Today

- Secure design principles
- Cryptography applications (besides encryption/decryption)

Security Properties

- Assume a system that processes requests from agents, and a request comes in. A secure system must be able to answer the following questions before performing the required action.
- **Authenticity**: is the agent’s claimed identity authentic?
- **Integrity**: is the request actually coming from the agent?
- **Authorization**: has a proper authority granted permission to this agent to perform this action?
- These three combined are called the principle of complete mediation.

Security Threats

- A secure system must be able to defend against the following threats.
- **Unauthorized information release**
  - An unauthorized person accesses information.
- **Unauthorized information modification**
  - An unauthorized person changes information.
- **Unauthorized denial of use**
  - An adversary prevents an authorized user from reading or modifying information.

Designing Secure Systems

- Your system is only as secure as your weakest component!
- One must demonstrate that the system is protected from every possible threat.
- Is the system secure?
  - Insecure: just needs to discover one example security hole.
  - Secure: must show there’s no security hole at all.
  - I don’t know: “We don’t know of any remaining security holes.”

Design Principles

- **Open design principle**
  - Let anyone comment on the design. You need all the help you can get.
  - Closed designs have been historically proven to almost always lead to flaws.
  - Open vs. closed debate has been going on for ages (e.g., open vs. closed door lock design).
- **Minimize secrets**
  - Because they probably won’t remain secret for long.
  - If secrets are minimized, when they are compromised, they’re easier to replace.
- **Economy of mechanism**
  - The less there is, the more likely you will get it right.
  - E.g., having 10,000 lines of security critical code vs. 1,000 lines of security critical code.
Design Principles

- Minimize common mechanism
  - Shared mechanisms provide unwanted communication paths.
  - E.g., putting a new feature in the kernel (shared by all users) vs. putting it in a library (per application): choose the latter.

- Fail-safe defaults
  - Most users won’t change them, so make sure that defaults do something safe.
  - E.g., default Wi-Fi router passwords: a lot of users don’t change them.

- Least privilege principle
  - Don’t store lunch in the safe with the jewels.
  - Give a program as fewest privileges as possible, as accidents can cause a lot of damage.
  - E.g., no need to run applications with sudo.

Safety Net Approach

- Never assume the design is right.

- Two principles
  - Be explicit
  - Design for iteration

- Be explicit
  - Make all assumptions explicit so they can be reviewed.
  - E.g., buffer overrun using: `gets(character array reference string_buffer)`

- Design for iteration
  - Assume you will make errors and prepare to iterate the design.

TCB (Trusted Computing Base)

- Applying the economy of mechanism principle together with the safety net approach
  - Organize a system design into two kinds of modules: Untrusted modules and trusted modules.
  - The correctness of the untrusted modules should not affect the security of the whole system.
  - The trusted modules must work correctly to make the system secure.
  - The collection of trusted modules are called the trusted computing base (TCB).
  - It is important to minimize the size of the TCB (the economy of mechanism principle).

Secure System Model

- A guard is commonly called a reference monitor.

CSE 486/586 Administriivia

- PA4 deadline: 5/10
- Survey & course evaluation
  - Survey: https://forms.gle/eg1wHN2G8S3Q1z3a9
  - Course evaluation: https://www.smartevals.com/login.aspx?c=buffalo
- If both have 80% or more participation,
  - For each of you, I’ll take the better one between the midterm and the final, and give the 30% weight for the better one and the 20% weight for the other one.
  - (Currently, it’s 20% for the midterm and 30% for the final.)
- No recitation this week; replaced with office hours

Cryptography

- Comes from Greek word meaning “secret”
  - Primitives also can provide integrity, authentication
- Cryptographers invent secret codes to attempt to hide messages from unauthorized observers
- Modern encryption:
  - Algorithm public, key secret and provides security
  - May be symmetric (secret) or asymmetric (public)
- Cryptographic algorithms goal
  - Given key, relatively easy to compute
  - Without key, hard to compute (invert)
  - The strength of security often based on the length of a key (to protect against brute-force guesses)
Window of Validity

• **The minimum time** to compromise a cryptographic algorithm.
• **Problem**
  – It can be shorter than the lifetime of your system.
• **Example**
  – SHA-0 was published in 1993.
  – A possible weakness was found in the algorithm and replaced in 1995 with SHA-1.
  – A way to compromise it was published in 2004.
  – A way to compromise SHA-1 was published in 2017.
• A system designer needs to be prepared to update their crypto function, perhaps more than once.

Three Types of Functions

• Cryptographic hash functions
  – Zero keys
• Secret-key functions
  – One key
• Public-key functions
  – Two keys

Cryptographic Hash Functions

• Take message, \( m \), of arbitrary length and produces a smaller (short) number, \( h(m) \)
• **Properties**
  – Easy to compute \( h(m) \)
  – Pre-image resistance (strong collision): Hard to find an \( m \), given \( h(m) \)
    » “One-way function”
  – Second pre-image resistance (weak collision): Hard to find two values that hash to the same \( h(m) \)
    » E.g. discover collision: \( h(m) = h(m') \) for \( m \neq m' \)
  – Often assumed: output of hash fn’s “looks” random

Symmetric (Secret) Key Crypto

• Also: “conventional / private-key / single-key”
  – Sender and recipient share a common key
  – All classical encryption algorithms are private-key
• Was only type of encryption prior to invention of public-key in 1970’s
  – Most widely used
• **Two requirements**
  – Strong encryption algorithm
  – Secret key must be known only to the sender/receiver
• **Goal**: Given key, generate 1-to-1 mapping to ciphertext that looks random if key unknown
  – Assume algorithm is known (no security by obscurity)
  – Implies secure channel to distribute key

Symmetric Key Crypto

![Symmetric Key Crypto Diagram]

Public (Asymmetric) Key Crypto

• Public invention Diffie & Hellman in 1976
  – Known earlier to classified community
• Involves two keys
  – **Public key**: can be known to anybody, used to encrypt and verify signatures
  – **Private key**: should be known only to the recipient, used to decrypt and sign signatures
  – Avoiding key distribution: secure communication without having to trust a key distribution center with your key
• **Asymmetric**
  – Can encrypt messages or verify signatures w/o ability to decrypt: breaks or create signatures
  – If “one-way function” goes \( c \leftarrow F(m) \), then public-key encryption is a “trap-door” function:
    » Easy to compute \( c \leftarrow F(m, pub) \)
    » Hard to compute \( m \leftarrow F^{-1}(c) \) without knowing priv
    » Easy to compute \( m \leftarrow F^{-1}(c, priv) \) by knowing priv
Public (Asymmetric) Key Crypto

Application: Storing Passwords

- **Password hashing**
  - A password system doesn’t store plaintext passwords.
  - All passwords are hashed and the hashes are stored.
  - Concerned with insider attacks, e.g., system admins.
- **Must compare typed passwords to stored passwords**
  - *Does hash (typed) == hash (password)?*
  - A salt is effectively a random number added to input.
  - It is stored together with the generated hash.
  - Avoids precomputation of all possible hashes in “rainbow tables” (available for download from file-sharing systems)
  - No need to be a secret: with a salt, pre-computation is not possible.

Application: Secure Digest

- A secure digest is a summary of a message.
  - A fixed-length that characterizes an arbitrary-length message
  - Typically produced by a cryptographic hash function, e.g., SHA-256
- E.g., Open-source Android Repo command verification

Application: MAC

- **MAC (Message Authentication Code)**
  - Uses symmetric crypto & hashing
  - Prevents sender masquerading & message tampering (but this is not about confidentiality)
- Answering the following two questions
  - Who sent the message (authenticity)?
  - What the sender says (integrity)?
- **Sender (sending a message M)**
  - Computes a message digest: SHA1(M)
  - Encrypts the message digest: H = AESKSHA1(M)
  - Sends <M, H>
- **Receiver**
  - Receives <M, H>
  - Computes a message digest: H’ = SHA1(M)
  - Decrypts the message digest: H” = AESK(H’)
  - Checks the equality: H” == H

Application: Digital Signature

- **Similar to MAC**
  - Verifies a message or a document is an unaltered copy of one produced by the signer
  - Both integrity & authenticity
  - Uses asymmetric crypto & hashing
- **Signer (writing a document, M)**
  - Computes a message digest: SHA1(M)
  - Signs the digest with the private key: H = RSAprivate SHA1(M)
  - Posts the message & the signature: <M, H>
- **Verifier**
  - Obtains <M, H>
  - Computes a message digest: H’ = SHA1(M)
  - Decrypts the signature with the public key: RSApublic(H)
  - Verifies the equality: RSApublic(H) == H’

HTTPS

- A use case for digital signatures
- **Threat model**
  - Eavesdropper listening on conversation (confidentiality)
  - Man-in-the-middle modifying content (integrity)
  - Adversary impersonating desired website (authentication, and confidentiality)
- **Enter HTTP-S**
  - HTTP sits on top of secure channels
  - All (HTTP) bytes written to secure channel are encrypted and authenticated
Encrypted Communication

Hey, I want to be more secure.

Sure, use this public key and encrypt your traffic.

Key: f-pub

What is wrong with this?
- How do you know you’re actually talking to Facebook and f-pub belongs to Facebook?

Digital Certificates

- A digital certificate is a statement signed by a third party principal, and can be reused.
  - A digital certificate has a public key, its organization, and a signature by a third party that attests that the public key belongs to the organization.
  - A third-party example: Verisign Certification Authority (CA)
- Example
  - Facebook sends its public key to Verisign.
  - Verisign uses its private key to digitally sign Facebook’s public key. This says that Verisign attests that the public key belongs to Facebook.
  - Verisign gives the signature to Facebook.
  - When you ask Facebook for its public key, Facebook sends you its public key as well as the signature (from Verisign).
  - You verify that the signature is from Verisign. If successful, you trust that the public key belongs to Facebook.

Digital Certificates

- Question still remains: how do you verify if the signature is from Verisign?
  - Verisign uses its private key to sign. What do you need to verify this signature?
  - You need its public key to verify the signature.
  - Full circle: in order to verify Facebook’s public key (which Verisign attests), you need to acquire Verisign’s public key and verify it.
- Chain of trust
  - You don’t trust Facebook’s public key, so Facebook says “trust Verisign’s public key.”
  - But in order to trust Verisign’s public key, some other trusted entity needs to verify the trustworthiness of Verisign’s public key.
  - You can establish a chain of trust that way.
  - The end of the chain is called the root of trust.

Digital Certificates

- This trust comes from your OS.
  - Your OS pre-stores Verisign’s public keys & certificates (self-signed by Verisign).
  - Use Verisign’s public key to verify Verisign’s signature for Facebook’s public key.
  - As long as you trust your OS, you trust Verisign’s public key as well as Facebook’s.
  - You can manually install other company’s certificates that you trust.
  - You can also self-sign your certificate, e.g., for testing HTTPS configuration.

X.509 Certificates

- The most widely used standard format for certificates
- Format
  - Subject: Distinguished Name, Public Key
  - Issuer: Distinguished Name, Signature
  - Period of validity: Not Before Date, Not After Date
  - Administrative information: Version, Serial Number
  - Extended Information
- Binds a public key to the subject
  - A subject: person, organization, etc.
  - The binding is in the signature issued by an issuer.
  - You need to either trust the issuer directly or indirectly (by establishing a root of trust).
X.509 Certificates

Certificate Pinning
- An application (e.g., a mobile app) frequently uses a back-end server.
- To use HTTPS, the server typically sends a certificate which the application verifies.
- Problem
  - A user can be tricked to install rogue certificates that verify an adversary’s server certificates.
  - E.g., a public Wi-Fi connection redirects you to a website and asks you to install a certificate.
  - The Iranian gov. is suspected to compromise a certificate authority and issued rogue certificate for Google.
- Certificate pinning
  - An application pre-stores a few certificates that it expects to receive from its server.

Android App Code Signing
- A use case for digital certificates
- Google requires all apps to be signed by their developers before release.
  - A developer uses their private key to sign an app.
  - The public key is provided as part of the app in a (self-signed) certificate.
- Installation & update
  - At installation time, Android verifies if it’s signed.
  - When updating an app, Android verifies if it’s signed by the same private key.
- Sharing
  - Different apps from the same developer can be signed with the same private key.
  - Android allows those apps to share data without permission.
  - E.g., Facebook app, Facebook Messenger, & Instagram

Android Platform Key
- Another use case for digital certificates
- When compiling the Android OS, a vendor (Google, Samsung, etc.) includes their certificate (public key) in the platform.
- A vendor, e.g., Google, signs their apps with their private key.
  - When installed from Google Play, Android verifies that those apps are Google apps (called platform apps, e.g., Google Play Services app).
  - They can have more privilege than apps from regular devs.
- An OS update package is also signed by the same private key and verified before installation.

Authentication
- Use of cryptography to have two principals verify each others’ identities.
  - Direct authentication: the server uses a shared secret key to authenticate the client.
  - Indirect authentication: a trusted authentication server (third party) authenticates the client.
  - The authentication server knows keys of principals and generates temporary shared key (ticket) to an authenticated client. The ticket is used for messages in this session.
    - E.g., Verisign servers

Direct Authentication
- Authentication with a secret key
  - “Nonce” (used as a “challenge”) random num.
  - Bob calculates $K_{A,B}(R_B)$ and matches with reply. Alice is the only one who could have replied correctly.
“Optimized” Direct Authentication

• Authentication with a secret key with three messages

A

1. $A, R_A$

2. $R_B, K_{AB}(R_A)$

3. $K_{AB}(R_B)$

Bob

• Anything wrong with this?

Reflection Attack

Needham-Schroeder Authentication

• An authentication server provides secret keys.
  – Every client shares a secret key with the server to encrypt their channels.
• If a client A wants to communicate with another client B,
  – The server sends a key to the client A in two forms.
  – First, in a plain form, so that the client A can use it to encrypt its channel to the client B.
  – Second, in an encrypted form (with the client B’s secret key), so that the client B can know that the key is valid.
  – The client A sends this encrypted key to the client B as well.
• Basis for Kerberos

Needham-Schroeder Authentication

K_B

K_A

K_B

System A

K_A

System B

Authentication System

$<B, K_{AB}, (K_{AB}, A) > K_A$

$< (K_{AB}, A) > K_A$

$<Ν_B > K_A$

$< Ν_B - 1 \text{ req}, K_{AB} >$

$< Ν_B \text{ res} > K_{AB}$

Ticket

A demonstrated that it is the sender of the previous message

Chuck has stolen $K_B$ and intercepted message 2. He can masquerade as the authentication system.

Kerberos

• Follows Needham-Schroeder closely
• Time values used for nonces
  – To prevent replay attacks
  – To enforce a lifetime for each ticket
• Very popular
  – An Internet standard
  – Default in MS Windows

Nonce $Ν_A$ in Message 1

Because we need to relate message 2 to message 1

Authentication System

K_B

K_A

System A

$<B, K_{AB}, (K_{AB}, A) > K_A$

System C

$<B, K_{AB}, (K_{AB}, A) > K_A$

$<B, K_{AB}, (K_{AB}, A) > K_A$

$<B, K_{AB}, (K_{AB}, A) > K_A$
**Kerberos**

1. Request for TGS ticket
2. TGS ticket
3. Request for server ticket
4. Server ticket
5. Service request

Request encrypted with session key
Reply encrypted with session key

**Summary**

- **Security properties**
  - Confidentiality, authenticity, integrity, availability, non-repudiation, access control
- **Three types of functions**
  - Cryptographic hash, symmetric key crypto, asymmetric key crypto
- **Applications**
  - Password store, secure digest, MAC, & digital signature

**Acknowledgements**

- These slides contain material from “Principles of Computer System Design: An Introduction,” Chapter 11
- These slides contain material developed and copyrighted by Indrani Gupta (UIUC), Jennifer Rexford (Princeton) and Michael Freedman (Princeton).