ORCON

• Problem: organization creating document wants to control its dissemination
  – Example: Secretary of Agriculture writes a memo for distribution to her immediate subordinates, and she must give permission for it to be disseminated further. This is “originator controlled” (here, the “originator” is a person).
Requirements

- Subject $s \in S$ marks object $o \in O$ as ORCON on behalf of organization $X$. $X$ allows $o$ to be disclosed to subjects acting on behalf of organization $Y$ with the following restrictions:
  1. $o$ cannot be released to subjects acting on behalf of other organizations without $X$’s permission; and
  2. Any copies of $o$ must have the same restrictions placed on it.
DAC Fails

- Owner can set any desired permissions
  - This makes 2 unenforceable
MAC Fails

• First problem: category explosion
  – Category $C$ contains $o$, $X$, $Y$, and nothing else. If a subject $y \in Y$ wants to read $o$, $x \in X$ makes a copy $o'$. Note $o'$ has category $C$. If $y$ wants to give $z \in Z$ a copy, $z$ must be in $Y$—by definition, it’s not. If $x$ wants to let $w \in W$ see the document, need a new category $C'$ containing $o$, $X$, $W$.

• Second problem: abstraction
  – MAC classification, categories centrally controlled, and access controlled by a centralized policy
  – ORCON controlled locally
Combine Them

- The owner of an object cannot change the access controls of the object.
- When an object is copied, the access control restrictions of that source are copied and bound to the target of the copy.
  - These are MAC (owner can’t control them)
- The creator (originator) can alter the access control restrictions on a per-subject and per-object basis.
  - This is DAC (owner can control it)
RBAC

• Access depends on function, not identity
  – Example:
    • Allison, bookkeeper for Math Dept, has access to financial records.
    • She leaves.
    • Betty hired as the new bookkeeper, so she now has access to those records
  – The role of “bookkeeper” dictates access, not the identity of the individual.
Definitions

- **Role** $r$: collection of job functions
  - $\text{trans}(r)$: set of authorized transactions for $r$
- **Active role of subject** $s$: role $s$ is currently in
  - $\text{actr}(s)$
- **Authorized roles of a subject** $s$: set of roles $s$ is authorized to assume
  - $\text{authr}(s)$
- $\text{canexec}(s, t)$ iff subject $s$ can execute transaction $t$ at current time
Axioms

• Let $S$ be the set of subjects and $T$ the set of transactions.

• **Rule of role assignment:**
  \[(\forall s \in S)(\forall t \in T) \ [\text{canexec}(s, t) \rightarrow \text{actr}(s) \neq \emptyset].\]
  – If $s$ can execute a transaction, it has a role
  – This ties transactions to roles

• **Rule of role authorization:**
  \[(\forall s \in S) \ [\text{actr}(s) \subseteq \text{authr}(s)].\]
  – Subject must be authorized to assume an active role
    (otherwise, any subject could assume any role)
Axiom

• Rule of transaction authorization:

\[(\forall s \in S)(\forall t \in T)\]
\[\left[\text{canexec}(s, t) \rightarrow t \in \text{trans}(\text{actr}(s))\right].\]

– If a subject \(s\) can execute a transaction, then the transaction is an authorized one for the role \(s\) has assumed
Containment of Roles

- Trainer can do all transactions that trainee can do (and then some). This means role $r$ contains role $r'$ ($r > r'$). So:

\[
(\forall s \in S)[ r' \in authr(s) \land r > r' \rightarrow r \in authr(s) ]
\]
Separation of Duty

• Let $r$ be a role, and let $s$ be a subject such that $r \in auth(s)$. Then the predicate $meauth(r)$ (for mutually exclusive authorizations) is the set of roles that $s$ cannot assume because of the separation of duty requirement.

• Separation of duty:

$$(\forall r_1, r_2 \in R) \left[ r_2 \in meauth(r_1) \rightarrow \left[ \left( \forall s \in S \right) \left[ r_1 \in authr(s) \rightarrow r_2 \notin authr(s) \right] \right] \right]$$
Key Points

- Hybrid policies deal with both confidentiality and integrity
  - Different combinations of these
- ORCON model neither MAC nor DAC
  - Actually, a combination
- RBAC model controls access based on functionality
Overview

• Classical Cryptography
  – Cæsar cipher
  – DES

• Public Key Cryptography
  – Diffie-Hellman
  – RSA

• Cryptographic Checksums
  – HMAC
Cryptosystem

- Quintuple \((E, D, M, K, C)\)
  - \(M\) set of plaintexts
  - \(K\) set of keys
  - \(C\) set of ciphertexts
  - \(E\) set of encryption functions \(e: M \times K \rightarrow C\)
  - \(D\) set of decryption functions \(d: C \times K \rightarrow M\)
Example

- Example: Cæsar cipher
  - $\mathcal{M} = \{\text{sequences of letters}\}$
  - $\mathcal{K} = \{i \mid i \text{ is an integer and } 0 \leq i \leq 25\}$
  - $\mathcal{E} = \{E_k \mid k \in \mathcal{K} \text{ and for all letters } m,\}
    \quad E_k(m) = (m + k) \mod 26\}$
  - $\mathcal{D} = \{D_k \mid k \in \mathcal{K} \text{ and for all letters } c,\}
    \quad D_k(c) = (26 + c - k) \mod 26\}$
  - $C = \mathcal{M}$
Attacks

- Opponent whose goal is to break cryptosystem is the *adversary*
  - Assume adversary knows algorithm used, but not key
- Three types of attacks:
  - *ciphertext only*: adversary has only ciphertext; goal is to find plaintext, possibly key
  - *known plaintext*: adversary has ciphertext, corresponding plaintext; goal is to find key
  - *chosen plaintext*: adversary may supply plaintexts and obtain corresponding ciphertext; goal is to find key
Basis for Attacks

• Mathematical attacks
  – Based on analysis of underlying mathematics
• Statistical attacks
  – Make assumptions about the distribution of letters, pairs of letters (digrams), triplets of letters (trigrams), \textit{etc}.
    • Called \textit{models of the language}
  – Examine ciphertext, correlate properties with the assumptions.
Classical Cryptography

- Sender, receiver share common key
  - Keys may be the same, or trivial to derive from one another
  - Sometimes called \textit{symmetric cryptography}

- Two basic types
  - Transposition ciphers
  - Substitution ciphers
  - Combinations are called \textit{product ciphers}
Transposition Cipher

• Rearrange letters in plaintext to produce ciphertext
• Example (Rail-Fence Cipher)
  – Plaintext is HELLO WORLD
  – Rearrange as
    HLOOL
    ELWRD
  – Ciphertext is HLOOL ELWRD
Attacking the Cipher

• Anagramming
  – If 1-gram frequencies match English frequencies, but other $n$-gram frequencies do not, probably transposition
  – Rearrange letters to form $n$-grams with highest frequencies
Example

- Ciphertext: HLOOLELWRD
- Frequencies of 2-grams beginning with H
  - HE  0.0305
  - HO  0.0043
  - HL, HW, HR, HD < 0.0010
- Frequencies of 2-grams ending in H
  - WH  0.0026
  - EH, LH, OH, RH, DH ≤ 0.0002
- Implies E follows H
Example

- Arrange so the H and E are adjacent
  
  HE
  LL
  OW
  OR
  LD

- Read off across, then down, to get original plaintext
Substitution Ciphers

• Change characters in plaintext to produce ciphertext

• Example (Cæsar cipher)
  – Plaintext is HELLO WORLD
  – Change each letter to the third letter following it (X goes to A, Y to B, Z to C)
    • Key is 3, usually written as letter ‘D’
  – Ciphertext is KHOOR ZRUOG
Attacking the Cipher

• Exhaustive search
  – If the key space is small enough, try all possible keys until you find the right one
  – Cæsar cipher has 26 possible keys

• Statistical analysis
  – Compare to 1-gram model of English
Statistical Attack

• Compute frequency of each letter in ciphertext:
  
  G 0.1  H 0.1  K 0.1  O 0.3  
  R 0.2  U 0.1  Z 0.1  

• Apply 1-gram model of English
  – Frequency of characters (1-grams) in English
    is on next slide
## Character Frequencies

<table>
<thead>
<tr>
<th>a</th>
<th>0.080</th>
<th>h</th>
<th>0.060</th>
<th>n</th>
<th>0.070</th>
<th>t</th>
<th>0.090</th>
</tr>
</thead>
<tbody>
<tr>
<td>b</td>
<td>0.015</td>
<td>i</td>
<td>0.065</td>
<td>o</td>
<td>0.080</td>
<td>u</td>
<td>0.030</td>
</tr>
<tr>
<td>c</td>
<td>0.030</td>
<td>j</td>
<td>0.005</td>
<td>p</td>
<td>0.020</td>
<td>v</td>
<td>0.010</td>
</tr>
<tr>
<td>d</td>
<td>0.040</td>
<td>k</td>
<td>0.005</td>
<td>q</td>
<td>0.002</td>
<td>w</td>
<td>0.015</td>
</tr>
<tr>
<td>e</td>
<td>0.130</td>
<td>l</td>
<td>0.035</td>
<td>r</td>
<td>0.065</td>
<td>x</td>
<td>0.005</td>
</tr>
<tr>
<td>f</td>
<td>0.020</td>
<td>m</td>
<td>0.030</td>
<td>s</td>
<td>0.060</td>
<td>y</td>
<td>0.020</td>
</tr>
<tr>
<td>g</td>
<td>0.015</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>z</td>
<td>0.002</td>
</tr>
</tbody>
</table>
Statistical Analysis

- $f(c)$ frequency of character $c$ in ciphertext
- $\varphi(i)$ correlation of frequency of letters in ciphertext with corresponding letters in English, assuming key is $i$
  
  \[ \varphi(i) = \sum_{0 \leq c \leq 25} f(c)p(c - i) \]

  - so here,
  
  \[ \varphi(i) = 0.1p(6 - i) + 0.1p(7 - i) + 0.1p(10 - i) + 0.3p(14 - i) + 0.2p(17 - i) + 0.1p(20 - i) + 0.1p(25 - i) \]

  - $p(x)$ is frequency of character $x$ in English
Correlation: $\varphi(i)$ for $0 \leq i \leq 25$

<table>
<thead>
<tr>
<th>$i$</th>
<th>$\varphi(i)$</th>
<th>$i$</th>
<th>$\varphi(i)$</th>
<th>$i$</th>
<th>$\varphi(i)$</th>
<th>$i$</th>
<th>$\varphi(i)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0.0482</td>
<td>7</td>
<td>0.0442</td>
<td>13</td>
<td>0.0520</td>
<td>19</td>
<td>0.0315</td>
</tr>
<tr>
<td>1</td>
<td>0.0364</td>
<td>8</td>
<td>0.0202</td>
<td>14</td>
<td>0.0535</td>
<td>20</td>
<td>0.0302</td>
</tr>
<tr>
<td>2</td>
<td>0.0410</td>
<td>9</td>
<td>0.0267</td>
<td>15</td>
<td>0.0226</td>
<td>21</td>
<td>0.0517</td>
</tr>
<tr>
<td>3</td>
<td>0.0575</td>
<td>10</td>
<td>0.0635</td>
<td>16</td>
<td>0.0322</td>
<td>22</td>
<td>0.0380</td>
</tr>
<tr>
<td>4</td>
<td>0.0252</td>
<td>11</td>
<td>0.0262</td>
<td>17</td>
<td>0.0392</td>
<td>23</td>
<td>0.0370</td>
</tr>
<tr>
<td>5</td>
<td>0.0190</td>
<td>12</td>
<td>0.0325</td>
<td>18</td>
<td>0.0299</td>
<td>24</td>
<td>0.0316</td>
</tr>
<tr>
<td>6</td>
<td>0.0660</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>25</td>
<td>0.0430</td>
</tr>
</tbody>
</table>
The Result

• Most probable keys, based on $\varphi$:
  – $i = 6$, $\varphi(i) = 0.0660$
    • plaintext EBIIL TLOLA
  – $i = 10$, $\varphi(i) = 0.0635$
    • plaintext AXEEH PHKEW
  – $i = 3$, $\varphi(i) = 0.0575$
    • plaintext HELLO WORLD
  – $i = 14$, $\varphi(i) = 0.0535$
    • plaintext WTAAD LDGAS
• Only English phrase is for $i = 3$
  – That’s the key (3 or ‘D’)
Cæsar’s Problem

• Key is too short
  – Can be found by exhaustive search
  – Statistical frequencies not concealed well
    • They look too much like regular English letters

• So make it longer
  – Multiple letters in key
  – Idea is to smooth the statistical frequencies to make cryptanalysis harder
Vigènere Cipher

- Like Cæsar cipher, but use a phrase
- Example
  - Message THE BOY HAS THE BALL
  - Key VIG
  - Encipher using Cæsar cipher for each letter:
    - key VIGVIGVIGVIGVIGVIGVIGVIGVIGVIG
    - plain THEBOYHASTHEBALL
    - cipher OPKWWECIYOPKWIRG
Relevant Parts of Tableau

- Tableau shown has relevant rows, columns only
- Example encipherments:
  - Key V, letter T: follow V column down to T row (giving “O”)
  - Key I, letter H: follow I column down to H row (giving “P”)

<table>
<thead>
<tr>
<th>A</th>
<th>G</th>
<th>I</th>
<th>V</th>
</tr>
</thead>
<tbody>
<tr>
<td>B</td>
<td>H</td>
<td>J</td>
<td>W</td>
</tr>
<tr>
<td>E</td>
<td>L</td>
<td>M</td>
<td>Z</td>
</tr>
<tr>
<td>H</td>
<td>N</td>
<td>P</td>
<td>C</td>
</tr>
<tr>
<td>L</td>
<td>R</td>
<td>T</td>
<td>G</td>
</tr>
<tr>
<td>O</td>
<td>U</td>
<td>W</td>
<td>J</td>
</tr>
<tr>
<td>S</td>
<td>Y</td>
<td>A</td>
<td>N</td>
</tr>
<tr>
<td>T</td>
<td>Z</td>
<td>B</td>
<td>O</td>
</tr>
<tr>
<td>Y</td>
<td>E</td>
<td>H</td>
<td>T</td>
</tr>
</tbody>
</table>
Useful Terms

• *period*: length of key
  – In earlier example, period is 3

• *tableau*: table used to encipher and decipher
  – Vigènere cipher has key letters on top, plaintext letters on the left

• *polyalphabetic*: the key has several different letters
  – Cæsar cipher is monoalphabetic
One-Time Pad

• A Vigenère cipher with a random key at least as long as the message
  – Provably unbreakable
  – Why? Look at ciphertext DXQR. Equally likely to correspond to plaintext DOIT (key AJIY) and to plaintext DONT (key AJDY) and any other 4 letters
  – Warning: keys must be random, or you can attack the cipher by trying to regenerate the key
    • Approximations, such as using pseudorandom number generators to generate keys, are not random
Overview of the DES

- A block cipher:
  - encrypts blocks of 64 bits using a 64 bit key
  - outputs 64 bits of ciphertext

- A product cipher
  - basic unit is the bit
  - performs both substitution and transposition (permutation) on the bits

- Cipher consists of 16 rounds (iterations) each with a round key generated from the user-supplied key
Generation of Round Keys

- Round keys are 48 bits each
Encipherment

input

IP

L_0

! f
K_1

L_1 = R_0

R_0

R_1 = L_0 \oplus f(R_0, K_1)

R_16 = L_{15} \oplus f(R_{15}, K_{16})

L_{16} = R_{15}

IP^{-1}

output

May 24, 2005
ECS 235, Computer and Information Security
Slide #37
The $f$ Function

$R_{i-1}$ (32 bits)\n
$E$\n
$R_{i-1}$ (48 bits)\n
$K_i$ (48 bits)\n
$S1$ $S2$ $S3$ $S4$ $S5$ $S6$ $S7$ $S8$\n
6 bits into each\n
4 bits out of each\n
$P$\n
32 bits
Controversy

• Considered too weak
  – Diffie, Hellman said in a few years technology would allow DES to be broken in days
    • Design using 1999 technology published
  – Design decisions not public
    • S-boxes may have backdoors
Undesirable Properties

- 4 weak keys
  - They are their own inverses
- 12 semi-weak keys
  - Each has another semi-weak key as inverse
- Complementation property
  - $\text{DES}_k(m) = c \Rightarrow \text{DES}_k(m') = c'$
- S-boxes exhibit irregular properties
  - Distribution of odd, even numbers non-random
  - Outputs of fourth box depends on input to third box
Differential Cryptanalysis

• A chosen ciphertext attack
  – Requires $2^{47}$ plaintext, ciphertext pairs
• Revealed several properties
  – Small changes in S-boxes reduce the number of pairs needed
  – Making every bit of the round keys independent does not impede attack
• Linear cryptanalysis improves result
  – Requires $2^{43}$ plaintext, ciphertext pairs
DES Modes

- **Electronic Code Book Mode (ECB)**
  - Encipher each block independently

- **Cipher Block Chaining Mode (CBC)**
  - Xor each block with previous ciphertext block
  - Requires an initialization vector for the first one

- **Encrypt-Decrypt-Encrypt Mode (2 keys: \(k, k'\))**
  - \(c = \text{DES}_k(\text{DES}_{k'}^{-1}(\text{DES}_k(m)))\)

- **Encrypt-Encrypt-Encrypt Mode (3 keys: \(k, k', k''\))**
  - \(c = \text{DES}_k(\text{DES}_{k'}(\text{DES}_{k''}(m)))\)
CBC Mode Encryption

\[ \begin{align*}
\text{init. vector} & \quad m_1 \oplus \text{DES} \quad m_2 \\
\text{DES} & \quad c_1 \\
\text{sent} & \quad \text{sent} \\
\ldots &
\end{align*} \]
CBC Mode Decryption

\[
\begin{align*}
\text{init. vector} & \rightarrow c_1 \oplus \text{DES} \rightarrow m_1 \\
c_1 & \rightarrow c_2 \oplus \text{DES} \rightarrow m_2 \\
& \quad \ldots
\end{align*}
\]
Self-Healing Property

- Initial message
  - 3231343336353837 3231343336353837
  - 3231343336353837 3231343336353837

- Received as (underlined 4c should be 4b)
  - ef7c4cb2b4ce6f3b f6266e3a97af0e2c
  - 746ab9a6308f4256 33e60b451b09603d

- Which decrypts to
  - efca61e19f4836f1
  - 3231333336353837
  - 3231343336353837
  - Incorrect bytes underlined
  - Plaintext “heals” after 2 blocks
Current Status of DES

• Design for computer system, associated software that could break any DES-enciphered message in a few days published in 1998
• Several challenges to break DES messages solved using distributed computing
• NIST selected Rijndael as Advanced Encryption Standard, successor to DES
  – Designed to withstand attacks that were successful on DES
Public Key Cryptography

• Two keys
  – *Private key* known only to individual
  – *Public key* available to anyone
    • Public key, private key inverses

• Idea
  – Confidentiality: encipher using public key, decipher using private key
  – Integrity/authentication: encipher using private key, decipher using public one
Requirements

1. It must be computationally easy to encipher or decipher a message given the appropriate key
2. It must be computationally infeasible to derive the private key from the public key
3. It must be computationally infeasible to determine the private key from a chosen plaintext attack
Diffie-Hellman

• Compute a common, shared key
  – Called a *symmetric key exchange protocol*

• Based on discrete logarithm problem
  – Given integers $n$ and $g$ and prime number $p$, compute $k$ such that $n = g^k \mod p$
  – Solutions known for small $p$
  – Solutions computationally infeasible as $p$ grows large
Algorithm

- Constants: prime $p$, integer $g \neq 0, 1, p-1$
  - Known to all participants
- Anne chooses private key $k_{Anne}$, computes public key $K_{Anne} = g^{k_{Anne}} \mod p$
- To communicate with Bob, Anne computes $K_{shared} = K_{Bob}^{k_{Anne}} \mod p$
- To communicate with Anne, Bob computes $K_{shared} = K_{Anne}^{k_{Bob}} \mod p$
  - It can be shown these keys are equal
Example

• Assume $p = 53$ and $g = 17$
  – Then $K_{Alice} = 17^5 \mod 53 = 40$

• Alice chooses $k_{Alice} = 5$
  – Then $K_{Bob} = 17^7 \mod 53 = 6$

• Bob chooses $k_{Bob} = 7$
  – Then $K_{Bob} = 17^7 \mod 53 = 6$

• Shared key:
  – $K_{Bob}^{k_{Alice}} \mod p = 6^5 \mod 53 = 38$
  – $K_{Alice}^{k_{Bob}} \mod p = 40^7 \mod 53 = 38$
RSA

- Exponentiation cipher
- Relies on the difficulty of determining the number of numbers relatively prime to a large integer $n$
Background

• Totient function $\phi(n)$
  – Number of positive integers less than $n$ and relatively prime to $n$
    • *Relatively prime* means with no factors in common with $n$
  • Example: $\phi(10) = 4$
    – 1, 3, 7, 9 are relatively prime to 10
  • Example: $\phi(21) = 12$
    – 1, 2, 4, 5, 8, 10, 11, 13, 16, 17, 19, 20 are relatively prime to 21
Algorithm

• Choose two large prime numbers $p$, $q$
  – Let $n = pq$; then $\phi(n) = (p-1)(q-1)$
  – Choose $e < n$ such that $e$ is relatively prime to $\phi(n)$.
  – Compute $d$ such that $ed \mod \phi(n) = 1$
• Public key: $(e, n)$; private key: $d$
• Encipher: $c = m^e \mod n$
• Decipher: $m = c^d \mod n$
Example: Confidentiality

- Take \( p = 7, \, q = 11 \), so \( n = 77 \) and \( \phi(n) = 60 \)
- Alice chooses \( e = 17 \), making \( d = 53 \)
- Bob wants to send Alice secret message HELLO (07 04 11 11 14)
  - \( 07^{17} \mod 77 = 28 \)
  - \( 04^{17} \mod 77 = 16 \)
  - \( 11^{17} \mod 77 = 44 \)
  - \( 11^{17} \mod 77 = 44 \)
  - \( 14^{17} \mod 77 = 42 \)
- Bob sends 28 16 44 44 42
Example

- Alice receives 28 16 44 44 42
- Alice uses private key, $d = 53$, to decrypt message:
  - $28^{53} \mod 77 = 07$
  - $16^{53} \mod 77 = 04$
  - $44^{53} \mod 77 = 11$
  - $44^{53} \mod 77 = 11$
  - $42^{53} \mod 77 = 14$
- Alice translates message to letters to read HELLO
  - No one else could read it, as only Alice knows her private key and that is needed for decryption
Example:
Integrity/Authentication

- Take $p = 7$, $q = 11$, so $n = 77$ and $\phi(n) = 60$
- Alice chooses $e = 17$, making $d = 53$
- Alice wants to send Bob message HELLO (07 04 11 11 14) so Bob knows it is what Alice sent (no changes in transit, and authenticated)
  - $07^{53} \mod 77 = 35$
  - $04^{53} \mod 77 = 09$
  - $11^{53} \mod 77 = 44$
  - $11^{53} \mod 77 = 44$
  - $14^{53} \mod 77 = 49$
- Alice sends 35 09 44 44 49
Example

- Bob receives 35 09 44 44 49
- Bob uses Alice’s public key, $e = 17$, $n = 77$, to decrypt message:
  - $35^{17} \mod 77 = 07$
  - $09^{17} \mod 77 = 04$
  - $44^{17} \mod 77 = 11$
  - $44^{17} \mod 77 = 11$
  - $49^{17} \mod 77 = 14$
- Bob translates message to letters to read HELLO
  - Alice sent it as only she knows her private key, so no one else could have enciphered it
  - If (enciphered) message’s blocks (letters) altered in transit, would not decrypt properly
Example: Both

- Alice wants to send Bob message HELLO both enciphered and authenticated (integrity-checked)
  - Alice’s keys: public (17, 77); private: 53
  - Bob’s keys: public: (37, 77); private: 13

- Alice enciphers HELLO (07 04 11 11 14):
  - \((07^{53} \mod 77)^{37} \mod 77 = 07\)
  - \((04^{53} \mod 77)^{37} \mod 77 = 37\)
  - \((11^{53} \mod 77)^{37} \mod 77 = 44\)
  - \((11^{53} \mod 77)^{37} \mod 77 = 44\)
  - \((14^{53} \mod 77)^{37} \mod 77 = 14\)

- Alice sends 07 37 44 44 14
Security Services

• Confidentiality
  – Only the owner of the private key knows it, so text enciphered with public key cannot be read by anyone except the owner of the private key

• Authentication
  – Only the owner of the private key knows it, so text enciphered with private key must have been generated by the owner
More Security Services

- **Integrity**
  - Enciphered letters cannot be changed undetectably without knowing private key

- **Non-Repudiation**
  - Message enciphered with private key came from someone who knew it
Warnings

• Encipher message in blocks considerably larger than the examples here
  – If 1 character per block, RSA can be broken using statistical attacks (just like classical cryptosystems)
  – Attacker cannot alter letters, but can rearrange them and alter message meaning
    • Example: reverse enciphered message of text ON to get NO
Cryptographic Checksums

- Mathematical function to generate a set of \( k \) bits from a set of \( n \) bits (where \( k \leq n \)).
  - \( k \) is smaller than \( n \) except in unusual circumstances

- Example: ASCII parity bit
  - ASCII has 7 bits; 8th bit is “parity”
  - Even parity: even number of 1 bits
  - Odd parity: odd number of 1 bits
Example Use

• Bob receives “10111101” as bits.
  – Sender is using even parity; 6 1 bits, so character was received correctly
    • Note: could be garbled, but 2 bits would need to have been changed to preserve parity
  – Sender is using odd parity; even number of 1 bits, so character was not received correctly
Definition

• Cryptographic checksum $h: A \rightarrow B$:
  1. For any $x \in A$, $h(x)$ is easy to compute
  2. For any $y \in B$, it is computationally infeasible to find $x \in A$ such that $h(x) = y$
  3. It is computationally infeasible to find two inputs $x, x' \in A$ such that $x \neq x'$ and $h(x) = h(x')$
     – Alternate form (stronger): Given any $x \in A$, it is computationally infeasible to find a different $x' \in A$ such that $h(x) = h(x')$. 
Collisions

- If \( x \neq x' \) and \( h(x) = h(x') \), \( x \) and \( x' \) are a collision
  - Pigeonhole principle: if there are \( n \) containers for \( n+1 \) objects, then at least one container will have 2 objects in it.
  - Application: if there are 32 files and 8 possible cryptographic checksum values, at least one value corresponds to at least 4 files
Keys

- Keyed cryptographic checksum: requires cryptographic key
  - DES in chaining mode: encipher message, use last $n$ bits. Requires a key to encipher, so it is a keyed cryptographic checksum.

- Keyless cryptographic checksum: requires no cryptographic key
  - MD5 and SHA-1 are best known; others include MD4, HAVAL, and Snefru
HMAC

- Make keyed cryptographic checksums from keyless cryptographic checksums
- \( h \) keyless cryptographic checksum function that takes data in blocks of \( b \) bytes and outputs blocks of \( l \) bytes. \( k' \) is cryptographic key of length \( b \) bytes
  - If short, pad with 0 bytes; if long, hash to length \( b \)
- \( ipad \) is 00110110 repeated \( b \) times
- \( opad \) is 01011100 repeated \( b \) times
- \( \text{HMAC-} h(k, m) = h(k' \oplus opad \| h(k' \oplus ipad \| m)) \)
  - \( \oplus \) exclusive or, \( \| \) concatenation
Key Points

• Two main types of cryptosystems: classical and public key
• Classical cryptosystems encipher and decipher using the same key
  – Or one key is easily derived from the other
• Public key cryptosystems encipher and decipher using different keys
  – Computationally infeasible to derive one from the other
• Cryptographic checksums provide a check on integrity
Overview

• Access control lists
• Capability lists
• Locks and keys
• Rings-based access control
• Propagated access control lists
Access Control Lists

- Columns of access control matrix

<table>
<thead>
<tr>
<th></th>
<th>file1</th>
<th>file2</th>
<th>file3</th>
</tr>
</thead>
<tbody>
<tr>
<td>Andy</td>
<td>rx</td>
<td>r</td>
<td>rwo</td>
</tr>
<tr>
<td>Betty</td>
<td>rwxo</td>
<td>r</td>
<td></td>
</tr>
<tr>
<td>Charlie</td>
<td>rx</td>
<td>rwo</td>
<td>w</td>
</tr>
</tbody>
</table>

ACLs:
- file1: { (Andy, rx) (Betty, rwxo) (Charlie, rx) }
- file2: { (Andy, r) (Betty, r) (Charlie, rwo) }
- file3: { (Andy, rwo) (Charlie, w) }
Default Permissions

• Normal: if not named, *no* rights over file
  – Principle of Fail-Safe Defaults

• If many subjects, may use groups or wildcards in ACL
  – UNICOS: entries are (*user*, *group*, *rights*)
    • If *user* is in *group*, has rights over file
    • ‘*’ is wildcard for *user*, *group*
      – (holly, *, r): holly can read file regardless of her group
      – (*, gleep, w): anyone in group gleep can write file
Abbreviations

• ACLs can be long … so combine users
  – UNIX: 3 classes of users: owner, group, rest
    – rwx rwx rwx
      rest
      group
      owner
  – Ownership assigned based on creating process
    • Some systems: if directory has setgid permission, file group owned by group of directory (SunOS, Solaris)
ACLs + Abbreviations

- Augment abbreviated lists with ACLs
  - Intent is to shorten ACL
- ACLs override abbreviations
  - Exact method varies
- Example: IBM AIX
  - Base permissions are abbreviations, extended permissions are ACLs with user, group
  - ACL entries can add rights, but on deny, access is denied
Permissions in IBM AIX

attributes:
base permissions
  owner(bishop):       rw-
  group(sys):          r--
  others:              --

extended permissions enabled
  specify    rw-         u:holly
  permit     -w-          u:heidi, g=sys
  permit     rw-          u:matt
  deny       -w-          u:holly, g=faculty
ACL Modification

• Who can do this?
  – Creator is given *own* right that allows this
  – System R provides a *grant* modifier (like a copy flag) allowing a right to be transferred, so ownership not needed
  • Transferring right to another modifies ACL
Privileged Users

• Do ACLs apply to privileged users (*root*)?
  – Solaris: abbreviated lists do not, but full-blown ACL entries do
  – Other vendors: varies
Groups and Wildcards

• Classic form: no; in practice, usually
  – AIX: base perms gave group sys read only
    \[ \text{permit } -w- \ u:heidi, \ g=sys \]
    line adds write permission for heidi when in that group
  – UNICOS:
    • holly : gleep : r
      – user holly in group gleep can read file
    • holly : * : r
      – user holly in any group can read file
    • * : gleep : r
      – any user in group gleep can read file
Conflicts

• Deny access if any entry would deny access
  – AIX: if any entry denies access, \textit{regardless or rights given so far}, access is denied

• Apply first entry matching subject
  – Cisco routers: run packet through access control rules (ACL entries) in order; on a match, stop, and forward the packet; if no matches, deny
    • Note default is deny so honors principle of fail-safe defaults
Handling Default Permissions

• Apply ACL entry, and if none use defaults
  – Cisco router: apply matching access control rule, if any; otherwise, use default rule (deny)

• Augment defaults with those in the appropriate ACL entry
  – AIX: extended permissions augment base permissions
Revocation Question

• How do you remove subject’s rights to a file?
  – Owner deletes subject’s entries from ACL, or rights from subject’s entry in ACL

• What if ownership not involved?
  – Depends on system
  – System R: restore protection state to what it was before right was given
    • May mean deleting descendent rights too …
Windows NT ACLs

- Different sets of rights
  - Basic: read, write, execute, delete, change permission, take ownership
  - Generic: no access, read (read/execute), change (read/write/execute/delete), full control (all), special access (assign any of the basics)
  - Directory: no access, read (read/execute files in directory), list, add, add and read, change (create, add, read, execute, write files; delete subdirectories), full control, special access
Accessing Files

- User not in file’s ACL nor in any group named in file’s ACL: deny access
- ACL entry denies user access: deny access
- Take union of rights of all ACL entries giving user access: user has this set of rights over file
Capability Lists

- Rows of access control matrix

<table>
<thead>
<tr>
<th></th>
<th>file1</th>
<th>file2</th>
<th>file3</th>
</tr>
</thead>
<tbody>
<tr>
<td>Andy</td>
<td>rx</td>
<td>r</td>
<td>rwo</td>
</tr>
<tr>
<td>Betty</td>
<td>rwxo</td>
<td>r</td>
<td></td>
</tr>
<tr>
<td>Charlie</td>
<td>rx</td>
<td>rwo</td>
<td>w</td>
</tr>
</tbody>
</table>

C-Lists:
- Andy: \{(file1, rx) \ (file2, r) \ (file3, rwo)\}
- Betty: \{(file1, rwxo) \ (file2, r)\}
- Charlie: \{(file1, rx) \ (file2, rwo) \ (file3, w)\}
Semantics

• Like a bus ticket
  – Mere possession indicates rights that subject has over object
  – Object identified by capability (as part of the token)
    • Name may be a reference, location, or something else
  – Architectural construct in capability-based addressing; this just focuses on protection aspects

• Must prevent process from altering capabilities
  – Otherwise subject could change rights encoded in capability or object to which they refer
Implementation

• Tagged architecture
  – Bits protect individual words
    • B5700: tag was 3 bits and indicated how word was to be treated (pointer, type, descriptor, etc.)

• Paging/segmentation protections
  – Like tags, but put capabilities in a read-only segment or page
    • CAP system did this
  – Programs must refer to them by pointers
    • Otherwise, program could use a copy of the capability—which it could modify
Implementation (con’t)

- Cryptography
  - Associate with each capability a cryptographic checksum enciphered using a key known to OS
  - When process presents capability, OS validates checksum
  - Example: Amoeba, a distributed capability-based system
    - Capability is (name, creating_server, rights, check_field) and is given to owner of object
    - check_field is 48-bit random number; also stored in table corresponding to creating_server
    - To validate, system compares check_field of capability with that stored in creating_server table
    - Vulnerable if capability disclosed to another process
Amplifying

• Allows *temporary* increase of privileges
• Needed for modular programming
  – Module pushes, pops data onto stack
    
    module stack ... endmodule.
  – Variable $x$ declared of type stack
    
    var $x$: module;
  – *Only* stack module can alter, read $x$
    • So process doesn’t get capability, but needs it when $x$ is referenced—a problem!
  – Solution: give process the required capabilities while it is in module
Examples

• HYDRA: templates
  – Associated with each procedure, function in module
  – Adds rights to process capability while the procedure or function is being executed
  – Rights deleted on exit

• Intel iAPX 432: access descriptors for objects
  – These are really capabilities
  – 1 bit in this controls amplification
  – When ADT constructed, permission bits of type control object set to what procedure needs
  – On call, if amplification bit in this permission is set, the above bits or’ed with rights in access descriptor of object being passed
Revocation

- Scan all C-lists, remove relevant capabilities
  - Far too expensive!
- Use indirection
  - Each object has entry in a global object table
  - Names in capabilities name the entry, not the object
    - To revoke, zap the entry in the table
    - Can have multiple entries for a single object to allow control of different sets of rights and/or groups of users for each object
  - Example: Amoeba: owner requests server change random number in server table
    - All capabilities for that object now invalid
Limits

• Problems if you don’t control copying of capabilities

The capability to write file \( lough \) is Low, and Heidi is High so she reads (copies) the capability; now she can write to a Low file, violating the *-property!
Remedies

• Label capability itself
  – Rights in capability depends on relation between its compartment and that of object to which it refers
    • In example, as as capability copied to High, and High dominates object compartment (Low), write right removed

• Check to see if passing capability violates security properties
  – In example, it does, so copying refused

• Distinguish between “read” and “copy capability”
  – Take-Grant Protection Model does this (“read”, “take”)
ACLs vs. Capabilities

• Both theoretically equivalent; consider 2 questions
  1. Given a subject, what objects can it access, and how?
  2. Given an object, what subjects can access it, and how?
     – ACLs answer second easily; C-Lists, first

• Suggested that the second question, which in the past has been of most interest, is the reason ACL-based systems more common than capability-based systems
  – As first question becomes more important (in incident response, for example), this may change
Locks and Keys

• Associate information (*lock*) with object, information (*key*) with subject
  – Latter controls what the subject can access and how
  – Subject presents key; if it corresponds to any of the locks on the object, access granted

• This can be dynamic
  – ACLs, C-Lists static and must be manually changed
  – Locks and keys can change based on system constraints, other factors (not necessarily manual)
Cryptographic Implementation

- Enciphering key is lock; deciphering key is key
  - Encipher object $o$; store $E_k(o)$
  - Use subject’s key $k'$ to compute $D_k(E_k(o))$
  - Any of $n$ can access $o$: store $o' = (E_1(o), \ldots, E_n(o))$
  - Requires consent of all $n$ to access $o$: store $o' = (E_1(E_2(\ldots(E_n(o))\ldots)))$
Example: IBM

- IBM 370: process gets access key; pages get storage key and fetch bit
  - Fetch bit clear: read access only
  - Fetch bit set, access key 0: process can write to (any) page
  - Fetch bit set, access key matches storage key: process can write to page
  - Fetch bit set, access key non-zero and does not match storage key: no access allowed
Example: Cisco Router

- **Dynamic access control lists**
  
  ```
  access-list 100 permit tcp any host 10.1.1.1 eq telnet
  access-list 100 dynamic test timeout 180 permit ip any host \\
  10.1.2.3 time-range my-time
  time-range my-time
  periodic weekdays 9:00 to 17:00
  line vty 0 2
  login local
 autocommand access-enable host timeout 10
  ```

- **Limits external access to 10.1.2.3 to 9AM–5PM**
  
  - Adds temporary entry for connecting host once user supplies name, password to router
  - Connections good for 180 minutes
    - Drops access control entry after that
Type Checking

• Lock is type, key is operation
  – Example: UNIX system call write can’t work on directory object but does work on file
  – Example: split I&D space of PDP-11
  – Example: countering buffer overflow attacks on the stack by putting stack on non-executable pages/segments
    • Then code uploaded to buffer won’t execute
    • Does not stop other forms of this attack, though …
More Examples

• **LOCK system:**
  – Compiler produces “data”
  – Trusted process must change this type to “executable” before program can be executed

• **Sidewinder firewall**
  – Subjects assigned domain, objects assigned type
    • Example: ingress packets get one type, egress packets another
  – All actions controlled by type, so ingress packets cannot masquerade as egress packets (and vice versa)
Sharing Secrets

• Implements separation of privilege
• Use \((t, n)\)-threshold scheme
  – Data divided into \(n\) parts
  – Any \(t\) parts sufficient to derive original data
• Or-access and and-access can do this
  – Increases the number of representations of data rapidly as \(n, t\) grow
  – Cryptographic approaches more common
Shamir’s Scheme

• Goal: use \((t, n)\)-threshold scheme to share cryptographic key encoding data
  – Based on Lagrange polynomials
  – Idea: take polynomial \(p(x)\) of degree \(t-1\), set constant term \(p(0)\) to key
  – Compute value of \(p\) at \(n\) points, \(excluding x = 0\)
  – By algebra, need values of \(p\) at any \(t\) distinct points to derive polynomial, and hence constant term (key)
Ring-Based Access Control

- Process (segment) accesses another segment
  - Read
  - Execute
- *Gate* is an entry point for calling segment
- Rights:
  - $r$ read
  - $w$ write
  - $a$ append
  - $e$ execute
Reading/Writing/Appending

- Procedure executing in ring $r$
- Data segment with access bracket $(a_1, a_2)$
- Mandatory access rule
  - $r \leq a_1$ allow access
  - $a_1 < r \leq a_2$ allow $r$ access; not $w, a$ access
  - $a_2 < r$ deny all access
Executing

• Procedure executing in ring $r$
• Call procedure in segment with *access bracket* $(a_1, a_2)$ and *call bracket* $(a_2, a_3)$
  – Often written $(a_1, a_2, a_3)$
• Mandatory access rule
  – $r < a_1$ allow access; ring-crossing fault
  – $a_1 \leq r \leq a_2$ allow access; no ring-crossing fault
  – $a_2 < r \leq a_3$ allow access if through valid gate
  – $a_3 < r$ deny all access
Versions

- Multics
  - 8 rings (from 0 to 7)
- Digital Equipment’s VAX
  - 4 levels of privilege: user, monitor, executive, kernel
- Older systems
  - 2 levels of privilege: user, supervisor
PACLs

- Propagated Access Control List
  - Implements ORGON
- Creator kept with PACL, copies
  - Only owner can change PACL
  - Subject reads object: object’s PACL associated with subject
  - Subject writes object: subject’s PACL associated with object
- Notation: $\text{PACL}_s$ means $s$ created object; $\text{PACL}(e)$ is PACL associated with entity $e$
Multiple Creators

• Betty reads Ann’s file *dates*
  \[ \text{PACL}(\text{Betty}) = \text{PACL}_{\text{Betty}} \cap \text{PACL}(\text{dates}) = \text{PACL}_{\text{Betty}} \cap \text{PACL}_{\text{Ann}} \]

• Betty creates file *dc*
  \[ \text{PACL}(\text{dc}) = \text{PACL}_{\text{Betty}} \cap \text{PACL}_{\text{Ann}} \]

• \( \text{PACL}_{\text{Betty}} \) allows Char to access objects, but \( \text{PACL}_{\text{Ann}} \) does not; both allow June to access objects
  - June can read *dc*
  - Char cannot read *dc*
Key Points

• Access control mechanisms provide controls for users accessing files

• Many different forms
  – ACLs, capabilities, locks and keys
    • Type checking too
  – Ring-based mechanisms (Mandatory)
  – PACLs (ORCON)