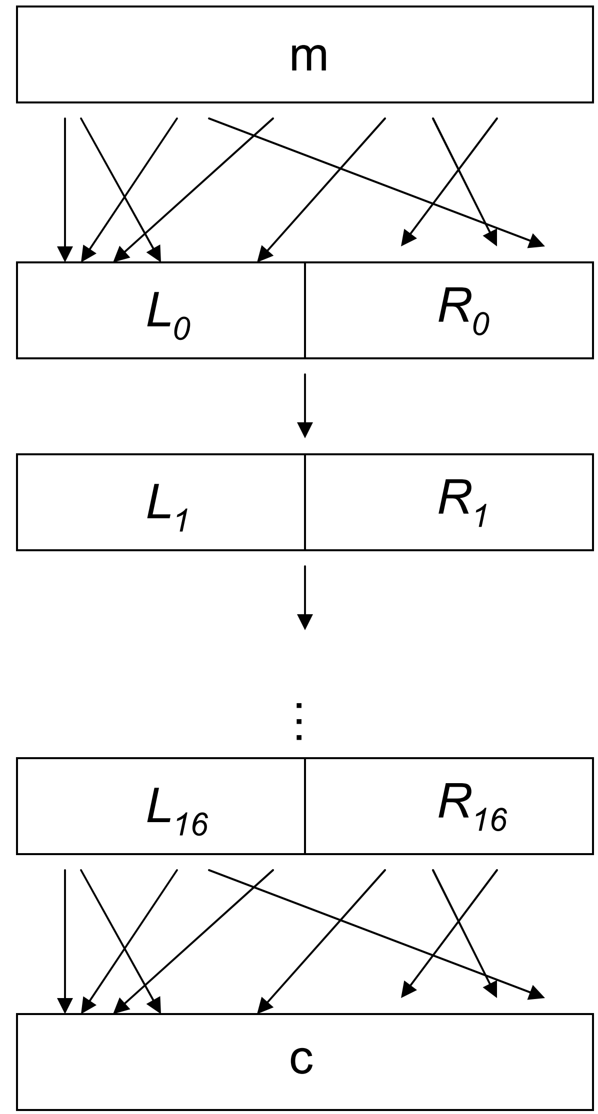
$$R_{i+1} \oplus F(R_i, K_i) = L_i$$

$$L_i = R_{i+1} \oplus F(L_{i+i}, K_i)$$

- Note:
  - Can invert the cipher without inverting F
  - Inversion is essentially the same as computing the cipher, but using the keys in the reverse order

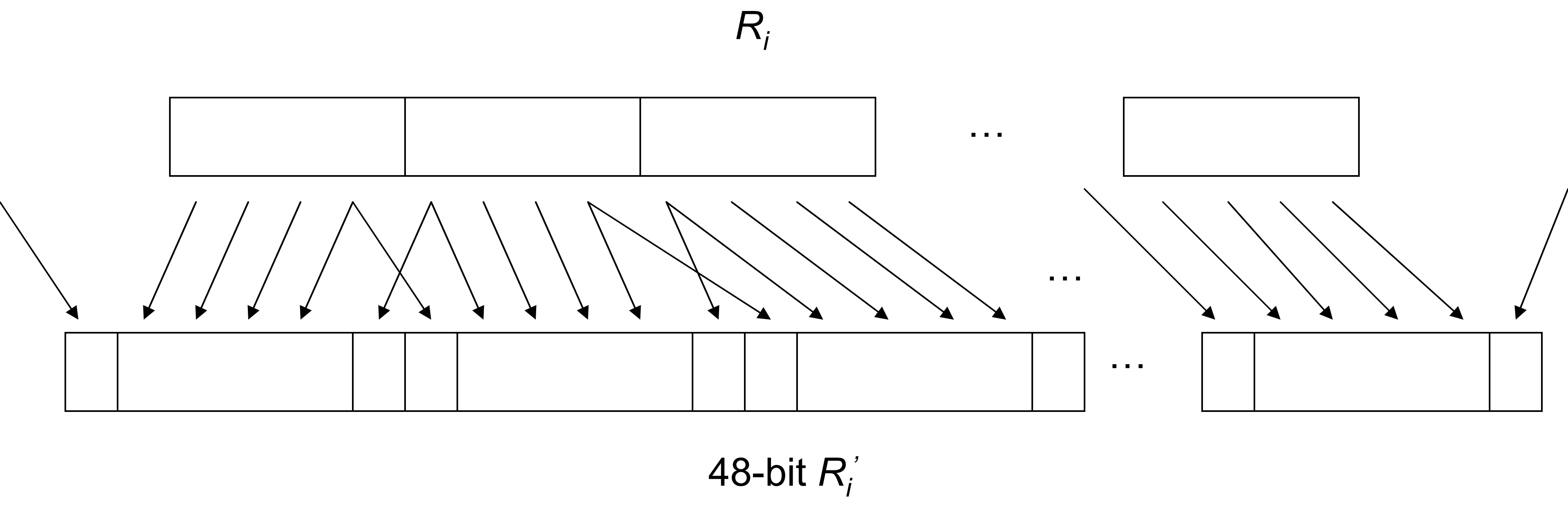
DES: Data Encryption Standard

- A 64-bit block cipher
  - 2 permutation rounds
  - 16 Feistel-based rounds
- 56-bit secret key *K*
- Developed in early 1970s by IBM and NSA
  - Considered obsolete now because the key size is too small



DES's Feistel function $F(R_i, K_i)$

- Step 1: expand R_i from 32 bits to 48 bits
  - Break R_i into 4-bit blocks
  - Copy bits from adjacent blocks



DES's Feistel function $F(R_i, K_i)$

- Step 2: Create K_i from 56-bit secret key K
  - Choose 48 bits from K using a fixed, pre-defined series of permutations and circular rotations
- Step 3: Compute 48-bit $R_i' \oplus K_i$

DES's Feistel function $F(R_i, K_i)$

- Step 4: Break $R_i' \oplus K_i$ into 6-bit blocks
  - Use fixed 6-bit-to-4-bit mappings ("Substitution boxes" or "S-boxes") to compute 32-bit R_{i+1}
- NSA helped IBM choose "good" S-boxes
  - ~15 years later a "new" cryptographic attack method was discovered…but the S-boxes had been designed to resist the attack!

Cryptographic hash functions

- Goal: summarize (or hash) long message *m* into a short digest *h*: $h = H(m)$
  - Given *h*, cannot find *m*
  - Given $H(m)$, cannot find *m'* such that $H(m) = H(m')$
- Modern cryptographic hash functions yield a 128-to-512-bit hash
  - MD5: 128 bits
  - SHA1: 160 bits
  - SHA2: 224 – 512 bits

Cryptographic hash functions

- Typically use “confuse and diffuse” techniques much like private-key crypto
  - Actually, outputting last ciphertext blocks of an encrypted message is not a bad hash technique
- 3-5x faster than private-key cryptography

Digital signature goals

- Authentication
  - Prove that a message has not been altered
- Unforgeability
  - Prove that the message was created by a specific person (a.k.a. the principal)
- Non-repudiation
  - Once a message is signed, the principal cannot deny that they signed the message

Signatures with public-key crypto

- One option (not used in practice):
  - Encrypt with private key to sign a message:

    $s = E(K_{priv}, m)$

    Send m,s
  - Decrypt with public key to verify the signature:

    $m' = D(K_{pub}, s)$

    Check that $m == m'$
  - Because private key is not shared, the signature is unforgeable and unrepudiable
  - Because public key is shared, anyone can verify the signature
  - A problem: public-key cryptography is slow

Signatures with public-key crypto

- An improvement:
  - Hash the message with a cryptographic hash function first, sign the hash:

    $h = H(m)$

    $s = E(K_{priv}, h)$

    Send *m,s*
  - Use the hash function and public key to verify the signature:

    $h = H(m)$

    $h' = D(K_{pub}, s)$

    Check that $h == h'$
  - Cryptographic hash functions are often 30-100x faster than public-key cryptography
    - Public-key crypto needed just to sign the short hash
  - Hash function must be cryptographic to prevent attacker from replacing *m* with *m'* such that $H(m') == H(m)$

Signatures with private-key crypto

- Using a cryptographic hash function *H* and shared private key *K*:

  $s = H(m + K)$

  Send *m,s*

  Bit-string append (not addition)

- To verify:

  Compute $s' = H(m + K)$

  Check $s == s'$

- Very fast: no encryption/decryption needed
- A problem: need to reveal private key to verify the signature