When You Think About Computer Security . . .

• What floats into your mind at first moment?
  – A 3000 lbs lock
  – An ocean of bits
  – A lock, and a key
  – Prof. Li giving a lecture
  – A hacker typing password repeatedly
  – A piece of paper (with security policies written), and a CD (software package implementing the policies)
  – Your choice?

Some Points to Make

• Class review should use lecture notes as the guideline instead just blindly reading textbook line by line
• Today’s review slides are based on previous slides
  – Not necessarily the same, though
  – Hopefully provide you an even clearer logical flow
• A tip: write down page numbers for those slides that you have questions
  – Then go home and figure out
• We’ll move on slides quite quickly

What We’ve Covered

• Overview of computer security
• Access control matrix
• Security policies
• Basic cryptography
• Key management
• Cipher techniques
• Authentication
• Security mechanisms
• Access control mechanisms

A Taxonomy

Security Mechanics

Security Policy
Confidentiality
Integrity
Availability

Cryptography

Authentication

Access Control Matrix

Bell-Lapadula Model

Key Management

Cipher Techniques

<not covered>

<not covered>
Overview of Computer Security

- Three basic components: C.I.A.
  - Confidentiality, integrity, availability
- Four classes of threats
  - Disclosure, deception, disruption, usurpation
- Three goals of computer security
  - Prevention, detection, recovery

Security Life Cycle

- Threats
- Policy
- Specification
- Design
- Implementation
- Operation and Maintenance

Security Policy and Mechanism

- Policy: a statement of what is allowed and what is not
  - In order to provide confidentiality, integrity, and availability in the context of a specific system
- Mechanism: a method, tool, or procedure for enforcing a security policy
  - So that those threats cannot endanger a system’s security

Mechanisms: secure, precise, or broad

- Denote
  - $P$: the set of all possible states
  - $Q$: the set of secure states as specified by security policy
  - $R$: the set of states that a system can enter with the security mechanisms provided
- A security mechanism is
  - Secure: if $R \subseteq Q$
  - Precise: if $R \cap Q$
  - Broad: if $\exists$ state $r \in R$ but $r \not\in Q$

Access Control Matrix

- A theoretical tool for expressing security policy
  - Can express any expressible security policy
  - Simple and precise
  - Ideal for theoretical analysis of security problems

Access Control Matrix Model

- Specifies the rights every subject $s$ has over every object $o$ in a system
  - Every matrix element is $a(s, o)$.
- Described what’s allowed and what’s not
  - Recall that’s exactly what a security policy does
Access Control Matrix vs. Protection State Transition

- An access control matrix describes how a secure protection state should look like.
- But every system is dynamic.
  - Thus incurring state transition.
  - Which leads to a new access control matrix.
- If we want a dynamic system to be always secure.
  - We need to ensure the new access control matrix at each new state still describes a secure protection state.

Protection State Transition

- A system’s state can be represented as \((S, O, A)\).
- Initial State: \(X_0 = (S_0, O_0, A_0)\).
- With a set of operations \(\{1, 2, \ldots\}\), successive states are \(X_0, X_1, \ldots\), where
  \[
  X_i |_{\{2_{n+1} \}} X_{i+1}
  \]
- Note that \(A_i\) also changes to \(A_{i+1}\).

Command

- To represent a sequence of state transitions after a series of operations, we use
  \[X(p) \times Y\]
- This series of operations are called command.
  - Primitive commands.
  - Complex commands.
  - Conditional commands.
- Transition by command can be written as
  \[X_i(p_i, s_i, 1, \ldots, p_{n+1, 0} ) \times X_{i+1}\]

Primitive Command

- \((S,O,A) |\{1, 2, \ldots\}, (S', O', A')\) that alters access control matrix.
- A set of primitive commands defined by Harrison etc.:
  - Create subject \(x\).
  - Create object \(o\).
  - Enter \(r\) into \(a_{lx0}\).
  - Delete \(r\) from \(a_{lx}\).
  - Destroy subject \(x\).
  - Destroy object \(o\).

Complex Command

- A sequence of multiple primitive commands.
- Examples:
  
<table>
<thead>
<tr>
<th>Command</th>
<th>Create file ((p,f))</th>
<th>Create object (f)</th>
</tr>
</thead>
<tbody>
<tr>
<td>create_file ((p,f))</td>
<td>create object (f)</td>
<td></td>
</tr>
<tr>
<td>enter (o) into (a_{lpf})</td>
<td>enter (o) into (a_{lpf})</td>
<td></td>
</tr>
<tr>
<td>enter (r) into (a_{lpf})</td>
<td>end</td>
<td></td>
</tr>
</tbody>
</table>

Conditional Commands

- Some specific conditions must be met before a state can be changed.
  
  Command
  
  ```
  grant_read_file \((p, f, q)\)
  if \(\text{in } a[p, f]\) then
  enter \(R\) into \(a[q, f]\)
  end
  ```
  
  Command
  
  ```
  grant_read_file \_2 \((p, f, q)\)
  if \(\text{in } a[p, f]\) and \(c\) in \(a[p, f]\) then
  enter \(r\) into \(a[q, f]\)
  end
  ```
Security Policies

- Earlier we had an informal definition of security policy
  - A security policy states what’s allowed and what’s not
- And used access control matrix to theoretically consider how to express a security policy
- Here we look at them more thoroughly
  - A security policy can be formally defined and modeled
    - Security model
    - Can be expressed in languages
      - High-level, or low-level
    - Can be inspected at each different aspect
      - Such as Bell-LaPadula model for confidentiality

Example 1: Java classes and methods

- Pandey and Hashii defined a security policy for downloaded Java programs
- Format: `deny (s op r) when b`
  - `s -` `x` : `s` creates an instance of class `x`
  - `x.()` `x` : `x` invokes a method `x`
  - `b` : a boolean expression
    - Means `s` cannot “op” `r` when `b` is true
- Policy can be statement like the following:
  - `deny (s (Socket) when (Network.muxconn >= 100)`
  - `deny (s (file.read) when (file.getfilename() == "/etc/passwd")`

Security Policy vs. C.I.A.

- A security policy considers aspects of confidentiality, integrity, and availability
  - Identify states in which info leakage happens (confidentiality)
  - Describes the condition and manner that data can be altered (integrity)
  - Describes what services must be provided (availability)
- Two major types
  - Military security policy (primarily confidentiality)
  - Commercial security policy (primarily integrity)

Access Control in Security Policy

- Discretionary access control (DAC)
  - An individual user can set an access control mechanism to allow or deny access to an object
  - Also called identity-based access control (IBAC)
- Mandatory access control (MAC)
  - A system mechanism controls access to an object
    - Individual users cannot alter that access
  - Also called rule-based access control
- Originator controlled access control (ORCON)
  - The creator has the control (not covered so far)

High-level Policy Language

- Describes constraints placed on entities and actions in a system
- No ambiguity
  - Requires mathematical or programmatic languages
- Ignores implementation issues
  - Just specify

Example 2: DTEL

- DTEL; Domain-type enforcement language
- Every subject has a domain
- Every object has a type
- DTEL constraints the actions that a subject can perform on an object
Case Study using DTEL

• How can we express the mandatory access control of files in a system

• \texttt{d\_user}: domain for ordinary users
• \texttt{d\_admin}: domain for administrative users
• \texttt{t\_sysbin}: type for executable files
• \texttt{t\_readable}: type for readable files
• \texttt{t\_writable}: type for writable files

(DTEL statements:
\texttt{type t\_readable, t\_writable, t\_sysbin;}
\texttt{domain d\_admin =}
\texttt{(\texttt{/usr/bin/sh)},
\texttt{(crwx -> t\_readable, t\_writable, t\_sysbin);}\n\texttt{domain d\_user =}
\texttt{(\texttt{usr/bin/sh}),
\texttt{(rxd -> t\_sysbin),
\texttt{(crw -> t\_writable),
\texttt{(rd->t\_readable);}})}

Bell-LaPadula Model for Confidentiality

• Corresponds to military-style security classifications
  – Top secret, secret, confidential, unclassified
• A subject has a security clearance
  – One of the four above
• An object has a security classification (or security level)
  – Also one of the four above
• Goal: prevent read access to objects at a security classification higher than the subject’s clearance

Simple Security Condition

• \( S \) can read \( O \) iff \( l_o \leq l_s \) and \( S \) has discretionary read access to \( O \).

*-Property

• \( S \) can write \( O \) iff \( l_s \leq l_o \) and \( S \) has discretionary write access to \( O \).

Basic Security Theorem

• \( \square \): a system with a secure initial state \( \square_0 \)
• \( T \): a set of state transformations
• If every elements of \( T \) preserves the simple security condition (preliminary version) and the *-property (preliminary version), then every state \( \square_i \) (\( i \geq 0 \)) is secure.
Redefined “Security Level”

- By adding the concept of categories
  - Objects are placed in categories
- “Need to know” principle
  - A subject should not access info that is unnecessary for its task
  - So a subject may only access objects in certain categories
- Security level now also contains category info
  - Expressed as \((L, C)\)
  - E.g. William cleared into the level \((\text{SECRET}, \{\text{EUR}\})\)
  - E.g. A doc classified as \((\text{CONFIDENTIAL}, \{\text{US}\})\)

Simple Security Condition

- \(S\) can read \(O\) iff
  \[ S \text{ dom } O \quad \text{and} \quad S \text{ has discretionary read access to } O. \]

*-Property

- \(S\) can write \(O\) iff
  \[ O \text{ dom } S \quad \text{and} \quad S \text{ has discretionary write access to } O. \]

Basic Security Theorem

- \([\ ]\): a system with a secure initial state \([s_0]\)
- \(T\): a set of state transformations
- If every elements of \(T\) preserves the simple security condition and the *-property, then every state \([s_i]\) \((i \geq 0)\) is secure.

Basic Cryptography

- Goal: keep enciphered info secret
  - A deep mathematical subject
- Usage: a cornerstone for secure communication
- Assumptions: attackers know the algorithm but not the key(s)
- Types: classical cryptosystems and public key cryptosystems

Four Main Topics Covered

- Classical cryptography
- Public key cryptography
- Cryptographic checksum function
- Digital signature
Definitions

- **Cryptography**: the art and science of concealing information
- **Cryptoanalysis**: code breaking
- **Cryptosystem**: basic component of cryptography
  - \( E, D, M, K, C \)
  - \( M \): plaintexts
  - \( K \): keys
  - \( C \): ciphertexts
  - \( E \): enciphering functions \( M \times K \rightarrow C \)
  - \( D \): deciphering functions \( C \times K \rightarrow M \)

Classical Cryptosystems

- Same key for encipherment and decipherment
  - Also called single-key cryptosystem
  - Or symmetric cryptosystem
- For all \( E_k \) there is \( D_k \) such that \( D_k = E_k^{-1} \)
- Examples:
  - Transposition cipher
  - Substitution cipher
  - Vigenere cipher, One-time pad, etc.
  - DES: the combination of both

Transposition Cipher

- Characters in plaintext are rearranged
  - Letters unchanged
- **Rail fence** cipher, as an example
  - "UNIV OF OREGON" becomes "UI O OEOV F RGN" or "UVFRA NINOOG"
  - "UI O OEOV F RGN" or "UVFRA NINOOG"

Substitution Cipher

- Characters are changed
  - Caesar cipher for example, where letters are simply shifted
- Examples:
  - Vigenere cipher
  - One-time pad

Vigenere Cipher

- Use a longer key to obscure the statistics
- The length of a key is called the **period** of the cipher
- A **tableau** is used to implement cipher
  - Table lookup for encipherment

  **Key**  B ENCHBENC HBENC HBENCH
  **Plaintext**  A LIMERICK PACKS LAUGHS
  **Ciphertext**  B PVOLSMPM WBGXU SBYTJZ

One-Time Pad

- A variant of the Vigenere cipher
- But key string is randomly chosen and **at least as long the message**!
  - No repetition
- Impossible to break! **Perfect secrecy :)**
  - Impossible to deploy either. :(
DES: Data Encryption Standard

- A classical cryptosystem
- Bit-level
- Uses both transposition and substitution
  - Also referred as product cipher

- Encipherment unit: 64-bit blocks
  - Input, output and keys are all in 64b blocks

Three Common Modes of DES

- CBC: Cipher Block Chaining
- EDE: Encrypt-Decrypt-Encrypt
- Triple DES: DES-DES-DES

CBC

<table>
<thead>
<tr>
<th>Initial</th>
<th>m_0</th>
<th>m_1</th>
<th>c_0</th>
<th>c_1</th>
</tr>
</thead>
<tbody>
<tr>
<td>DES</td>
<td></td>
<td></td>
<td>DES</td>
<td></td>
</tr>
<tr>
<td>DES</td>
<td>c_0</td>
<td></td>
<td></td>
<td>c_1</td>
</tr>
<tr>
<td>Encipherment</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

EDE

- Two 64-bit keys: k and k'

\[ c = DES_k(DES_{k'}^{-1}(DES_{k'}(m))) \]

Triple DES

- Three 64-bit keys: k, k', and k''

\[ c = DES_k(DES_{k'}(DES_{k''}(m))) \]

AES: Advanced Encryption Standard

- DES is no longer as secure as designed in its early days

- 2001. NIST selects Rijndael as AES.
### Public Key Cryptography
- Use two different keys for encryption and decryption
- An entity has two keys: a **public key** and a **private key**
  - Hard to derive the private key from the public key
- Examples:
  - Diffie-Hellman
  - RSA
  - . . . . .

### Properties of Public Key
- Assuming $x$ has a public key $e$ and a private key $d$
- Message encrypted with $e$ can only be decrypted by $x$ using $d$
  - Useful to send an encrypted message to $x$
- If a message can be decrypted with $e$, then it must be encrypted by $x$ using $d$
  - Useful to verify whether or not a message is from $x$

### Combine Confidentiality and Authentication
- For confidentiality, the message has to be encrypted with B’s public key
  - So that B’s private key has to be used to decrypt
- For origin authentication, the message has to be encrypted with A’s private key
  - So that A’s public key has to be used to decrypt
  - Everybody knows A’s public key
- Question: can we switch the two above?

### Cryptographic Checksums
- Motivating question: How can Bob verify messages received from Alice is not changed?
- Answer: digital signature
  - Which relies on cryptographic checksum function
  - Digital signature will be covered later
- Cryptographic checksum function also has many other usages
  - Such as S/Key protocol (covered in Authentication later)

### Cryptographic Checksum Function
- Also called **strong hash function**
  - Or strong one-way function
- $h: A \rightarrow B$
  - For any $x \in A$, $h(x)$ is easy to compute
  - For any $y \in B$, computationally infeasible to find $x \in A$ such that $h(x) = y$
  - No collision pairs.

### Prevention of Collision Pairs
- **Statement A:**
  - Computationally infeasible to find $x, x' \in A$ such that $x \neq x'$ but $h(x) = h(x')$
- **Statement B:**
  - Given any $x \in A$, computationally infeasible to find another $x' \in A$ such that $x \neq x'$ but $h(x) = h(x')$
- **Statement B** is much harder than **Statement A**.
Keyed or Keyless Cryptographic Checksum

- **A keyed cryptographic checksum** requires a cryptographic key as part of hashing computation
  - E.g. DES-MAC (DES in CBC mode)
  - Use last block output as the hash result
  - DES needs a key
- **A keyless cryptographic checksum** does not
  - MD2, MD4, MD5
  - SHA-1 (Secure Hash Algorithm)
  - Snefru
  - HAVAL

Digital Signature

- **A digital signature** is a construct that authenticates both the origin and contents of a message in a manner that is provable to a disinterested third party.
- Provides a service of nonrepudiation

Classical Signature

Let Cathy be a trusted third party
Alice shared a secret key $k_{AC}$ with Cathy
Bob shared a secret key $k_{BC}$ with Cathy
1. AliceÆBob: $\{m\}_{k_{AC}}$
2. BobÆCathy: $\{m\}_{k_{AC}}$
   - Cathy deciphers with $k_{AC}$ and re-enciphers with $k_{BC}$
3. CathyÆBob: $\{m\}_{k_{BC}}$
   - Bob then gets $m$

Classical Signature Verification

- Verification question: is $m$ created by Alice?
- Verification method:
  - Judge takes the disputed messages $\{m\}_{k_{AC}}$ & $\{m\}_{k_{BC}}$
  - Ask Cathy to decrypt $\{m\}_{k_{AC}}$ using $k_{AC}$ and $\{m\}_{k_{BC}}$ using $k_{BC}$
  - And compare $\{m\}_{k_{AC}} = \{m\}_{k_{BC}}$?

Public Key Signature

- Instead of using $(m)_{d_{ABC}}$, Alice actually signs the message as
  $$(h(m))_{d_{ABC}}$$
  where $h$ is a cryptographic hash function
- And sends Bob $m$, $h(m)$, $d_{ABC}$
- Q: how does Bob verifies the signature?

Key Management

- Key distribution
- Key generation
- Key maintenance
- Key revocation (not covered)
Key Exchange

- Goal: enable Alice and Bob to communicate secretly using a shared key
- Requirements
  - The shared key cannot be transmitted in the clear
  - Alice and Bob may decide to trust a third party (Cathy)
  - Cryptosystems and protocols are publicly known

Classical Cryptographic Key Exchange

- A simple scheme
- Needham-Schroeder protocol
- Revised Needham-Schroeder protocol by Denning and Sacco
- Otway-Rees protocol
- Kerberos

A Simple Scheme

(1) Alice\[\text{Cathy : \{request for session key to Bob\}\{k\}_A\text{C}
(2) Cathy\[\text{Alice : \{\{k\}_A\text{C} \{\{k\}_B\text{C}\}
(3) Alice\[\text{Bob : \{\{k\}_B\text{C}\}
(4) Both Alice and Bob have the session key \{k\}

Subject to replay attack!

Needham-Schroeder Protocol

(1) Alice\[\text{Cathy : \{AliceBob\}\{\text{rand}\}_A
(2) Cathy\[\text{Alice : \{AliceBob\}\{\text{rand}\}_A\{\{k\}_B\text{C}\}
(3) Alice\[\text{Bob : \{\{k\}_B\text{C}\}
(4) Bob\[\text{Alice : \{\text{rand}\}_A\{\{k\}_B\text{C}\}
(5) Alice\[\text{Bob : \{\text{rand}\}_A\{\{k\}_B\text{C}\}

Eve may be able to predict the session key!

Revised Needham-Schroeder Protocol

- Use timestamp for replay prevention

(1) Alice\[\text{Cathy : \{AliceBob\}\{\text{rand}\}_A
(2) Cathy\[\text{Alice : \{AliceBob\}\{\text{rand}\}_A\{\{k\}_B\text{C}\}
(3) Alice\[\text{Bob : \{\{k\}_B\text{C}\}
(4) Bob\[\text{Alice : \{\text{rand}\}_A\{\{k\}_B\text{C}\}
(5) Alice\[\text{Bob : \{\text{rand}\}_A\{\{k\}_B\text{C}\}

Requires synchronized clocks!

Otway-Rees Protocol

(1) Alice\[\text{Bob : num\{AliceBob,\
(2) Bob\[\text{Cathy : num\{AliceBob,\
(3) Cathy\[\text{Bob : num,\{\text{rand}\}_A\{\{k\}_B\text{C}\}
(4) Bob\[\text{Alice : num,\{\text{rand}\}_A\{\{k\}_B\text{C}\}

Finally work!
1. Alice [Æ] Cerberus: Alice|Barnum
2. Cerberus [Æ] Alice: \{k_{A,B}\} k_{A,C} || T_{A,B}  
   (Ticket) T_{A,B} = Bll(AllA’s address|valid timell k_{A,B}) k_B

3. Alice [Æ] Barnum: Guttenberg || T_{A,G}  
   T_{A,G} = Bll(AllA’s address|valid timell k_{A,B}) k_B 
   [Authenticator] A_{A,G} = (All generation timell k_t) k_{A,B}
4. Barnum [Æ] Alice: Alliance(\{k_{A,G}\} k_{A,B} || T_{A,G})

5. Alice [Æ] Guttenberg: A_{A,G} || T_{A,G}  
   T_{A,G} = Gll(AllA’s address|valid timell k_{A,G}) k_G  
   A_{A,G} = (All generation timell k_t) k_{A,G}

6. Guttenberg [Æ] Alice: \{t+1\}k_{A,G}

Fix the Flaw: Sign \( k_s \)

Alice [Æ] Bob : Alice, \{\{k_s\}d_{Alice} \} e_{Bob}

- Question: What if Alice does not have Bob’s public key?

Public Key Cryptography Key Exchange

- Very easy
  
  Alice [Æ] Bob : \{k_s\} e_{Bob}  
  - Alice and Bob then can securely communicate using a symmetric cryptosystem

- Flaw: Bob does not know who sent the message  
  - Could be Eve forging such a message

Key Generation

- Critical to generate keys that are hard to figure out by attackers
- Key generation issue is regarded as a \textit{randomness} issue  
  - Given a set of \( K \) potential keys, the minimum probability of successfully guessing the key is achieved when the key is \textit{randomly} selected
Random Numbers

- A sequence of (cryptographically) random numbers is a sequence of numbers $n_1, n_2, \ldots$ such that for any $k$, an observer cannot predict $n_k$ even if $n_1, n_2, \ldots, n_{k-1}$ are known.
- A random number generator requires a physical source of randomness:
  - Disk latency at different time
  - Background radiation
  - Oregon’s rain in April
  - Having a CIS 410 quiz

Pseudorandom Numbers

- Physical sources of randomness are often not available.
- A sequence of (cryptographically) pseudorandom numbers is a sequence of numbers intended to simulate a sequence of cryptographically random numbers:
  - But generated by an algorithm
  - Such algorithm is difficult

Pseudorandom Number Generation

- A linear congruential generator:
  $$n_k = (an_{k-1} + b) \mod n$$ (a, b, n are relatively prime)
- A polynomial congruential generator:
  $$n_k = (a_0 n_{k-1} + \ldots + a_p n_{k-p}) \mod n$$
- A mixing function $f$
  $$y = f(x_1, x_2, \ldots)$$
  where every bit of $y$ is a nonlinear function of all the bits of $x_1, x_2, \ldots$ (e.g., DES, MD4, MD5, SHA-1)
  - Initial input to $f$ must be unpredictable, e.g., (date; ps augpaj | md5)

Key Maintenance

- In classical cryptosystems, every key is shared between two (or more) entities:
  - Then the key should be stored at every sharing entity during the key’s lifetime
  - Simple
  - In public key cryptosystems, every entity has a public key and private key:
    - Which are bound to the entity
  - We focus on key maintenance for public key cryptography

Identification and Public Key

- Every node has an ID
- Every node has a public key
- The association between the ID and the key is critical
- A central question: is this the public key for node X?
  - X is the ID

A Preliminary Solution

- A node encrypts (signs) its public key with its private key $\{e\} d$ recipient
- The recipient can only decrypt using the public key in question
- Thus confirm that $e$ is the public key of the signing guy
- But who is the signing guy?
- It won’t help by adding the ID, either $\{e, Alice\} d$ recipient
  since the $e$ and $d$ here can actually belong to Eve!
Certificate-Based Solution

• A certificate is a token that binds an identity to a cryptographic key

\[ C_{\text{Alice}} = \{e_{\text{Alice}} \parallel \text{Alice} \parallel T \} d_{\text{Cathy}} \]

• A certificate authority (CA) issues certificates

Certificate Verification

• Suppose Bob knows Cathy’s public key \( e_{\text{Cathy}} \)

• When Bob obtains \( C_{\text{Alice}} \)
  – Deciphers \( C_{\text{Alice}} \) using \( e_{\text{Cathy}} \)
  – Then knows that Cathy is vouching that \( e_{\text{Alice}} \) is Alice’s public key, issued at time \( T \)
  – If Bob trusts what Cathy believes
  – Then Bob knows \( e_{\text{Alice}} \) is Alice’s public key

But, Bob Has to Know \( e_{\text{Cathy}} \)

• Two solutions
  – Merkle’s Tree Authentication Scheme
    • Eliminates Cathy’s signature
    – Chain of certificates
    • There is another certificate for \( e_{\text{Cathy}} \)

Merkle’s Tree Authentication Scheme

• All \(<\text{id}, \text{public key}>\) pairs are stored in a file
• A cryptographic hash function creates a digest of the file
  – The digest is known to the public
• If any pair is changed, it will be detected
  – Since the digest will be different

Public Key Verification

• How can Bob verify whether or not Alice’s public key is 72384927894027.
• Bob will re-compute the digest, and compare that with the publicly known value of the digest
  – If Alice’s public key is not 72384927894027, a discrepancy will be detected
Authentication Path

- Bob knows $Y_3$
- Bob needs to know $h(4,4)$ and $h(1,2)$
- $Y_3, h(4,4)$ and $h(1,2)$ is the authentication path for Alice’s public key
  - They can put together and used for certifying Alice’s public key
- This is a certificate without signature!

Certificate Signature Chains

- X.509
- PGP
- Tree-like CA hierarchy employed
  - Every node has a local CA
  - A local CA has its CA, the parent
  - The parent CA has its parent
  - And there is a root CA
  - Together, a tree of CAs!

Cipher Techniques

- Cipher techniques must be used wisely
  - Very sensitive to the environment
- A mathematically strong cryptosystem is vulnerable when used incorrectly
  - Examples include: precomputing the possible messages, misordered blocks, and statistical regularities.
- So we introduced block cipher and stream cipher, and try to strengthen both!

Block Cipher

- $E$: an encipherment algorithm
- $E_k(b)$: encipherment of msg $b$ with key $k$
- Message $m = b_1b_2...$
  - where each $b_i$ is of fixed length
- Block cipher: $E_k(m) = E_k(b_1) E_k(b_2) ...$
- Q: is DES a block cipher?

Stream Cipher

- $E$: an encipherment algorithm
- $E_k(b)$: encipherment of msg $b$ with key $k$
- Message $m = b_1b_2...$
  - where each $b_i$ is of fixed length
- Stream cipher: $E_k(m) = E_k(b_1) E_k(b_2) ...$
- Q: is Vigenere a stream cipher?
  - Yes, and also a periodic stream cipher

Types of Stream Ciphers

- Two types, depending on how keys are generated:
  - Synchronous stream cipher
  - Self-synchronous stream cipher
Synchronous Stream Ciphers

- Generates bits of a key from a particular source
  - Not from the message itself
  - Hopefully the newly generated key is random and long
- Several techniques
  - LFSR (Linear feedback shift register)
  - NLFSR (Nonlinear feedback shift register)
  - Output feedback mode
  - Counter method

LFSR (linear feedback shift register)

- An n-bit register \( r = r_{n-1} \ldots r_0 \) (a variable)
- An n-bit tap sequence \( t = t_{n-1} \ldots t_0 \) (a constant)
- Use \( r_0 \) as current key bit
- Right shift \( r \), and \( r_{n-1} = (r_{n-1} \cdot t_{n-1}) \oplus \ldots \oplus (r_0 \cdot t_0) \)

Output Feedback Mode

- \( m \): the message to encrypt
- \( E \): encipherment function
- \( k \): cryptography key to generate
- \( r \): a register
  - \( r = E_k(r) \)
  - \( k_i = r_0 \) (\( r \)'s rightmost bit)
  - \( c_i = m_i \oplus k_i \)

NLFSR (nonlinear feedback shift register)

- New bit is a function of current register bits
  - No tap sequence used

<table>
<thead>
<tr>
<th>current reg</th>
<th>key</th>
<th>new ( r_{n-1} ) bit</th>
<th>new reg</th>
</tr>
</thead>
<tbody>
<tr>
<td>0010</td>
<td>0</td>
<td>( f(0,0,1,0) = 0 )</td>
<td>0001</td>
</tr>
<tr>
<td>0001</td>
<td>1</td>
<td>( f(0,0,0,1) = 0 )</td>
<td>0000</td>
</tr>
<tr>
<td>0000</td>
<td></td>
<td>( f = r_0 \oplus (r_2 \text{ and } r_0) )</td>
<td></td>
</tr>
</tbody>
</table>

Counter Method

- \( m \): the message to encrypt
- \( E \): encipherment function
- \( k \): cryptography key to generate
- \( i_0 \): initial value of a counter

- \( k_i = (i+i_0)'s \) rightmost bit \((\text{for } i=0, 1, 2, \ldots)\)
- \( c_i = m_i \oplus k_i \)

Self-Synchronous Stream Ciphers

- Generate a key from the message itself
  - Could be from plaintext, could be from ciphertext
  - Also called autokey cipher

<table>
<thead>
<tr>
<th>Key</th>
<th>Plaintext</th>
<th>Ciphertext</th>
</tr>
</thead>
<tbody>
<tr>
<td>XTHEBOYHASTHEBA</td>
<td>THEBOYHASTHEBA</td>
<td>THEBOYHASHECAT</td>
</tr>
<tr>
<td>XKBQOVUGAGNTT</td>
<td>QBQOVUGAGNTT</td>
<td>QBQOVUGAGNTT</td>
</tr>
</tbody>
</table>

The key stream can have a period of \( 2^n - 1 \) (maximal value)
If using plaintext, key selection is an issue
– Key will display same statistical regularities as it’s derived from plaintext
If using ciphertext, weak
– A character in ciphertext = \( f (X, \text{a previous character in ciphertext}) \)

**Cipher Feedback Mode**

\( m \): the message to encrypt
\( E \): encryption function
\( k \): cryptography key to generate
\( r \): a register

\[ x = E_k (r) \]
\[ c_i = m_i \oplus x_0(\text{x’s rightmost bit}) \]
\[ r = x_{n-1} r_{n-2} \ldots r_1 \]

**Block Cipher**

- Multiple bits each time
  – Faster than stream cipher in software implementations
- But an identical plaintext block will produce an identical ciphertext block
  – If using the same key

**Strengthening Block Cipher**

1. Insert extra bits into a block, often related to block position
   – Sequence number of a block
   – Bits from preceding ciphertext block
2. **Cipher block chaining (CBC)**
   - \( c_0 = E_k (m_0 \oplus I) \)
   - \( c_i = E_k (m_i \oplus c_{i-1}) \) for \( i > 0 \)

**Authentication**

- Authentication is the binding of an identity to a subject, which is acting on behalf of an entity
  – Or, the binding of an identity to an entity
- **How?**
  - What the entity knows (e.g., passwords)
  - What the entity has (e.g., a badge)
  - What the entity is (e.g., fingerprints)
  - Where the entity is (e.g., in front of a particular terminal)
  - . . . .
Authentication Process

- Obtain authentication info from an entity
- Analyze the info
- Determine whether or not the info is associated with the entity

For the purpose of analysis, the entity's info must be stored and managed

An authentication system

Authentication System

- $A$: the set of authentication info with which entities prove their identities
- $C$: the set of complementary info that the system stores and uses to validate the authentication info
- $F$: the set of complementation functions that for $f \in F, f: A \rightarrow C$
- $L$: the set of authentication functions that for $l \in L, l: A \times C \rightarrow \{\text{true, false}\}$

Authentication Systems We Learned

- Password
- Challenge-Response
  - One-time password
  - S/Key
  - Hardware-supported challenge-response
- Biometrics
- Location

Passwords

- A password is information associated with an entity that confirms the entity's identity
  - Simplest example: some sequence of characters
  - e.g., login, su, etc. in Unix
- $C$ may not be the same as $A$
  - Mostly because $C$ must be protected
  - e.g., /etc/passwd (or shadow password files) in Unix
- $F$
  - $f \in F$ is based upon DES in Unix
- $S$
  - e.g., passwd command in Unix

Challenge-Response

- Fundamental flaw of passwords: reusability
  - Can be replayed if known before
  - What if every time one uses different authentication information
- In a challenge-response authentication system
  - User $U$ and System $S$ share a secret function $f$
  - $S$ sends a random message $m$ (challenge)
  - $U$ replies with $f(m)$ (response)
  - $S$ validates $r$ by computing it separately

One-Time Password

- One-time password: a password that is invalidated as soon as it is used
- Also a challenge-response mechanism
  - Challenge: the number of authentication attempt
  - Response: the one-time password
S/Key

- $h$: a one-way hash function
- $k$: an initial seed chosen by the user

keys: $h(k) = k_1$, $h(k_1) = k_2$, \ldots, $h(k_{n-1}) = k_n$

passwd: $p_1 = k_n$, $p_2 = k_{n-1}$, \ldots, $p_{n-1} = k_2$, $p_n = k_1$

If Eve intercepts $p_i$, we know $p_i = h(p_{i+1})$, and $h$ is a one-way hash function, so $p_{i+1}$ cannot be derived from $p_i$.

S/Key Authentication Protocol

- User Matt supplies his name to the server
- The server replies with the number $i$ stored in the skeykeys file
- Matt supplies password $p_i$
- Server computes $h(p_i)$ and compares it with the stored password $p_{i-1}$.
  - If match, Authentication succeeds
  - $i \not= i + 1$, $p_{i+1} \not= p_i$

Note: errors on page 326

Hardware-Supported Challenge-Response Procedures

- **Token** device
  - System sends a challenge
  - User enters it into the device (PIN maybe needed)
  - The device returns a response, by hashing (or enciphering) the challenge
  - The user sends the response over

(cont’d)

- Temporally based device
  - Every 60 seconds, a different number displayed
  - The system knows what number to be displayed for a user
  - When the user logs in, he enters the number currently shown
  - Followed by a fixed password
  - e.g., RSA SecureID card

Biometrics

- As old as humanity
- Fingerprints
- Voices
- Eyes
- Faces
- Keystrokes
- Combinations

Location

- Anna is logging from Russia
  - But we know she is now working at California
- Dennis and MacDoran’s scheme: use Global Positioning System (GPS)
  - An entity obtains a location signature using GPS
  - Transmits it
  - The System uses a location signature sensor (LSS) to obtain a similar location signature
  - Compare the two signatures to authenticate
Design and Implementation of Security Mechanisms

- Design principles
- Access control mechanisms

Eight Principles for Security Mechanisms

- Principle of least privilege
- Principle of fail-safe defaults
- Principle of economy of mechanism
- Principle of complete mediation
- Principle of open design
- Principle of separation of privilege
- Principle of least common mechanism
- Principle of psychological acceptability

Access Control Mechanisms

- Earlier in the class we discussed access control matrix from theoretical point of view
- An access control matrix is difficult to literally represent in practice
  - Many rows and columns
  - Many blank or repeated entries
  - Many changes with additions or deletions

Optimizations for Simplicity and Convenience

- Access control list
- Capability
- Lock and key

Access Control List

$S$: set of subjects
$R$: set of rights

- An access control list (ACL) $l$ is a set of pairs $l = \{(s, r): s \in S, r \in R\}$
  - Also called classical ACL
- Let $acl(o)$ be a function that determines the ACL $l$ associated with a particular object $o$, then
  $$acl(o) = \{(s, r_i): n \geq 1\}$$
  where subject $s_i$ may access $o$ using any right in $r_i$.

Abbreviated ACL for UNIX File Access Control

- Classical ACL is not convenient sometimes
  - So use abbreviated ACL
- Case study: UNIX abbreviated ACL
  - Three classes of users: owner, group, other
  - File permissions are represented as three triplets
    - e.g. rwxr-xr--
Full ACL for AIX’s Extended Access Control

- Abbreviated ACL causes loss of granularity
  - How to implement “everybody but foo can access this file” in Unix?
  - Augment it with full ACL
- Case study: full ACL by AIX (a version of UNIX)
  - The abbreviated ACL provides base permissions
  - Then augment with extended permissions
    - Which can specify the rights of each individual user or group
  - Full ACL = base + extended permissions

ACL Creation and Maintenance

- Which subjects can modify an object’s ACL?
- Do the ACLs apply to a privileged user?
- Does the ACL support groups and wildcards?
- Conflicts
- ACLs and default permissions

Capability Definition

\( O \): set of objects
\( R \): set of rights
- A capability list (C-List) \( c \) is a set of pairs \( c = \{ (o, r): o \in O, r \in R \} \)
- Let \( \text{cap}(s) \) be a function that determines the C-List \( c \) associated with a subject \( s \), then
  \[ \text{cap}(s) = \{ (o_i, r_i): \text{forall } 1 \leq i \leq n \} \]
  where subject \( s \) may access \( o_i \) using any right in \( r_i \)

Comparison with ACL

- Two underlying questions for access control:
  - Given a subject, what object can it access, and how?
  - Given an object, what subject can access it, and how?
- C-List: simple for first, complicated for second
- ACL: complicated for first, simple for second
- In practice, the second question asked more often
  - Thus ACL-based system is more popular
  - Could change in the unforeseeable future?

Locks and Keys

- Combines the features of ACL and capability
  - Lock \( \Box \): object
  - Key \( \Box \): subject (and the desired access manner)
- Compare the key of a subject and the lock of an object when the subject access the object

L&K 1: Gifford Implementation

- Object \( o \) is enciphered with a cryptographic key
- Subject \( s \) has a deciphering key
- The subject needs to decipher the object in order to access the object
(cont’d)

- **Or-access**: $aE = (E_1(o), \ldots, E_n(o))$
  - Enciphering $n$ copies of an object using $n$ different keys
  - Thus allowing $n$ subjects to access the data

- **And-access**: $aE = E_1(\ldots(E_n(o)\ldots))$
  - Iterating the cipher using $n$ different keys, one per subject
  - Thus denying all accesses except on the request of all $n$ subjects

L&K 2: Type Checking

- A specific type of access must be on a specific type of object
  - A key matches a lock when the type matches
  - e.g., in UNIX, a process cannot write to a directory, cannot run non-executables, etc.

- *DTEL* is one example for type checking
  - See Lecture 2.

L&K 3: Sharing Secrets

- Q: how to allow any 3 of 10 given people to gain access to an object?

- A *(t, n)*-threshold scheme is a cryptographic scheme in which a datum is divided into $n$ parts, any $t$ of which are sufficient to determine the original datum
  - The $n$ parts are called shadows

That’s All For Today’s Review!
But Wait a Second . . .

Now, Close Your Eyes Again . . .

- What jumps into your mind when you think about computer security?
Good luck in midterm!