Introduction

- There are two main issues:
  - Authentication
  - Authorization
- Authentication: is validating the user and the messages sent by the authenticated user.
- Authorization: refers to access control of resources after a user/message has been authenticated.
- Security primarily refers to the authentication issue. This is discussed quite nicely in chapter 7 of your text.
- For access control models we will discuss Java Authentication and Authorization Service (JAAS).
Cryptography

- Cryptography is the basis for authentication of messages.
- We need security protocols to exploit it.
- Selection of cryptographic algorithms and management of keys are critical issues for effectiveness, performance and usefulness of security mechanisms.
- Public-key cryptography is good for key distribution but inadequate for encryption of bulk data.
- Secret-key cryptography is suitable for bulk encryption tasks.
- Hybrid protocols such as SSL (Secure Socket Layer) establish a secure channel using public-key cryptography and then use it exchange secret keys for subsequent data exchanges.

10/13/2004  B.Ramamurthy

---

Historical context: the evolution of security needs

<table>
<thead>
<tr>
<th>Platforms</th>
<th>1965-75</th>
<th>1975-89</th>
<th>1990-99</th>
<th>Current</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Multi-user timesharing computers</td>
<td>Distributed systems</td>
<td>The Internet, wide-area</td>
<td>The Internet + mobile devices</td>
</tr>
<tr>
<td></td>
<td></td>
<td>based on local networks</td>
<td>services</td>
<td></td>
</tr>
<tr>
<td>Shared resources</td>
<td>Memory, files</td>
<td>Local services (e.g., NFS),</td>
<td>Email, web sites,</td>
<td>Distributed objects, mobile</td>
</tr>
<tr>
<td></td>
<td></td>
<td>local networks</td>
<td>Internet commerce</td>
<td>code</td>
</tr>
<tr>
<td>Security</td>
<td>User identification and</td>
<td>Protection of services</td>
<td>Strong security for</td>
<td>Access control for</td>
</tr>
<tr>
<td>requirements</td>
<td>authentication</td>
<td></td>
<td>commercial transactions</td>
<td>individual objects,</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>secure mobile code</td>
</tr>
<tr>
<td>Security</td>
<td>Single authority, single</td>
<td>Single authority, delegation,</td>
<td>Many authorities, no</td>
<td>Per-activity authorities,</td>
</tr>
<tr>
<td>management</td>
<td>authorization database (e.g.,</td>
<td>replicated authorization</td>
<td>network-wide authorities</td>
<td>groups with shared</td>
</tr>
<tr>
<td>environment</td>
<td>/etc/passwd)</td>
<td>databases (e.g., NIS)</td>
<td></td>
<td>Responsibilities, mass</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>authentication</td>
</tr>
</tbody>
</table>

10/13/2004  B.Ramamurthy
Encryption

Most schemes include algorithms for encrypting and decrypting messages based on secret codes called keys.

Two common models:
- Shared secret keys
- Public/private key pairs: A message encrypted with the public key of the receiver can be decrypted only by the private key of the recipient.

Familiar names for the protagonists in security protocols

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alice</td>
<td>First participant</td>
</tr>
<tr>
<td>Bob</td>
<td>Second participant</td>
</tr>
<tr>
<td>Carol</td>
<td>Participant in three- and four-party protocols</td>
</tr>
<tr>
<td>Dave</td>
<td>Participant in four-party protocols</td>
</tr>
<tr>
<td>Eve</td>
<td>Eavesdropper</td>
</tr>
<tr>
<td>Mallory</td>
<td>Malicious attacker</td>
</tr>
<tr>
<td>Sara</td>
<td>A server</td>
</tr>
</tbody>
</table>
Cryptography notations

<table>
<thead>
<tr>
<th>Notation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$K_A$</td>
<td>Alice’s secret key</td>
</tr>
<tr>
<td>$K_B$</td>
<td>Bob’s secret key</td>
</tr>
<tr>
<td>$K_{AB}$</td>
<td>Secret key shared between Alice and Bob</td>
</tr>
<tr>
<td>$K_{priv}$</td>
<td>Alice’s private key (known only to Alice)</td>
</tr>
<tr>
<td>$K_{pub}$</td>
<td>Alice’s public key (published by Alice for all to read)</td>
</tr>
</tbody>
</table>

$\{M\}_K$ Message $M$ encrypted with key $K$

$[M]_K$ Message $M$ signed with key $K$

Cryptographic Algorithms

- Plain text $\rightarrow$ cipher text $\rightarrow$ Decipher text
- $E(K,M) = \{M\}_K$ where $E$ is the encryption function, $M$ is the message and $K$ is the key.
- Decryption: $D(K,E(K,M)) = M$
- Same key is used in encrypting and decrypting. So it is called symmetric cryptography.
Stream cipher

Cryptographic algorithms

- Shannon’s principles of cryptography: introduce “confusion” (XORing, bit shifting etc.) and “diffusion” (adding noise bits to diffuse the information)
- We will look at Tiny Encryption Algorithm (TEA) as an example of symmetric algorithm and Rivest, Shamir and Adelman (RSA) an an example for asymmetric algorithms.
TEA Encryption Function

```c
void encrypt(unsigned long k[], unsigned long text[]) {
    unsigned long y = text[0], z = text[1];
    unsigned long delta = 0x9e3779b9, sum = 0; int n;
    for (n= 0; n < 32; n++) {
        sum += delta;
        y += ((z << 4) + k[0]) ^ (z+sum) ^ ((z >> 5) + k[1]);
        z += ((y << 4) + k[2]) ^ (y+sum) ^ ((y >> 5) + k[3]);
    }
    text[0] = y; text[1] = z;
}
```

TEA decryption function

```c
void decrypt(unsigned long k[], unsigned long text[]) {
    unsigned long y = text[0], z = text[1];
    unsigned long delta = 0x9e3779b9, sum = delta << 5; int n;
    for (n= 0; n < 32; n++) {
        z -= ((y << 4) + k[2]) ^ (y + sum) ^ ((y >> 5) + k[3]);
        y -= ((z << 4) + k[0]) ^ (z + sum) ^ ((z >> 5) + k[1]);
        sum -= delta;
    }
    text[0] = y; text[1] = z;
}
```
void tea(char mode, FILE *infile, FILE *outfile, unsigned long k[]) {
    /* mode is 'e' for encrypt, 'd' for decrypt, k[] is the key. */
    char ch, Text[8]; int i;
    while(!feof(infile)) {
        i = fread(Text, 1, 8, infile); /* read 8 bytes from infile into Text */
        if (i <= 0) break;
        while (i < 8) { Text[i++] = ' '}; /* pad last block with spaces */
        switch (mode) {
        case 'e':
            encrypt(k, (unsigned long*) Text); break;
        case 'd':
            decrypt(k, (unsigned long*) Text); break;
        }
        fwrite(Text, 1, 8, outfile); /* write 8 bytes from Text to outfile */
    }
}

RSA Encryption

To find a key pair $e$, $d$:
1. Choose two large prime numbers, $P$ and $Q$ (each greater than 10100), and form:
   $N = P \times Q$
   $Z = (P-1) \times (Q-1)$

   2. For $d$ choose any number that is relatively prime with $Z$ (that is, such that $d$
      has no common factors with $Z$).

   We illustrate the computations involved using small integer values for $P$
   and $Q$:
   $P = 13$, $Q = 17$ \(\Rightarrow\) $N = 221$, $Z = 192$
   $d = 5$

   3. To find $e$ solve the equation:
   $e \times d \equiv 1 \text{ mod } Z$

   That is, $e \times d$ is the smallest element divisible by $d$ in the series
   $Z+1$, $2Z+1$, $3Z+1$, ...

   $e \times d = 1 \text{ mod } 192$ \(\Rightarrow\) $e = 385$

   $385 \text{ is divisible by } d$

   $e = 385/5 = 77$
RSA Encryption (contd.)

To encrypt text using the RSA method, the plaintext is divided into equal blocks of length $k$ bits where $2^k < N$ (that is, such that the numerical value of a block is always less than $N$; in practical applications, $k$ is usually in the range 512 to 1024).

$k = 7$, since $2^7 = 128$

The function for encrypting a single block of plaintext $M$ is: $(N = P \times Q = 13 \times 17 = 221), e = 77, d = 5$:

$$E'(e, N, M) = M^e \text{ mod } N$$

for a message $M$, the ciphertext is $M^{77} \text{ mod } 221$

The function for decrypting a block of encrypted text $c$ to produce the original plaintext block is:

$$D'(d, N, c) = c^d \text{ mod } N$$

The two parameters $e, N$ can be regarded as a key for the encryption function, and similarly $d, N$ represent a key for the decryption function.

So we can write $K_e = <e, N>$ and $K_d = <d, N>$, and we get the encryption function:

$E(K_e, M) = [M]^e_N$ (the notation here indicating that the encrypted message can be decrypted only by the holder of the private key $K_d$) and $D(K_d, [M]^e_N) = M$.

$<e, N>$ - public key, $d$ – private key for a station

Application of RSA

- Lets say a person in Atlanta wants to send a message $M$ to a person in Buffalo:
- Atlanta encrypts message using Buffalo’s public key $B \rightarrow E(M, B)$
- Only Buffalo can read it using it private key $b$: $E(p, E(M, B)) \rightarrow M$
- In other words for any public/private key pair determined as previously shown, the encrypting function holds two properties:
  - $E(p, E(M, p)) \rightarrow M$
  - $E(P, E(M, p)) \rightarrow M$
How can you authenticate “sender”?

- (In real life you will use signatures: the concept of signatures is introduced.)
- Instead of sending just a simple message, Atlanta will send a signed message signed by Atlanta’s private key:
  - $E(B, E(M, a))$
- Buffalo will first decrypt using its private key and use Atlanta’s public key to decrypt the signed message:
  - $E(b, E(B, E(M, a))) \rightarrow E(M, a)$
  - $E(A, E(M, a)) \rightarrow M$

Digital Signatures

- Strong digital signatures are essential requirements of a secure system. These are needed to verify that a document is:
  - Authentic: source
  - Not forged: not fake
  - Non-repudiable: The signer cannot credibly deny that the document was signed by them.
Digest Functions

- Are functions generated to serve as signatures. Also called secure hash functions.
- It is message dependent.
- Only the Digest is encrypted using the private key.

Alice’s bank account certificate

<table>
<thead>
<tr>
<th>Field</th>
<th>Information</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Certificate type</td>
<td>Account number</td>
</tr>
<tr>
<td>2. Name</td>
<td>Alice</td>
</tr>
<tr>
<td>3. Account</td>
<td>6262626</td>
</tr>
<tr>
<td>4. Certifying authority</td>
<td>Bob’s Bank</td>
</tr>
<tr>
<td>5. Signature</td>
<td>{Digest(field 2 + field 3)}<em>{KB</em>{priv}}</td>
</tr>
</tbody>
</table>
Digital signatures with public keys

- **Signing**
  - $H(M)$
  - $E(K_{pri}, h)$
  - 128 bits

- **Verifying**
  - $D(K_{pub}(h))$
  - $h'$
  - $h = h'$?

Low-cost signatures with a shared secret key

- **Signing**
  - $H(M+K)$

- **Verifying**
  - $H(M+K)$
  - $h = h'$?
X509 Certificate format

<table>
<thead>
<tr>
<th>Subject</th>
<th>Distinguished Name, Public Key</th>
</tr>
</thead>
<tbody>
<tr>
<td>Issuer</td>
<td>Distinguished Name, Signature</td>
</tr>
<tr>
<td>Period of validity</td>
<td>Not Before Date, Not After Date</td>
</tr>
<tr>
<td>Administrative information</td>
<td>Version, Serial Number</td>
</tr>
</tbody>
</table>

Certificates are widely used in e-commerce to authenticate Subjects. A Certificate Authority is a trusted third party, which certifies Public Key's do truly belong to their claimed owners. Certificate Authorities: Verisign, CREN (Corp for Educational Research Networking), Thawte

See also Netscape SSL2.0 Certificate format: [http://wp.netscape.com/eng/security/ssl_2.0_certificate.html#SSL2cert](http://wp.netscape.com/eng/security/ssl_2.0_certificate.html#SSL2cert)

The Needham–Schroeder secret-key authentication protocol

<table>
<thead>
<tr>
<th>Header</th>
<th>Message</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. A-&gt;S:</td>
<td>A, B, N₁</td>
<td>A requests S to supply a key for communication with B.</td>
</tr>
<tr>
<td>2. S-&gt;A:</td>
<td>{N₁, B, K₁₄₈ {K₁₄₈ A}_K₄₈ }</td>
<td>S returns a message encrypted in A’s secret key, containing a newly generated key K₁₄₈ and a ‘ticket’ encrypted in B’s secret key. The nonce N₁ demonstrates that the message was sent in response to the preceding one. A believes that S sent the message because only S knows A’s secret key.</td>
</tr>
<tr>
<td>3. A-&gt;B:</td>
<td>{K₁₄₈ A}_K₄₈</td>
<td>A sends the ‘ticket’ to B.</td>
</tr>
<tr>
<td>4. B-&gt;A:</td>
<td>{N₂ {K₁₄₈ }</td>
<td>B decrypts the ticket and uses the new key K₁₄₈ to encrypt another nonce N₂.</td>
</tr>
<tr>
<td>5. A-&gt;B:</td>
<td>{N₂ - 1 }_K₁₄₈</td>
<td>A demonstrates to B that it was the sender of the previous message by returning an agreed transformation of N₂.</td>
</tr>
</tbody>
</table>
System architecture of Kerberos

SSL protocol stack
SSL handshake protocol

Client Hello
→ Server Hello

→ Certificate

→ Certificate Request

→ Server Hello Done

→ Certificate

→ Certificate Verify

→ Change Cipher Spec

→ Finished

→ Change Cipher Spec

→ Finished

Establish protocol version, session ID, cipher suite, compression method, exchange random values

Optionally send server certificate and request client certificate

Send client certificate response if requested

Change cipher suite and finish handshake

SSL handshake configuration options

<table>
<thead>
<tr>
<th>Component</th>
<th>Description</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>Key exchange method</td>
<td>the method to be used for exchange of a session key</td>
<td>RSA with public-key certificates</td>
</tr>
<tr>
<td>Cipher for data transfer</td>
<td>the block or stream cipher to be used for data</td>
<td>IDEA</td>
</tr>
<tr>
<td>Message digest function</td>
<td>for creating message authentication codes (MACs)</td>
<td>SHA</td>
</tr>
</tbody>
</table>
SSL record protocol

- Application data
- Record protocol units
- Compressed units
- MAC
- Encrypted
- TCP packet

Millicent architecture

- Vendor
- Customer
- Validation
- Spent scrip list
- Master customer secrets
- Optional secure channel based on customer secret
- Scrip layout
- Purchase (item name, scrip)
- Completion (item, scrip change)
- Scrip store
- Customer secret
- Scrip
- Value
- Scrip ID
- Properties
- Customer ID
- Expiry date
- Certificate