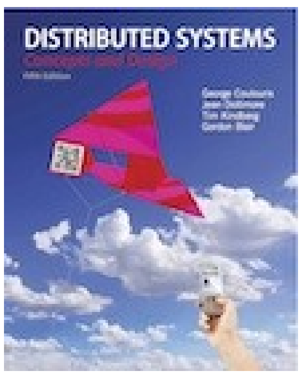
Slides for Chapter 11: Security



From **Coulouris, Dollimore, Kindberg and Blair**

Distributed Systems: Concepts and Design

Figure 11.1
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

Figure 11.2
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

Figure 11.3
Alice's bank account certificate

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

Figure 11.4
Public-key certificate for Bob's Bank

1.	*Certificate type*:	Public key
2.	*Name*:	Bob's Bank
3.	*Public key*:	K_{Bpub}
4.	*Certifying authority*:	Fred – The Bankers Federation
5.	*Signature*:	$\{Digest(field\ 2 + field\ 3)\}_{K_{Fpriv}}$

Figure 11.5
Cipher block chaining

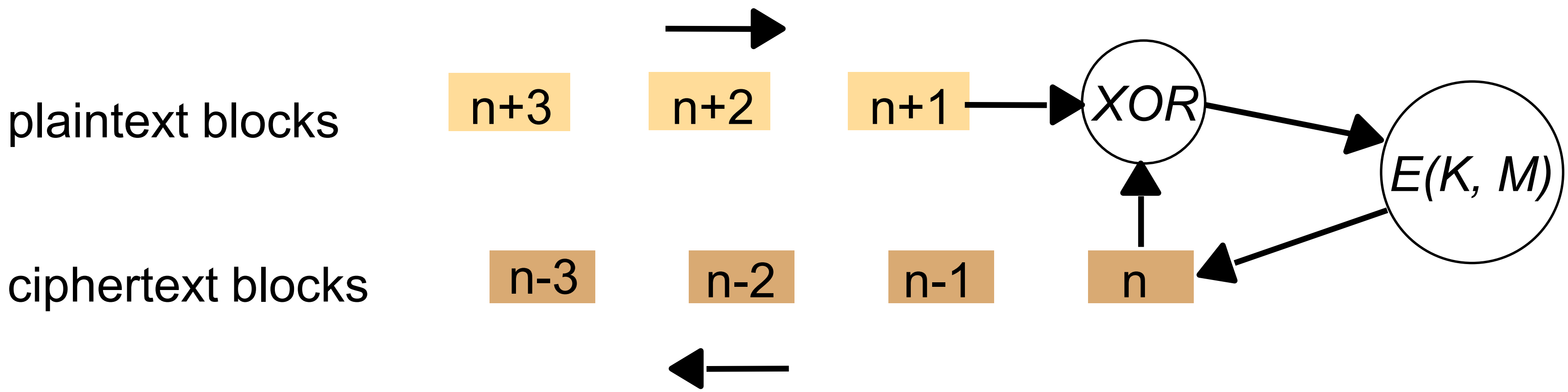


Figure 11.6
Stream cipher

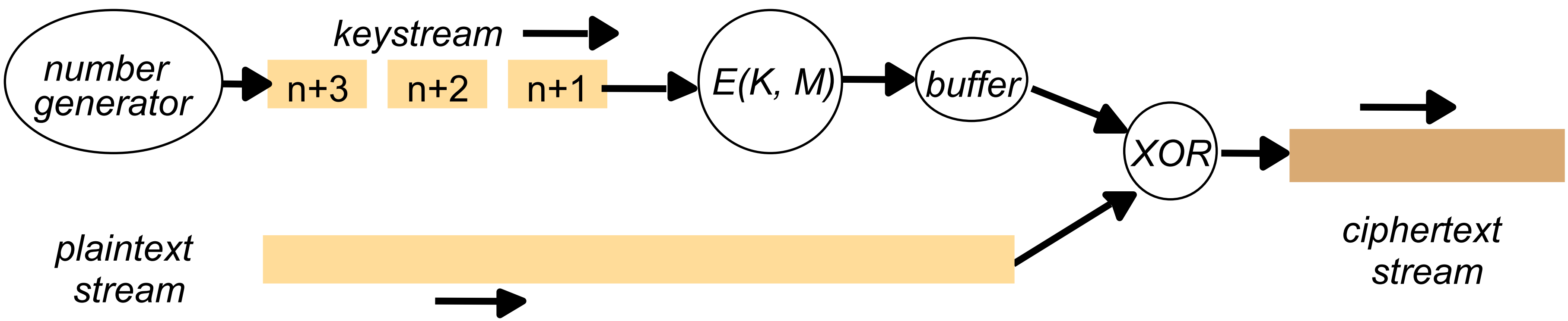


Figure 11.7
TEA encryption function

```
void encrypt(unsigned long k[], unsigned long text[]) {
    unsigned long y = text[0], z = text[1];                                1
    unsigned long delta = 0x9e3779b9, sum = 0; int n;                      2
    for (n= 0; n < 32; n++) {                                              3
        sum += delta;                                                      4
        y += ((z << 4) + k[0]) ^ (z+sum) ^ ((z >> 5) + k[1]);              5
        z += ((y << 4) + k[2]) ^ (y+sum) ^ ((y >> 5) + k[3]);              6
    }
    text[0] = y;  text[1] = z;                                             7
}
```

Figure 11.8
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;
}
```

Figure 11.9
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 - 1

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, 2*Z*+1, 3*Z*+1, ...

$e \times d = 1 \bmod 192 = 1, 193, 385, \ldots$

385 is divisible by *d*

e = 385/5 = 77

RSA Encryption - 2

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 27 = 128

The function for encrypting a single block of plaintext M is:

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

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:

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

Rivest, Shamir and Adelman prove*d* that E' and D' are mutual inverses (that is, $E'(D'(x)) = D'(E'(x)) = x$) for all values of P in the range $0 \leq P \leq N$.

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:

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

Figure 11.10
Digital signatures with public keys

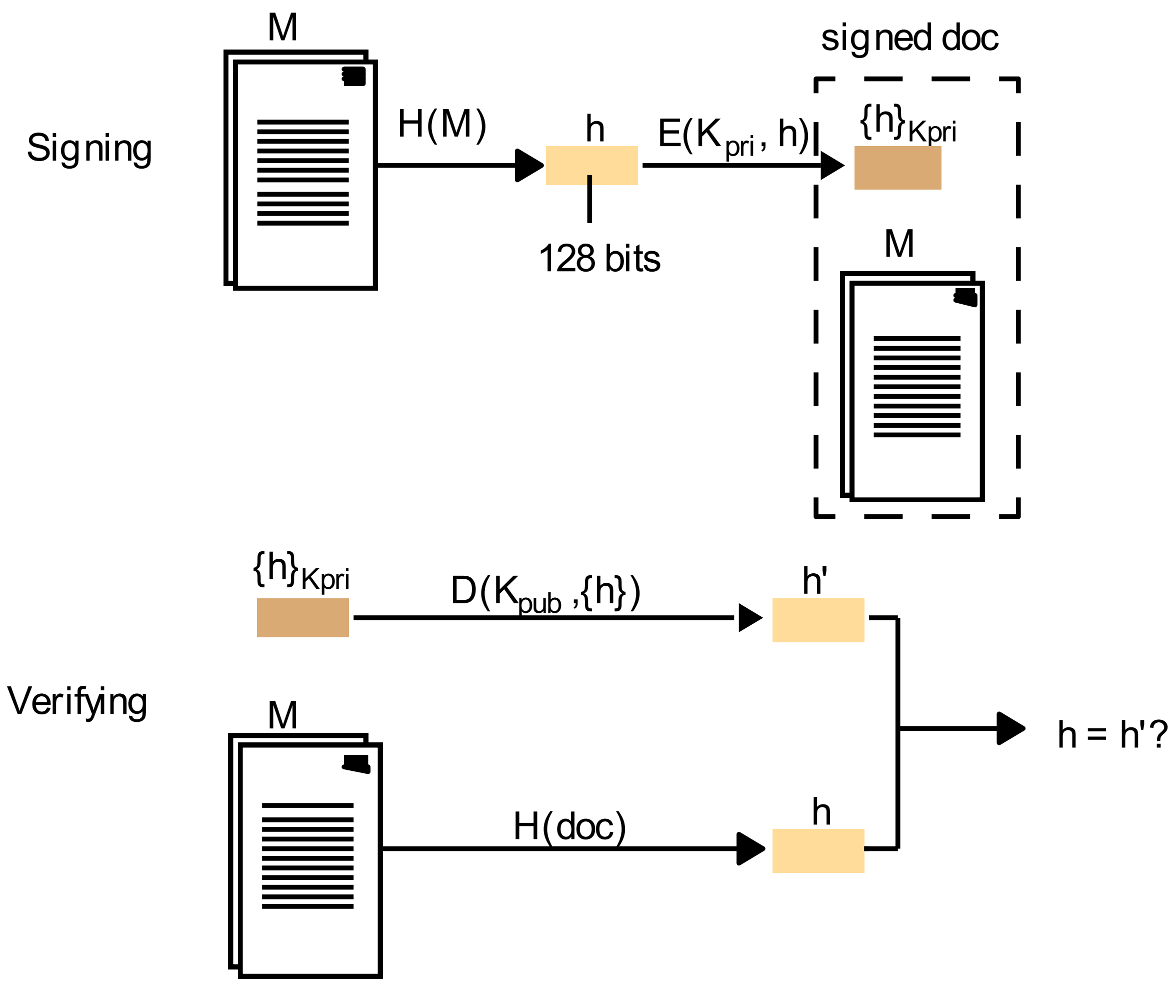


Figure 11.11
Low-cost signatures with a shared secret key

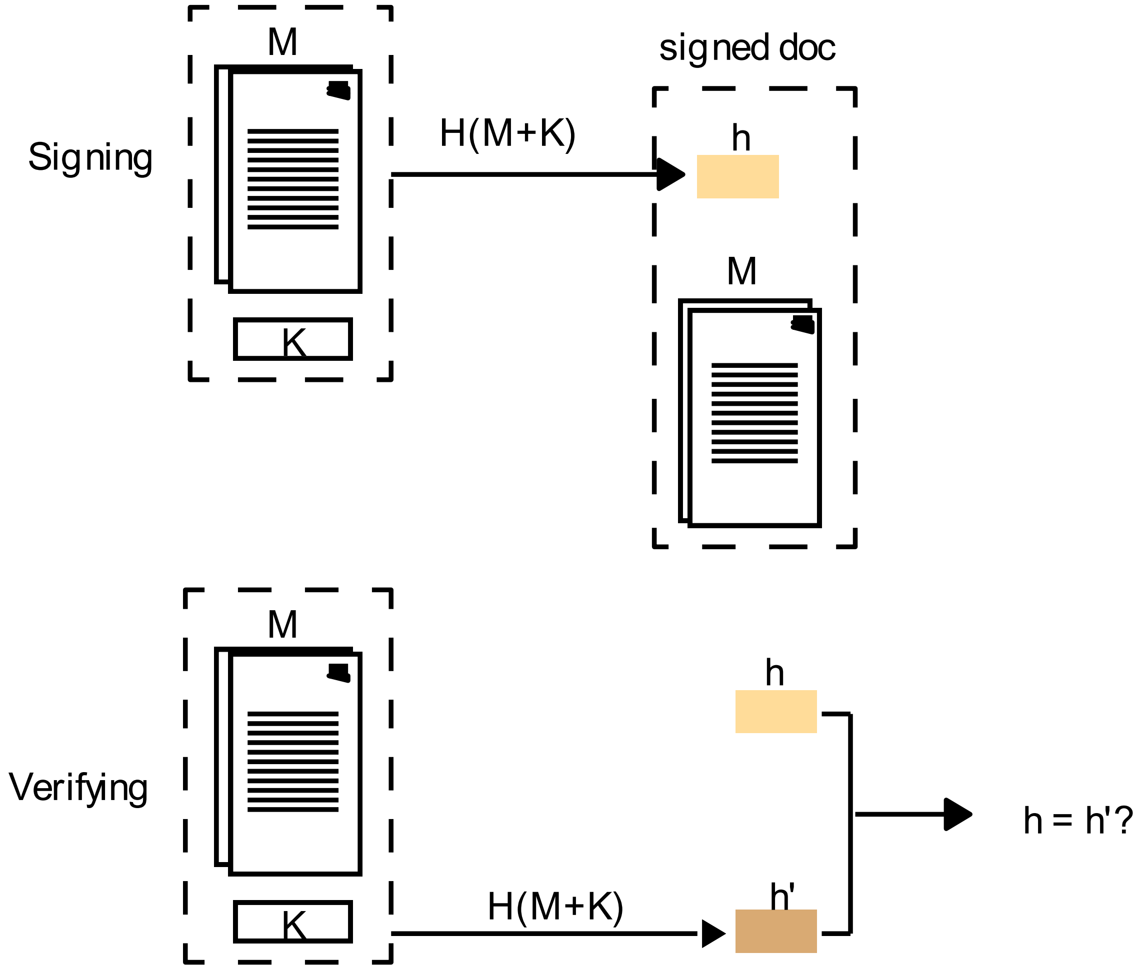


Figure 11.12
X509 Certificate format

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	

Figure 11.13
Performance of symmetric encryption and secure digest algorithms

	Key size/hash size (bits)	*PRB optimized 90 MHz Pentium 1 (Mbytes/s)*	*Crypto++ 2.1 GHz Pentium 4 (Mbytes/s)*
TEA	128	–	23.801
DES	56	2.113	21.340
Triple-DES	112	0.775	9.848
IDEA	128	1.219	18.963
AES	128	–	61.010
AES	192	–	53.145
AES	256	–	48.229
MD5	128	17.025	216.674
SHA-1	160	–	67.977

Figure 11.14
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.

Figure 11.15
System architecture of Kerberos

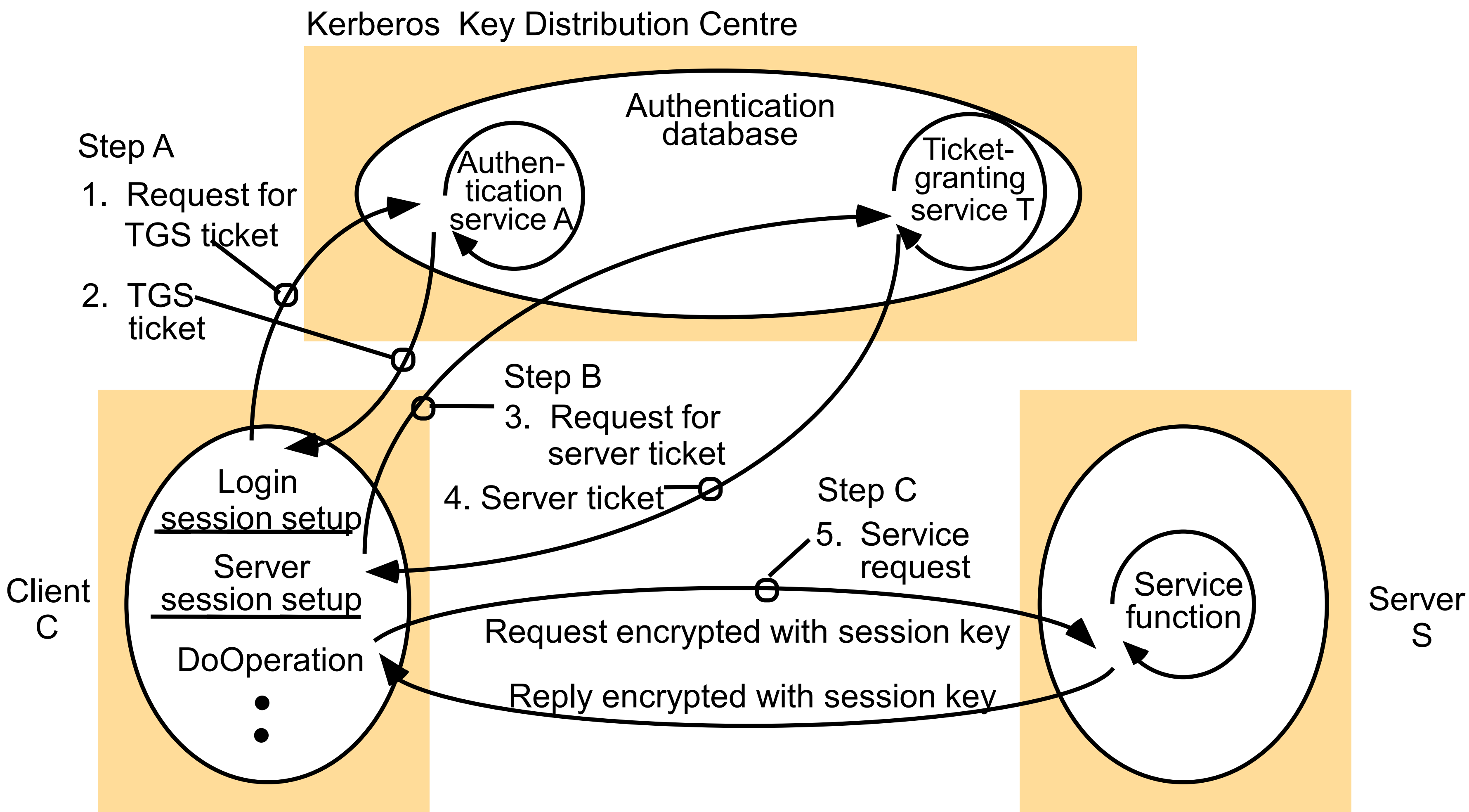


Figure 11.16
SSL protocol stack

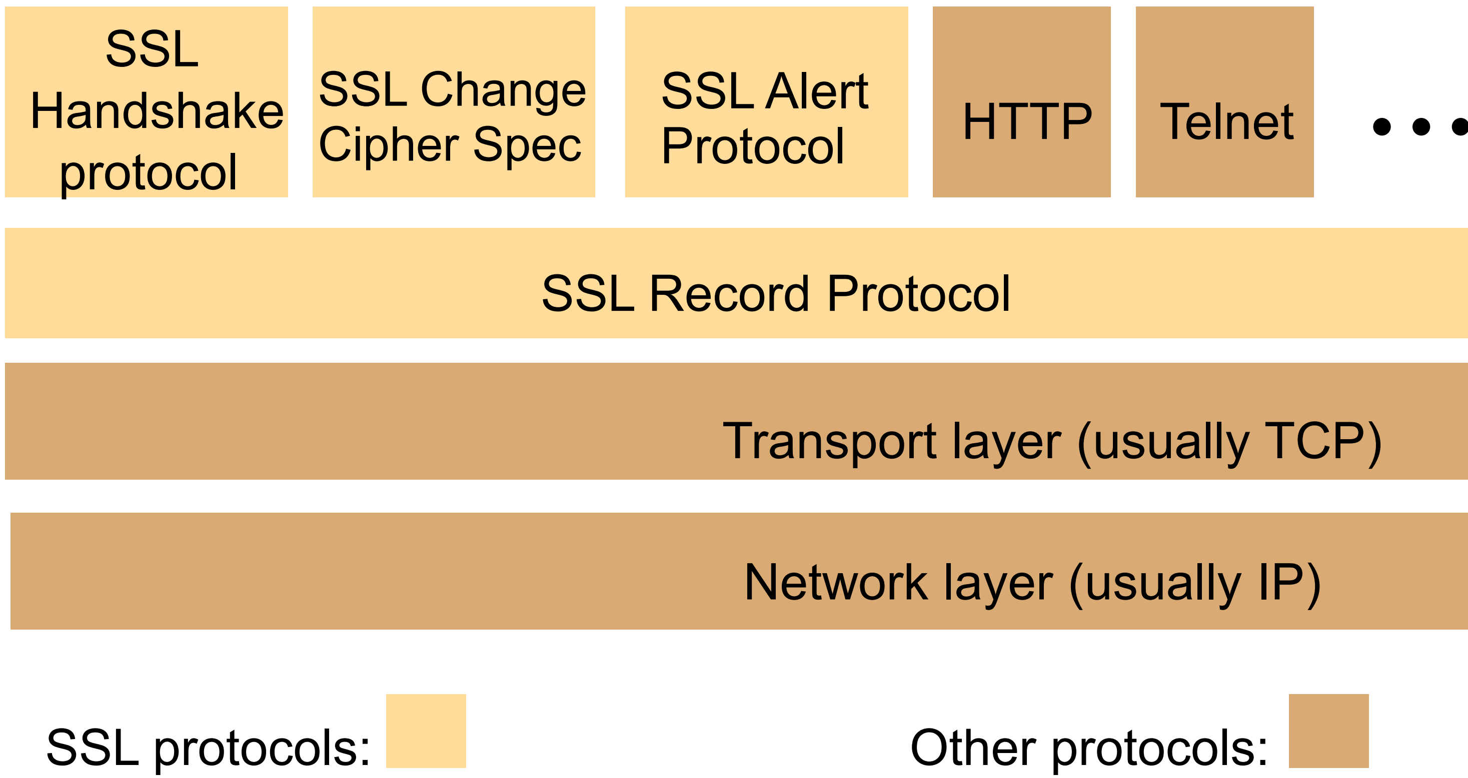


Figure 11.17
TLS handshake protocol

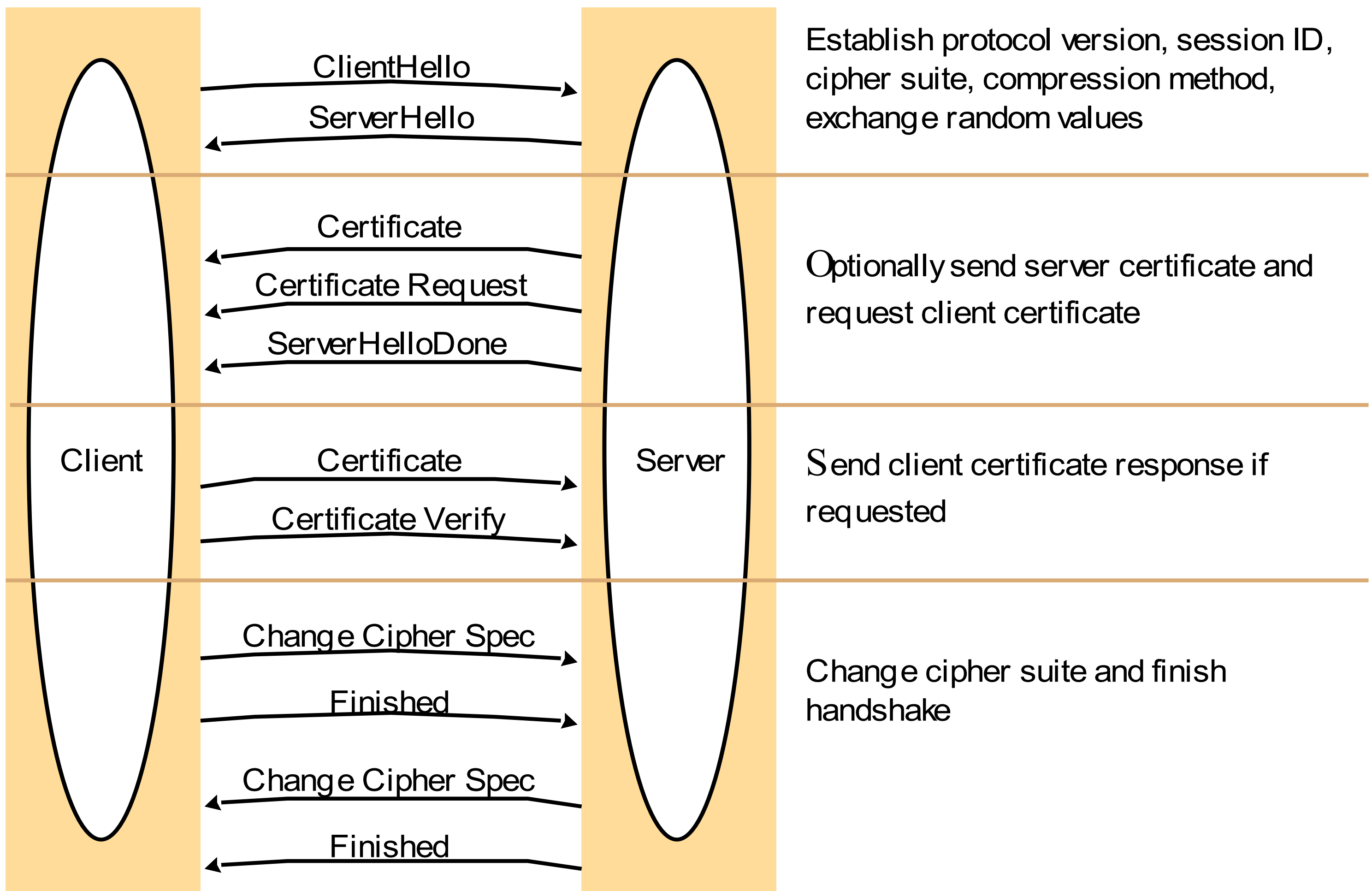


Figure 11.18
TLS handshake configuration options

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-1

Figure 11.19
TLS record protocol

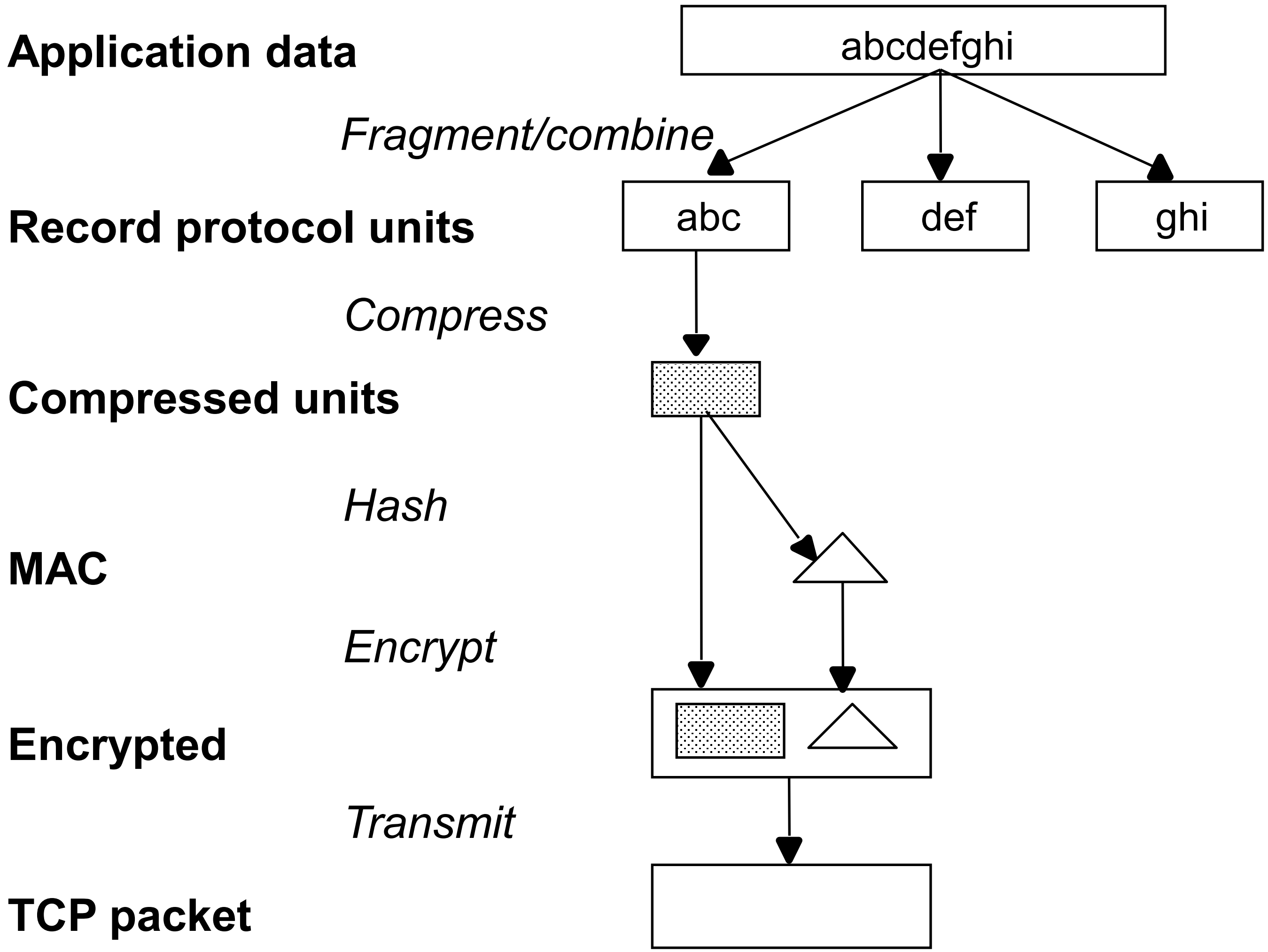


Figure 11.20
Use of RC4 stream cipher in IEEE 802.11 WEP