CSE 486/586 Distributed Systems Security

Steve Ko
Computer Sciences and Engineering
University at Buffalo

Today

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

Security Properties

- Assume a system that processes requests from agents, and a request comes in from an agent. 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 defaults, so make sure that they 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 (or execute it with) 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:

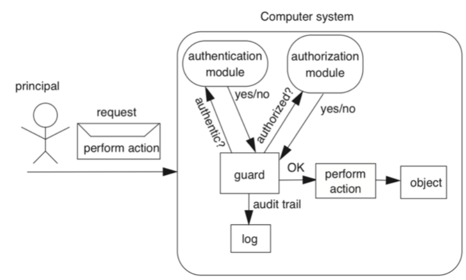
    *gets(**character array reference** string_buffer)*

    If the program allocates 30 bytes, and 250 bytes come in, then there's a buffer overrun problem.
- 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), so you can get it right.

Secure System Model



- A guard is commonly called a reference monitor.

CSE 486/586 Administrivia

- PA4 deadline: 5/10
- Survey & course evaluation
  - Survey: https://forms.gle/eq1wHN2G8S6GVz3e9
  - Course evaluation: https://www.smartevals.com/login.aspx?s=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

plaintext —**encryption**→ **ciphertext** —**decryption**→ **plaintext**

- 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 SHA-0 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

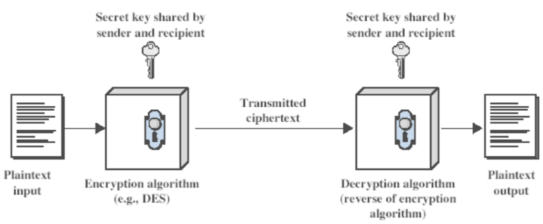
Cryptographic Hash Functions

- Takes message, *m*, of arbitrary length and produce 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 != 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 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

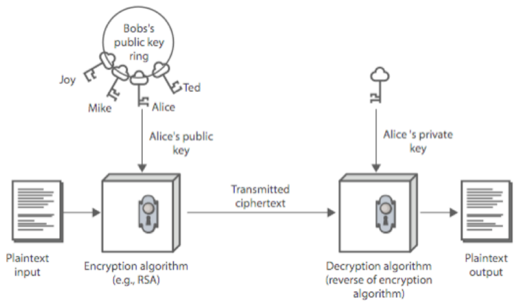
Symmetric Key Crypto



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 msgs or create signatures
  - If "one-way function" goes c ← F(m), then public-key encryption is a "trap-door" function:
    - » Easy to compute c ← F(m, pub)
    - » Hard to compute m ← F^{-1}(c) without knowing priv
    - » Easy to compute m ← F^{-1}(c,priv) by knowing priv

Public (Asymmetric) Key Crypto

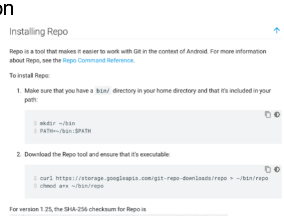


Application: Storing Passwords

- Password hashing
  - Password systems don'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)?
- Actually, a salt is often used: hash (input || salt)
  - 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


Installing Repo

Repo is a tool that makes it easier to work with Git in the context of Android. For more information about Repo, see the Repo Command Reference.

To install Repo:

1. Make sure that you have a `bin/` directory in your home directory and that it's included in your path:

```
$ mkdir ~/bin
$ PATH=~/bin:$PATH
```

2. Download the Repo tool and ensure that it's executable:

```
$ curl https://storage.googleapis.com/git-repo-downloads/repo > ~/bin/repo
$ chmod a+x ~/bin/repo
```

For version 1.25, the SHA-256 checksum for Repo is `d06f33115aea44e583c8669375b35aad397176a411de3461897444d247b6c220`.

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 = AES_K(SHA1 (M))
  - Sends <M, H>
- Receiver
  - Receives <M, H>
  - Computes a message digest: SHA1 (M)
  - Encrypts the message digest: H' = AES_K(SHA1 (M))
  - Checks the equality: H' == H

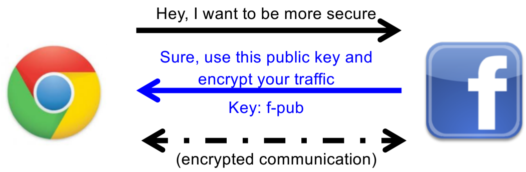
Application: Digital Signature

- 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 = RSA_K(SHA1(M))
  - Posts the message & the signature: <M, H>
- Verifier
  - Obtains <M, H>
  - Computes a message digest: H' = SHA1(M)
  - Decrypt the signature with the public key: $RSA_{K'}$(H)
  - Verifies the equality: $RSA_{K'}$(H) == H'

HTTPS

- A use case for digital signatures and public key encryption
- 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



- 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 (that you trust).
  - The (trusted) third party basically vouches that a public key belongs to an organization.
  - 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 Verisign's 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." (trust delegation)
  - But in order to trust Verisign's public key, some other trusted entity needs to verify the trustworthiness of Verisign's public key. (another trust delegation necessary)
  - You can establish a chain of trust that way. But delegation has to stop somewhere and you need to actually trust something.
  - This end of the chain is called the root of trust (something that you actually 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.

On My Mac…

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:
   Data:
       Version: 1 (0x0)
       Serial Number: 7829 (0x1e95)
       Signature Algorithm: md5WithRSAEncryption
       Issuer: C=ZA, ST=Western Cape, L=Cape Town, O=Thawte Consulting cc,
               OU=Certification Services Division,
               CN=Thawte Server CA/emailAddress=server-certs@thawte.com
       Validity
           Not Before: Jul  9 16:04:02 1998 GMT
           Not After : Jul  9 16:04:02 1999 GMT
       Subject: C=US, ST=Maryland, L=Pasadena, O=Brent Baccala,
                OU=FreeSoft, CN=www.freesoft.org/emailAddress=baccala@freesoft.org
       Subject Public Key Info:
           Public Key Algorithm: rsaEncryption
           RSA Public Key: (1024 bit)
               Modulus (1024 bit):
                   00:b4:31:98:0a:c4:bc:62:c1:88:aa:dc:b0:c8:bb:
                   33:35:19:d5:0c:64:b9:3d:41:b2:96:fc:f3:31:e1:
                   66:36:d0:8e:56:12:44:ba:75:eb:e8:1c:9c:5b:66:
                   70:33:52:14:c9:ec:4f:91:51:70:39:de:53:85:17:
                   16:94:6e:ee:f4:d5:6f:d5:ca:b3:47:5e:1b:0c:7b:
                   c5:cc:2b:6b:c1:90:c3:16:31:0d:bf:7a:c7:47:77:
                   8f:a0:21:c7:4c:d0:16:65:00:c1:0f:d7:b8:80:e3:
                   d2:75:6b:c1:ea:9e:5c:5c:ea:7d:c1:a1:10:bc:b8:
                   e8:35:1c:9e:27:52:7e:41:8f
               Exponent: 65537 (0x10001)
   Signature Algorithm: md5WithRSAEncryption
       93:5f:8f:5f:c5:af:bf:0a:ab:a5:6d:fb:24:5f:b6:59:5d:9d:
       92:2e:4a:1b:8b:ac:7d:99:17:5d:cd:19:f6:ad:ef:63:2f:92:
       ab:2f:4b:cf:0a:13:90:ee:2c:0e:43:03:be:f6:ea:8e:9c:67:
       d0:a2:40:03:f7:ef:6a:15:09:79:a9:46:ed:b7:16:1b:41:72:
       0d:19:aa:ad:dd:9a:df:ab:97:50:65:f5:5e:85:a6:ef:19:d1:
       5a:de:9d:ea:63:cd:cb:cc:6d:5d:01:85:b5:6d:c8:f3:d9:f7:
       8f:0e:fc:ba:1f:34:e9:96:6e:6c:cf:f2:ef:9b:bf:de:b5:22:
       68:9f
```


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.
  - E.g., the Iranian gov. has been suspected to compromise a certificate authority and issued rogue certificates that approve rogue websites that masquerade as 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

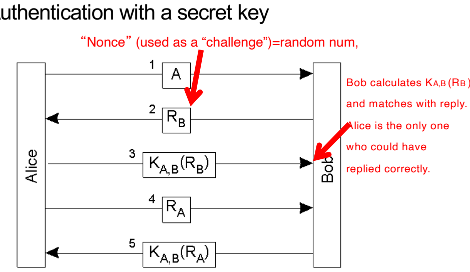
- Another use case for digital certificates
- When compiling the Android OS, a vendor (Google, Samsung, etc.) includes their certificate (public key) in the platform.
- A vendor, e.g., Google, signs their apps with their private key.
  - When installed from Google Play, Android verifies that those apps 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

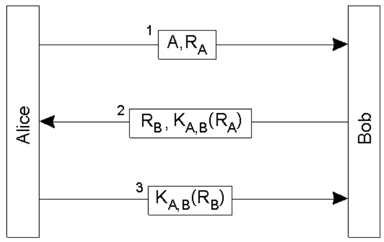
Direct Authentication

- Authentication with a secret key



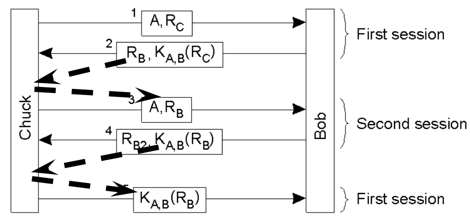
"Optimized" Direct Authentication

- Authentication with a secret key with three messages



- Anything wrong with this?

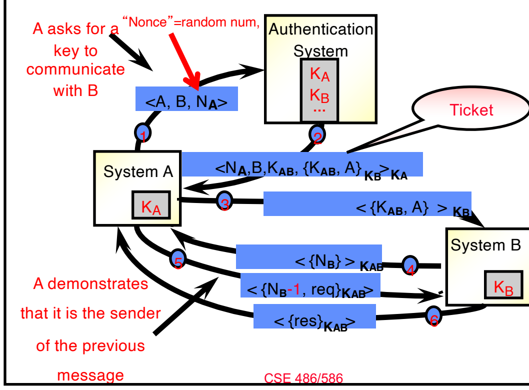
Reflection Attack



Needham-Schroeder Authentication

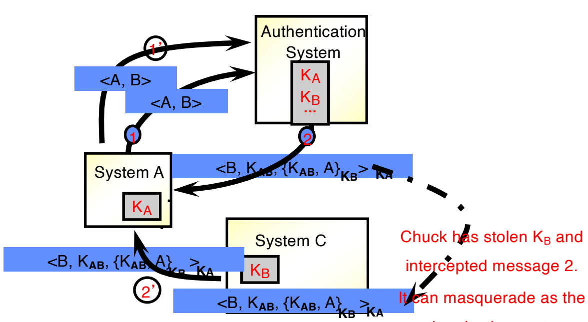
- An authentication server provides secret keys.
  - Every client shares a secret key with the server to encrypt their channels.
- If a client A wants to communicate with another client B,
  - The server sends a key to the client A in two forms.
  - First, in a plain form, so that the client A can use 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



Nonce N_A in Message 1

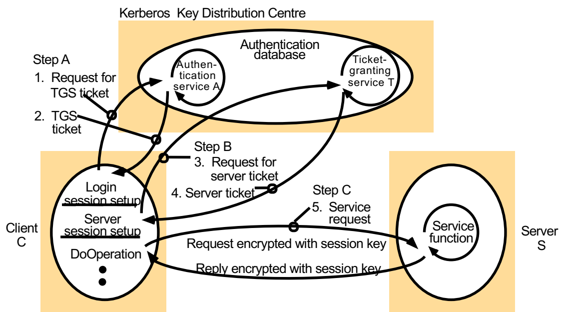
Because we need to relate message 2 to message 1



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

Kerberos



Summary

- Secure system design
  – Design principles, the safety net approach, TCB, etc.
- Three types of functions
  – Cryptographic hash, symmetric key crypto, asymmetric key crypto
- Applications
  – Password store, secure digest, MAC, digital signature, and digital certificates.

Acknowledgements

- These slides contain material from "Principles of Computer System Design: An Introduction," Chapter 11
  – https://ocw.mit.edu/resources/res-6-004-principles-of-computer-system-design-an-introduction-spring-2009/online-textbook/protection_open_5_0.pdf
- These slides contain material developed and copyrighted by Indranil Gupta (UIUC), Jennifer Rexford (Princeton) and Michael Freedman (Princeton).