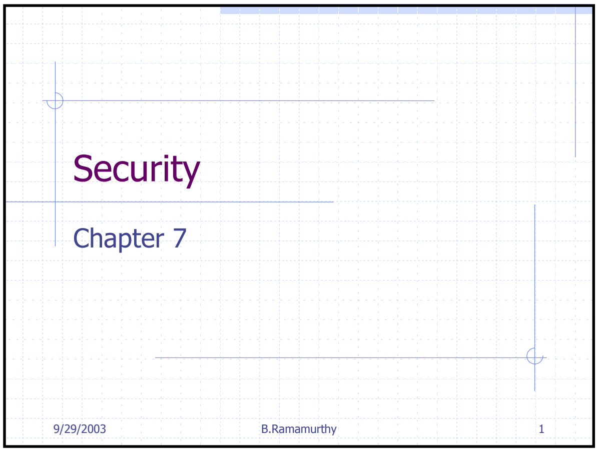
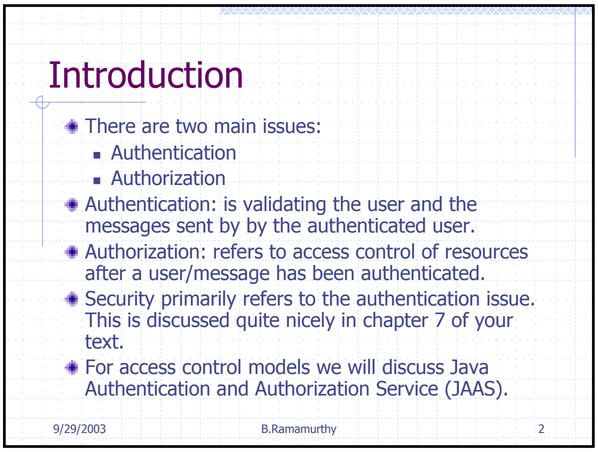


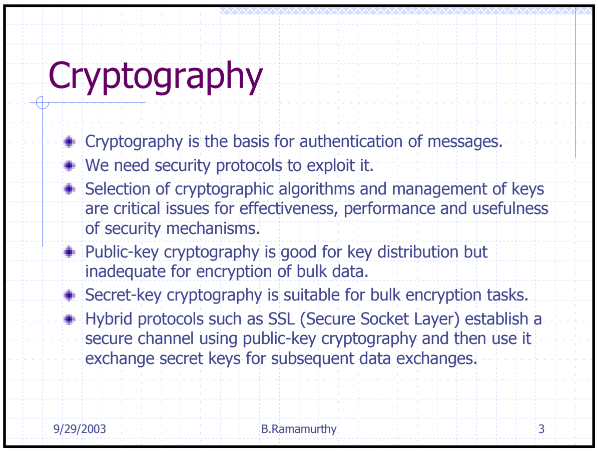
Security

Chapter 7



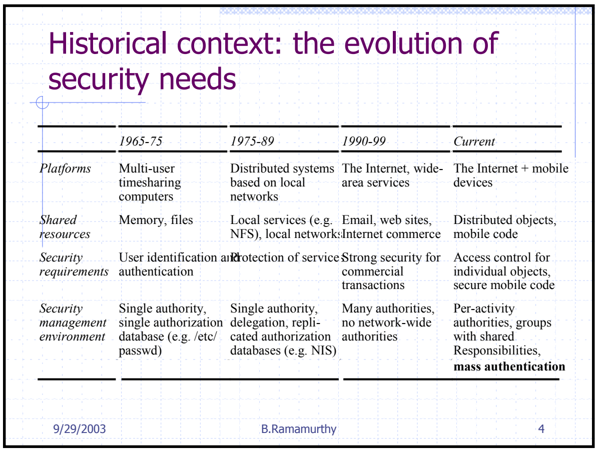
Introduction

- There are two main issues:
  - Authentication
  - Authorization
- Authentication: is validating the user and the messages sent by 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).



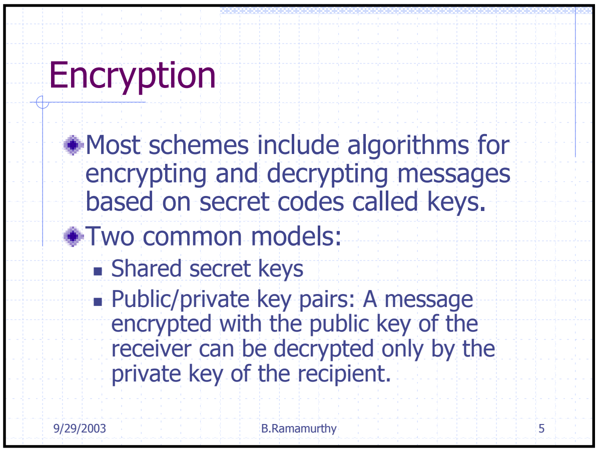
Cryptography

- 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.



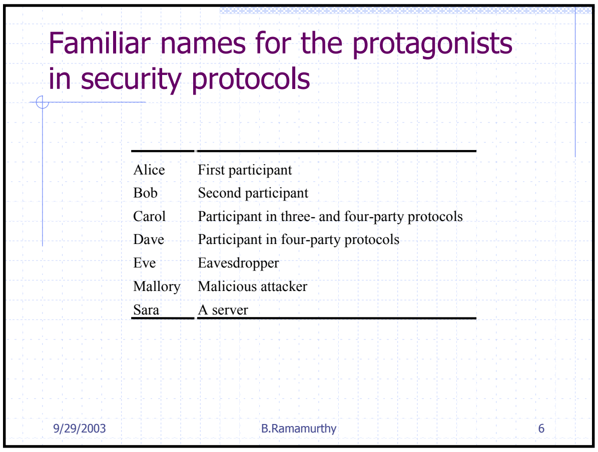
Historical context: the evolution of security needs

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



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

Alice	First participant
Bob	Second participant
Carol	Participant in three- and four-party protocols
Dave	Participant in four-party protocols
Eve	Eavesdropper
Mallory	Malicious attacker
Sara	A server

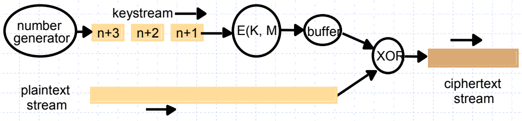
Cryptography notations

K_A	Alice's secret key
K_B	Bob's secret key
K_{AB}	Secret key shared between Alice and Bob
K_{Apriv}	Alice's private key (known only to Alice)
K_{Apub}	Alice's public key (published by Alice for all to read)
$\{M\}_K$	Message M encrypted with key K
$[M]_K$	Message M signed with key K

Cryptographic Algorithms

- Plain text → cipher text→ 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

```
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

```
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;
}
```

TEA in use

```
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:
   $P = 13$, $Q = 17$ –> $N = 221$, $Z = 192$
   $d = 5$
3. To find e solve the equation:
   $e \times d = 1 \bmod Z$

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

$e \times d = 1 \bmod 192 = 1, 193, 385, \ldots$
385 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 X Q = 13X17 = 221), e = 77, d = 5:

$E'(e,N,M) = M^e \bmod N$

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

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

$D'(d,N,c) = c^d \bmod N$

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 $K_e = \langle e,N \rangle$ and $K_d = \langle d,N \rangle$, and we get the encryption function:

$E(K_e, M) = \{M\}_K$ (the notation here indicating that the encrypted message can be decrypted only by the holder of the private key K_d) and $D(K_d, =\{M\}_K) = M$.

<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 → E(M,B)
- Only Buffalo can read it using it private key b: E(p, E(M,B)) → 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)) → M
  - E(P, E(M,p)) → 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)) → E(M,a)
  - E(A,E(M,a)) → 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.

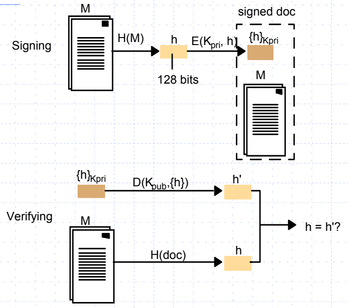
Digest Functions

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

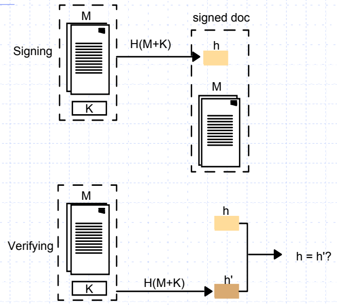
Alice's bank account certificate

1. *Certificate type*	Account number
2. *Name*	Alice
3. *Account*	6262626
4. *Certifying authority*	Bob's Bank
5. *Signature*	$\{Digest(field\ 2 + field\ 3)\}_{K_{Bpriv}}$

Digital signatures with public keys



Low-cost signatures with a shared secret key



X509 Certificate format

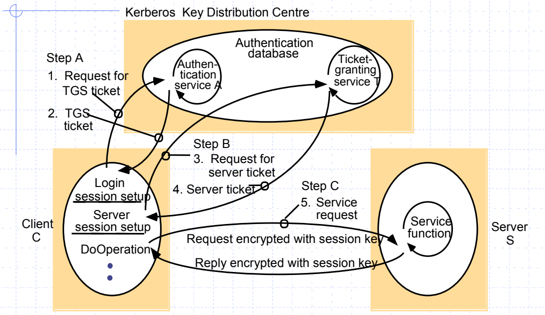
Subject	Distinguished Name, Public Key
Issuer	Distinguished Name, Signature
Period of validity	Not Before Date, Not After Date
Administrative information	Version, Serial Number
Extended Information	

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

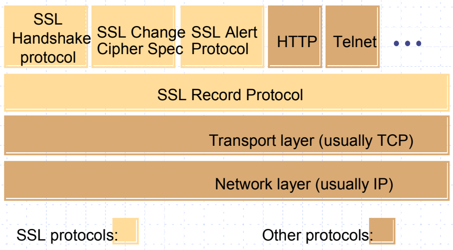
The Needham–Schroeder secret-key authentication protocol

Header	*Message*	*Notes*
1. A->S:	A, B, N_A	A requests S to supply a key for communication with B.
2. S->A:	$\{N_A, B, K_{AB}, \{K_{AB}, A\}_{K_B}\}_{K_A}$	S returns a message encrypted in A's secret key, containing a newly generated key K_{AB} and a 'ticket' encrypted in B's secret key. The nonce N_A 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.
3. A->B:	$\{K_{AB}, A\}_{K_B}$	A sends the 'ticket' to B.
4. B->A:	$\{N_B\}_{K_{AB}}$	B decrypts the ticket and uses the new key K_{AB} to encrypt another nonce N_B.
5. A->B:	$\{N_B - 1\}_{K_{AB}}$	A demonstrates to B that it was the sender of the previous message by returning an agreed transformation of N_B.

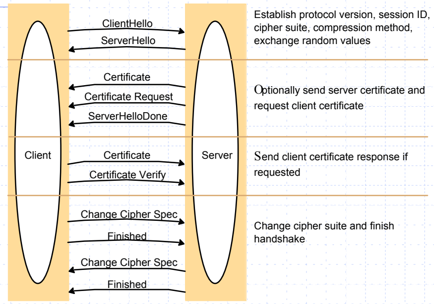
System architecture of Kerberos


Kerberos Key Distribution Centre
Authentication database
Authen-tication service A
Ticket-granting service T
Step A
1. Request for TGS ticket
2. TGS ticket
Step B
3. Request for server ticket
4. Server ticket
Step C
5. Service request
Login session setup
Server session setup
DoOperation
Client C
Request encrypted with session key
Reply encrypted with session key
Service function
Server S

SSL protocol stack



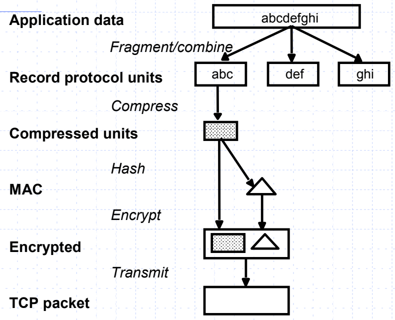
SSL handshake protocol


Client
Server
ClientHello
ServerHello
Certificate
Certificate Request
ServerHelloDone
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

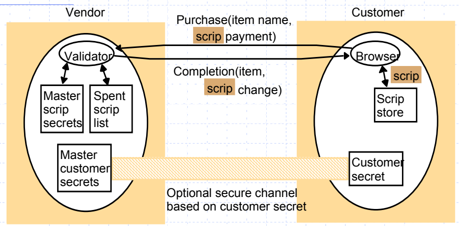
SSL handshake configuration options

Component	*Description*	*Example*
Key exchange method	the method to be used for exchange of a session key	RSA with public-key certificates
Cipher for data transfer	the block or stream cipher to be used for data	IDEA
Message digest function	for creating message authentication codes (MACs)	SHA

SSL record protocol



Millicent architecture



Scrip layout

Vendor	Value	Scrip ID	Customer ID	Expiry date	Properties	Certificate