Figure 7.1 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

Figure 7.2 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 7.3 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 7.4 Alice's bank account certficate

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 7.5 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 7.6 Cipher block chaining

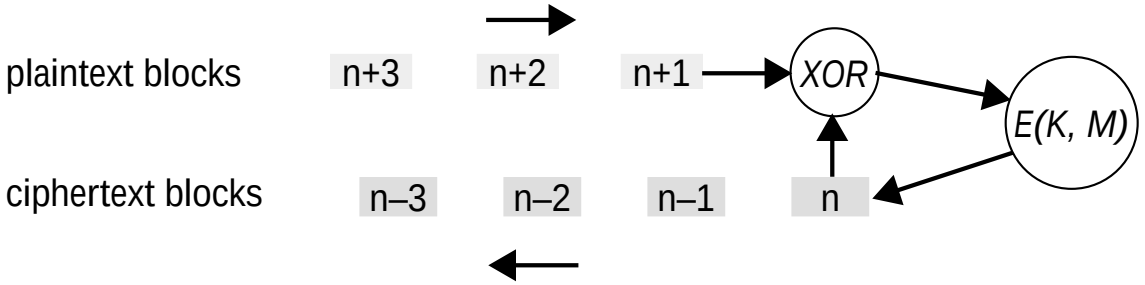


Figure 7.7 Stream cipher

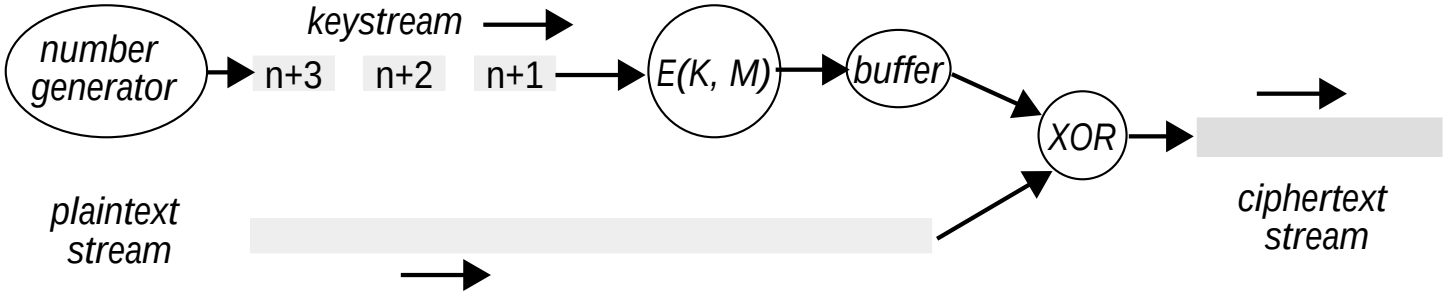


Figure 7.8 TEA encryption function

```
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 7.9 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 7.10 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 10^{100}), 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:
   $e \times d = 1 \ mod \ Z$

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

$e \times d = 1 \ mod \ 192 \quad = 1, 193, 385, ...$
385 is divisible by d
$e = 385/5 = 77$

RSA Encryption (cont'd)

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:

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

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 proved 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 functions $E(K_e, M) = \{M\}_K$ (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 7.11 Digital signatures with public keys

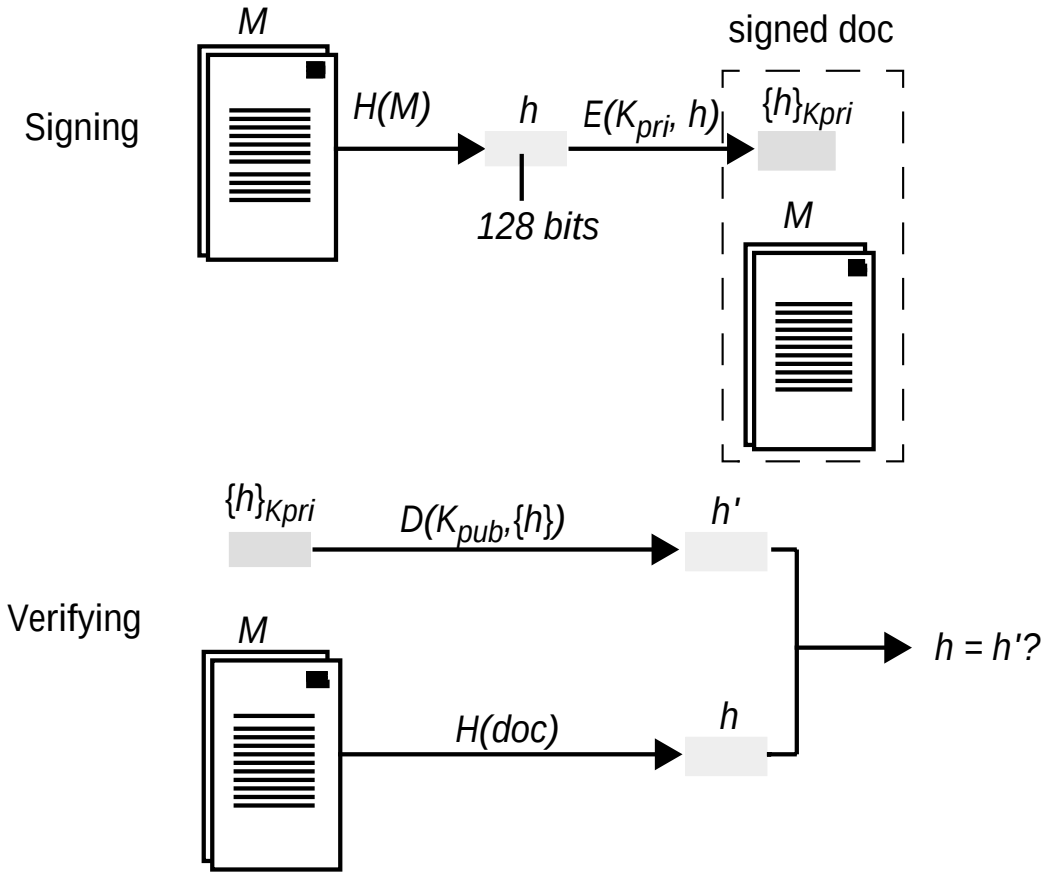


Figure 7.12 Low-cost signatures with a shared secret key

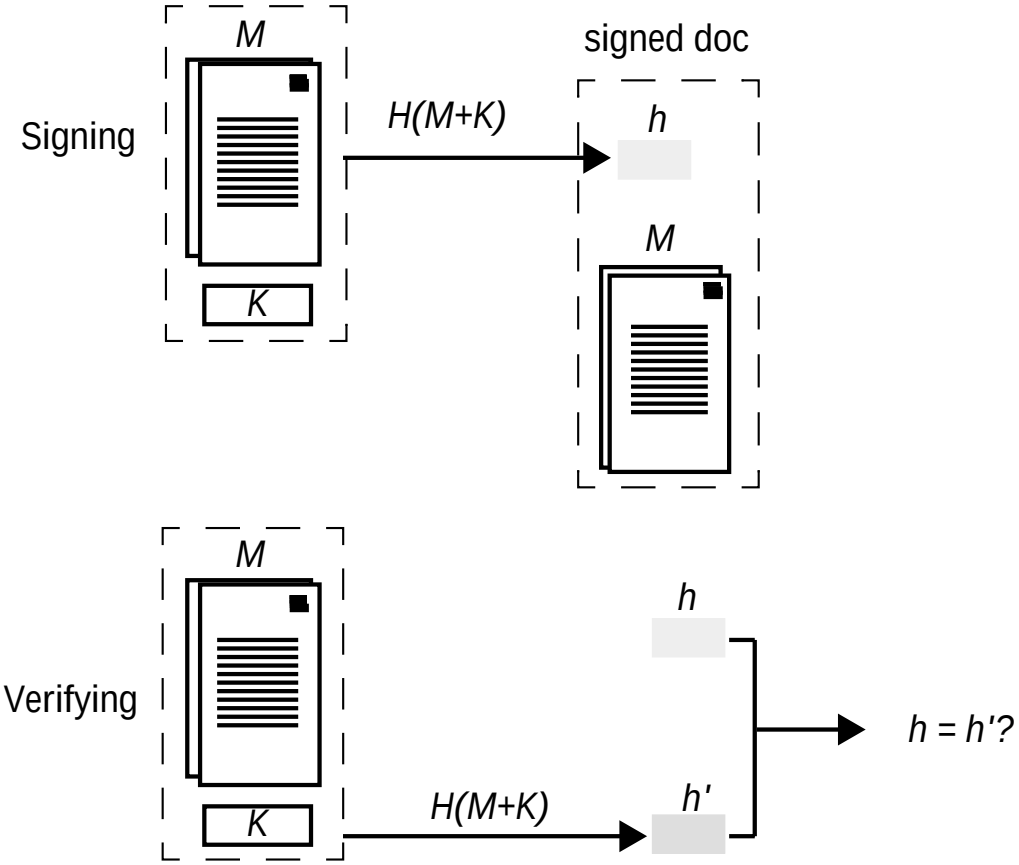


Figure 7.13 X509 Certificate format

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	

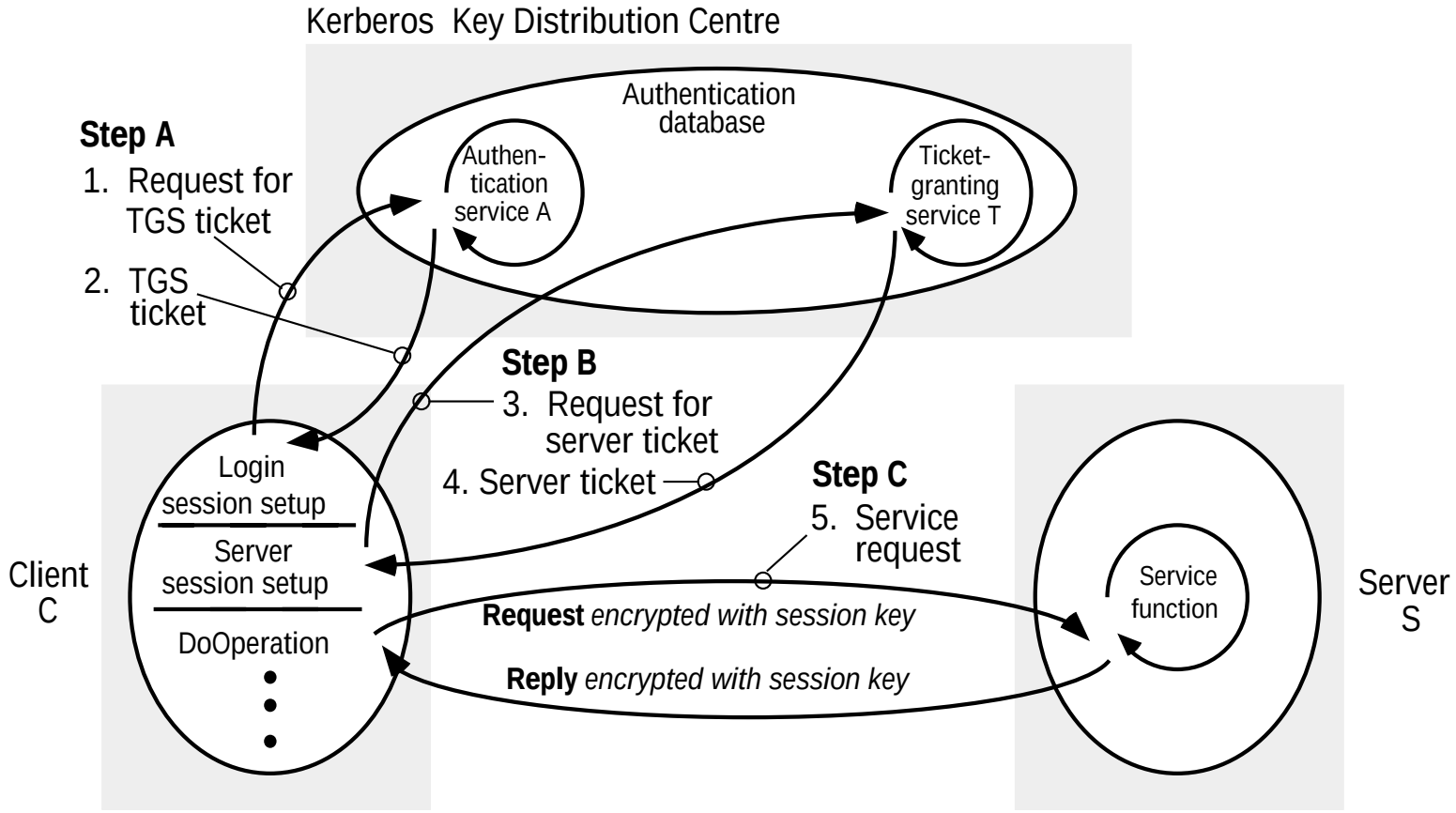
Figure 7.14 Performance of encryption and secure digest algorithms

	Key size/hash size (bits)	*Extrapolated speed (kbytes/sec.)*	*PRB optimized (kbytes/s)*
TEA	128	700	-
DES	56	350	7746
Triple-DES	112	120	2842
IDEA	128	700	4469
RSA	512	7	-
RSA	2048	1	-
MD5	128	1740	62425
SHA	160	750	25162

Figure 7.15 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 7.16 System architecture of Kerberos



Kerberos Protocol

A. Obtain Kerberos session key and TGS ticket, once per login session

Header	*Message*	*Notes*
1. C → A: Request for TGS ticket	C, T, n	Client C requests the Kerberos authentication server A to supply a ticket for communication with the ticket-granting service T.
2. A → C: TGS session key and ticket	$\{K_{CT}, n\}_{K_C}$, $\{ticket(C,T)\}_{K_T}$ containing C, T, t_1, t_2, K_{CT}	A returns a message containing a ticket encrypted in its secret key and a session key for C to use with T. The inclusion of the nonce n encrypted in K_C shows that the message comes from the recipient of message 1, who must know K_C.

B. Obtain ticket for a server S, once per client-server session

3. C → T: Request ticket for service S	$\{auth(C)\}_{K_{CT}}$, $\{ticket(C,T)\}_{K_T}$, S, n	C requests the ticket-granting server T to supply a ticket for communication with another server S.
4. T → C: Service ticket	$\{K_{CS}, n\}_{K_{CT}}$, $\{ticket(C,S)\}_{K_S}$	T checks the ticket. If it is valid T generates a new random session key K_{CS} and returns it with a ticket for S (encrypted in the server's secret key K_S).

Kerberos Protocol (cont'd)

C.	*Issue a server request with a ticket*	
5. C → S: Service request	$\{auth(C)\}_{K_{CS}}$, $\{ticket(C,S)\}_{K_S}$, *request, n*	C sends the ticket to S with a newly generated authenticator for C and a request. The request would be encrypted in K_{CS} if secrecy of the data is required.
D.	*Authenticate server (optional)*	
6. S → C: Server auth-entication	$\{n\}_{K_{CS}}$	(Optional): S sends the nonce to C, encrypted in K_{CS}.

Figure 7.17 SSL protocol stack

(The figures in this section are based on diagrams in Hirsch [1997] and are published with Frederick Hirsch's permission)

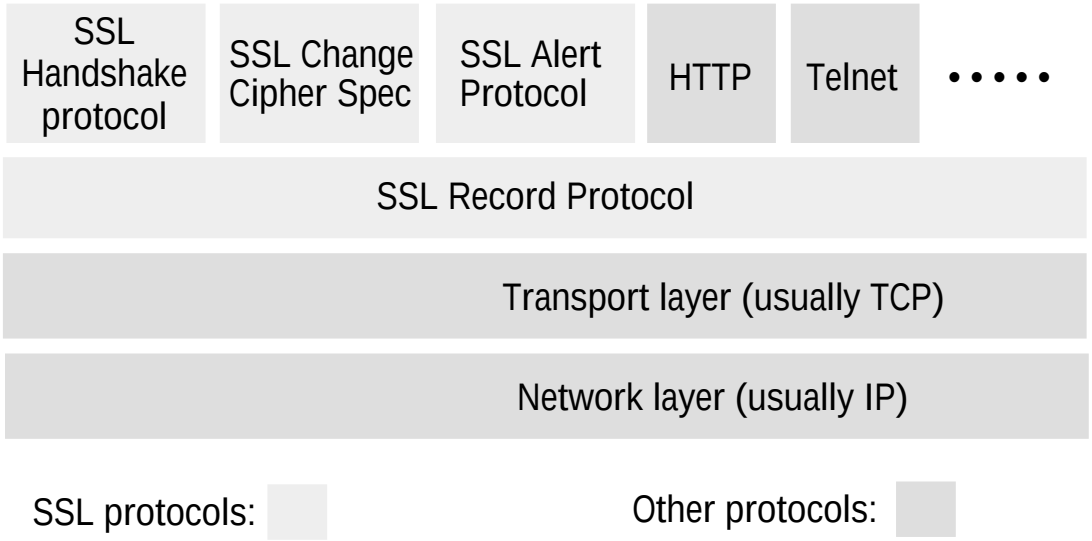


Figure 7.18 SSL handshake protocol

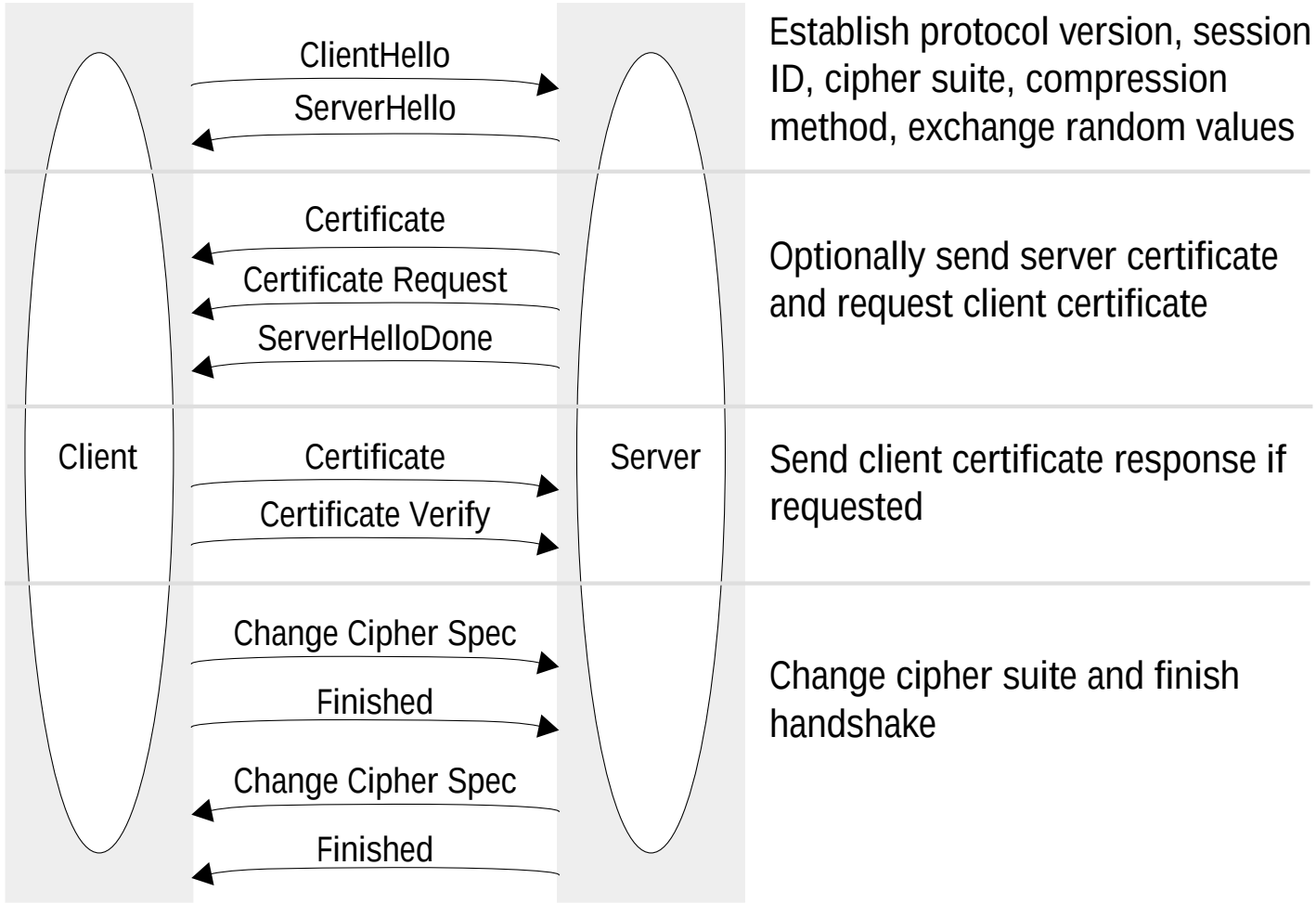
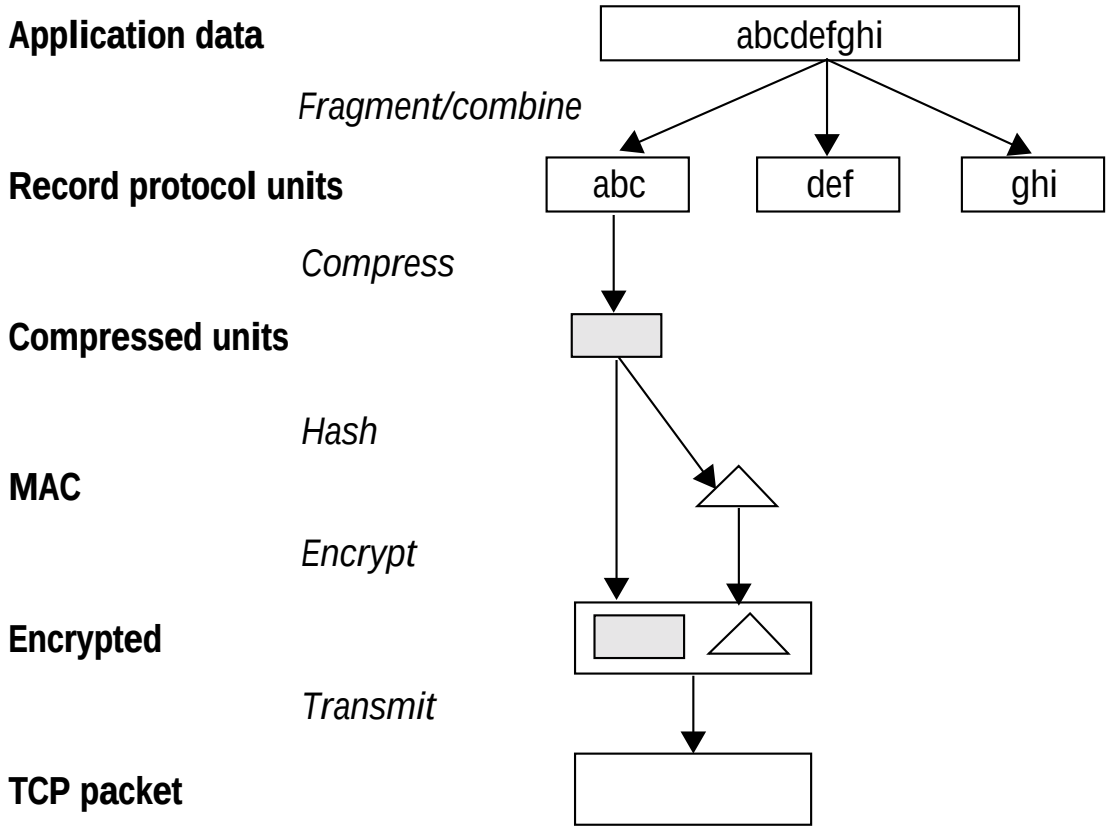


Figure 7.19 SSL handshake configuration options

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

Figure 7.20 SSL record protocol



Scrip layout

Vendor	Value	Scrip ID	Customer ID	Expiry date	Properties	Certificate

Figure 7.21 Millicent architecture