AES, RC4, Location of Encryption Devices, and Symmetric Key Distribution

Dr. Demetrios Glinos
University of Central Florida

CIS3360 - Security in Computing

Readings

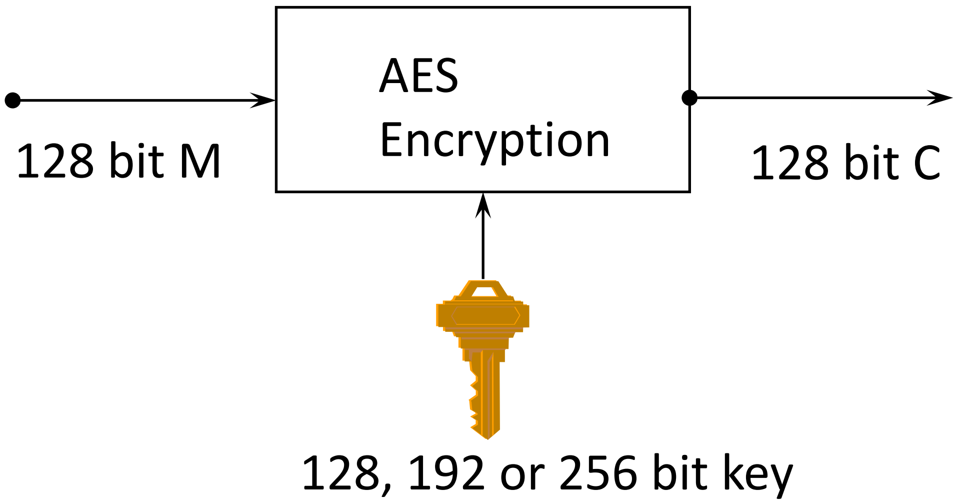
- "Computer Security: Principles and Practice", 3rd Edition, by William Stallings and Lawrie Brown
  - Sec. 20.3, 20.4, 20.6, 20.7

Outline

- The Advanced Encryption Standard (AES)
- The RC4 Stream Cipher
- Location of Symmetric Encryption Devices
- Symmetric Key Distribution

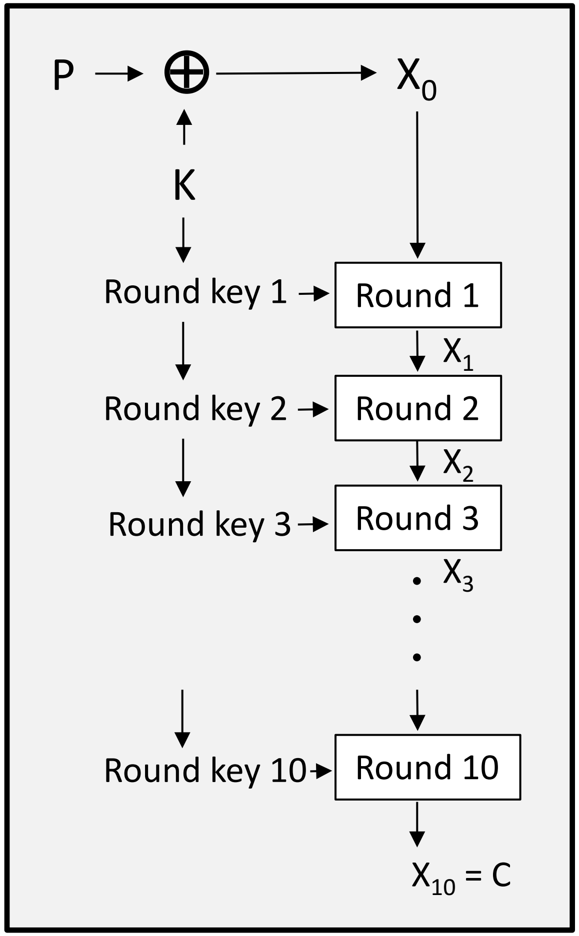
AES Background

- **Advanced Encryption Standard (AES)**
  - Published in 2001, standardized in 2002.
  - AES is based on **Rijndael** cipher structure (not Feistel)
    - ***Rijndael structure uses advanced mathematics (group, ring, and field theory) which we will not cover***
  - **Key size:** 128, 192 or 256 bits
  - **Block size:** 128 bits



AES Round Structure

- The 128-bit version of AES uses 10 rounds to encrypt each block of the input plaintext
- Each round performs an invertible transformation on a 128-bit array, arranged as a **4-byte by 4-byte square array** called the **state**.
- The initial state X_0 is the **XOR** of the plaintext **P** with the key **K**: $X_0 = P \oplus K$.
- Round i (i = 1, …, 10) receives state X_{i-1} as input and produces state X_i.
- The ciphertext C (for the block) is the output of the final round: $C = X_{10}$.

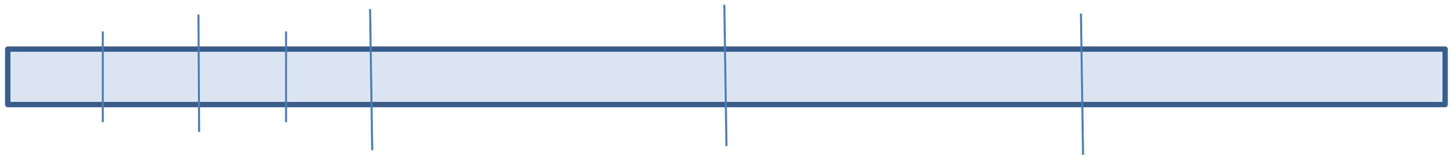


AES Round Processing

- *In each round*, the state undergoes:
  - **SubBytes step:**
    - byte *substitution*: same S-box used on ***every*** element of the state (8 bits each)
  - **ShiftRows step:**
    - shift rows: *permutation* of the bytes in each row
  - **MixColumns step:**
    - *mix values in each column* using matrix multiplication
    - basically, applies a Hill Cipher to each column
  - **AddRoundkey step:**
    - XOR the state with the *round key* derived from the 128-bit encryption key

State Representation of 128-bit Block

128 bits = 16 bytes of 8 bits each – interpreted in column major order



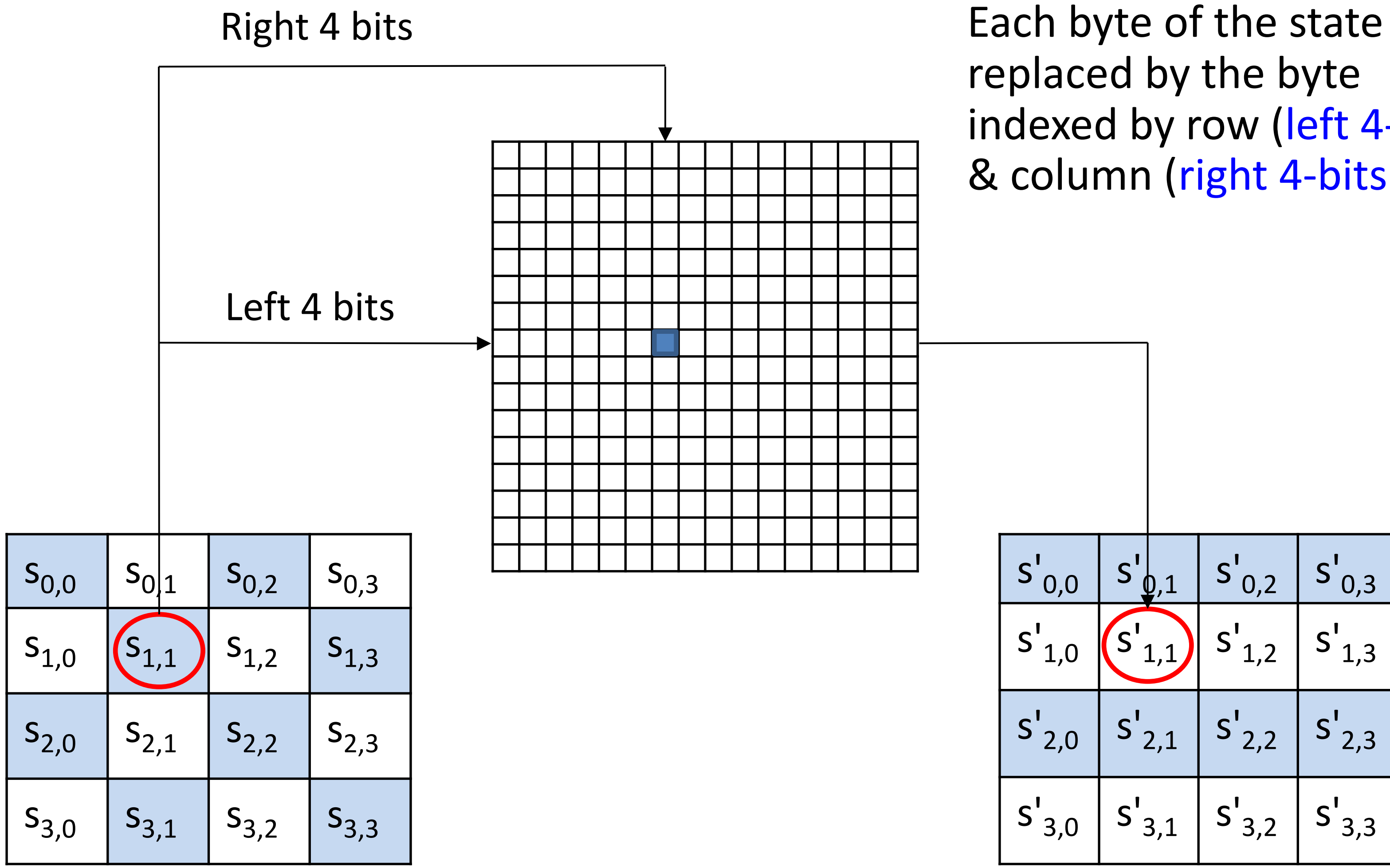
$(a_{0,0} | a_{1,0} | a_{2,0} | a_{3,0} | a_{0,1} | a_{1,1} | a_{2,1} | a_{3,1} | a_{0,2} | a_{1,2} | a_{2,2} | a_{3,2} | a_{0,3} | a_{1,3} | a_{2,3} | a_{3,3})$

$a_{0,0}$	$a_{0,1}$	$a_{0,2}$	$a_{0,3}$
$a_{1,0}$	$a_{1,1}$	$a_{1,2}$	$a_{1,3}$
$a_{2,0}$	$a_{2,1}$	$a_{2,2}$	$a_{2,3}$
$a_{3,0}$	$a_{3,1}$	$a_{3,2}$	$a_{3,3}$

This array is called the **State**

SubBytes Step: Byte Substitution

Each byte of the state is replaced by the byte indexed by row (left 4-bits) & column (right 4-bits)



S-Box (16 x 16)

Example: Byte {95} is replaced by byte in row 9 column 5, which has value {2A}

		y															
		0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
x	0	63	7C	77	7B	F2	6B	6F	C5	30	01	67	2B	FE	D7	AB	76
	1	CA	82	C9	7D	FA	59	47	F0	AD	D4	A2	AF	9C	A4	72	C0
	2	B7	FD	93	26	36	3F	F7	CC	34	A5	E5	F1	71	D8	31	15
	3	04	C7	23	C3	18	96	05	9A	07	12	80	E2	EB	27	B2	75
	4	09	83	2C	1A	1B	6E	5A	A0	52	3B	D6	B3	29	E3	2F	84
	5	53	D1	00	ED	20	FC	B1	5B	6A	CB	BE	39	4A	4C	58	CF
	6	D0	EF	AA	FB	43	4D	33	85	45	F9	02	7F	50	3C	9F	A8
	7	51	A3	40	8F	92	9D	38	F5	BC	B6	DA	21	10	FF	F3	D2
	8	CD	0C	13	EC	5F	97	44	17	C4	A7	7E	3D	64	5D	19	73
	9	60	81	4F	DC	22	2A	90	88	46	EE	B8	14	DE	5E	0B	DB
	A	E0	32	3A	0A	49	06	24	5C	C2	D3	AC	62	91	95	E4	79
	B	E7	C8	37	6D	8D	D5	4E	A9	6C	56	F4	EA	65	7A	AE	08
	C	BA	78	25	2E	1C	A6	B4	C6	E8	DD	74	1F	4B	BD	8B	8A
	D	70	3E	B5	66	48	03	F6	0E	61	35	57	B9	86	C1	1D	9E
	E	E1	F8	98	11	69	D9	8E	94	9B	1E	87	E9	CE	55	28	DF
	F	8C	A1	89	0D	BF	E6	42	68	41	99	2D	0F	B0	54	BB	16

source: Table 20.2(a)

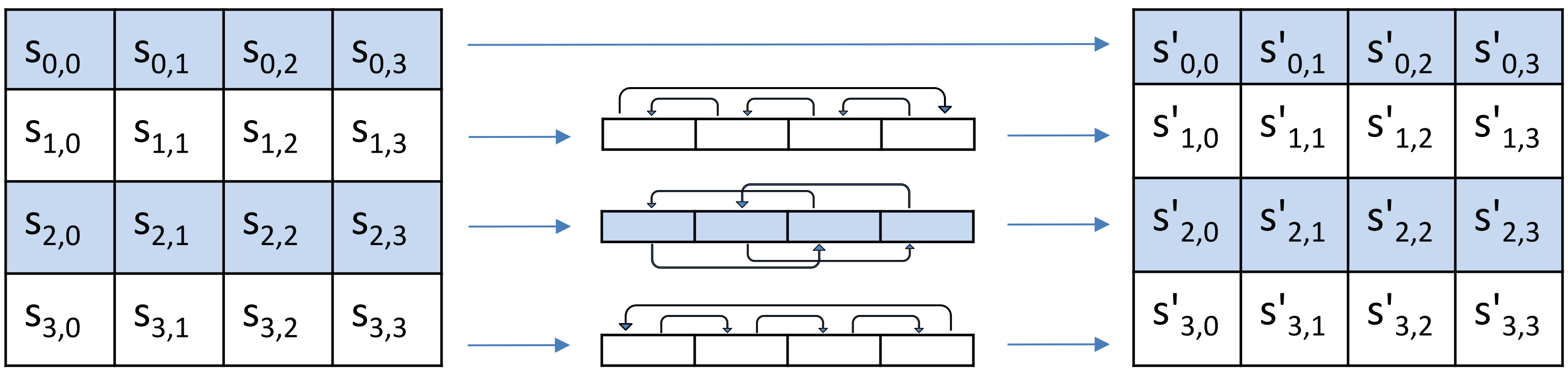
Inverse S-Box

NOTE: This table is used for decryption

		y															
		0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
x	0	52	09	6A	D5	30	36	A5	38	BF	40	A3	9E	81	F3	D7	FB
	1	7C	E3	39	82	9B	2F	FF	87	34	8E	43	44	C4	DE	E9	CB
	2	54	7B	94	32	A6	C2	23	3D	EE	4C	95	0B	42	FA	C3	4E
	3	08	2E	A1	66	28	D9	24	B2	76	5B	A2	49	6D	8B	D1	25
	4	72	F8	F6	64	86	68	98	16	D4	A4	5C	CC	5D	65	B6	92
	5	6C	70	48	50	FD	ED	B9	DA	5E	15	46	57	A7	8D	9D	84
	6	90	D8	AB	00	8C	BC	D3	0A	F7	E4	58	05	B8	B3	45	06
	7	D0	2C	1E	8F	CA	3F	0F	02	C1	AF	BD	03	01	13	8A	6B
	8	3A	91	11	41	4F	67	DC	EA	97	F2	CF	CE	F0	B4	E6	73
	9	96	AC	74	22	E7	AD	35	85	E2	F9	37	E8	1C	75	DF	6E
	A	47	F1	1A	71	1D	29	C5	89	6F	B7	62	0E	AA	18	BE	1B
	B	FC	56	3E	4B	C6	D2	79	20	9A	DB	C0	FE	78	CD	5A	F4
	C	1F	DD	A8	33	88	07	C7	31	B1	12	10	59	27	80	EC	5F
	D	60	51	7F	A9	19	B5	4A	0D	2D	E5	7A	9F	93	C9	9C	EF
	E	A0	E0	3B	4D	AE	2A	F5	B0	C8	EB	BB	3C	83	53	99	61
	F	17	2B	04	7E	BA	77	D6	26	E1	69	14	63	55	21	0C	7D

source: Table 20.2(b)

ShiftRows Step



1st row is unchanged
2nd row does 1 byte circular shift to left
3rd row does 2 byte circular shift to left
4th row does 3 byte circular shift to left

<u>NOTE</u>: same set of shifts every time

MixColumns Step

- **Each column is processed separately**
- Each byte is replaced by a value dependent on all 4 bytes in the column
- Mix columns matrix is part of AES, just like the S-box and Inverse S-box

Mix Columns Matrix *State* *New State*

$$\begin{bmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{bmatrix} \begin{bmatrix} s_{0,0} & s_{0,1} & s_{0,2} & s_{0,3} \\ s_{1,0} & s_{1,1} & s_{1,2} & s_{1,3} \\ s_{2,0} & s_{2,1} & s_{2,2} & s_{2,3} \\ s_{3,0} & s_{3,1} & s_{3,2} & s_{3,3} \end{bmatrix} = \begin{bmatrix} s'_{0,0} & s'_{0,1} & s'_{0,2} & s'_{0,3} \\ s'_{1,0} & s'_{1,1} & s'_{1,2} & s'_{1,3} \\ s'_{2,0} & s'_{2,1} & s'_{2,2} & s'_{2,3} \\ s'_{3,0} & s'_{3,1} & s'_{3,2} & s'_{3,3} \end{bmatrix}$$

Example: $s'_{1,2} = 1*s_{0,2} + 2*s_{1,2} + 3*s_{2,2} + 1*s_{3,2}$

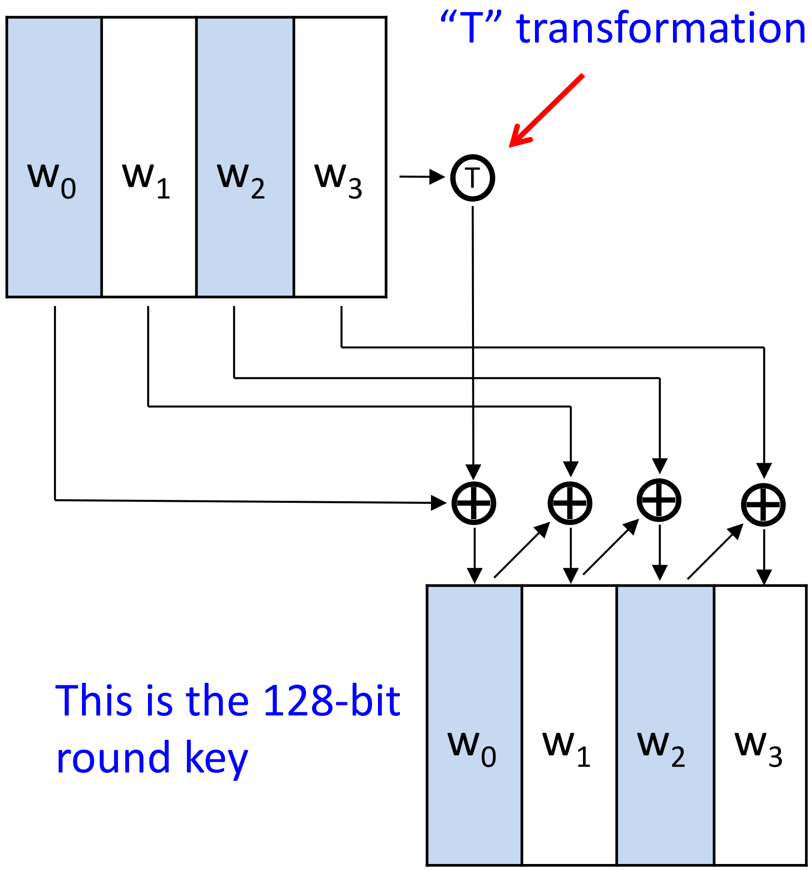
AddRoundKey Step

$$\begin{bmatrix} s_{0,0} & s_{0,1} & s_{0,2} & s_{0,3} \\ s_{1,0} & s_{1,1} & s_{1,2} & s_{1,3} \\ s_{2,0} & s_{2,1} & s_{2,2} & s_{2,3} \\ s_{3,0} & s_{3,1} & s_{3,2} & s_{3,3} \end{bmatrix} \oplus \begin{bmatrix} w_i & w_{i+1} & w_{i+2} & w_{i+3} \end{bmatrix} = \begin{bmatrix} s'_{0,0} & s'_{0,1} & s'_{0,2} & s'_{0,3} \\ s'_{1,0} & s'_{1,1} & s'_{1,2} & s'_{1,3} \\ s'_{2,0} & s'_{2,1} & s'_{2,2} & s'_{2,3} \\ s'_{3,0} & s'_{3,1} & s'_{3,2} & s'_{3,3} \end{bmatrix}$$

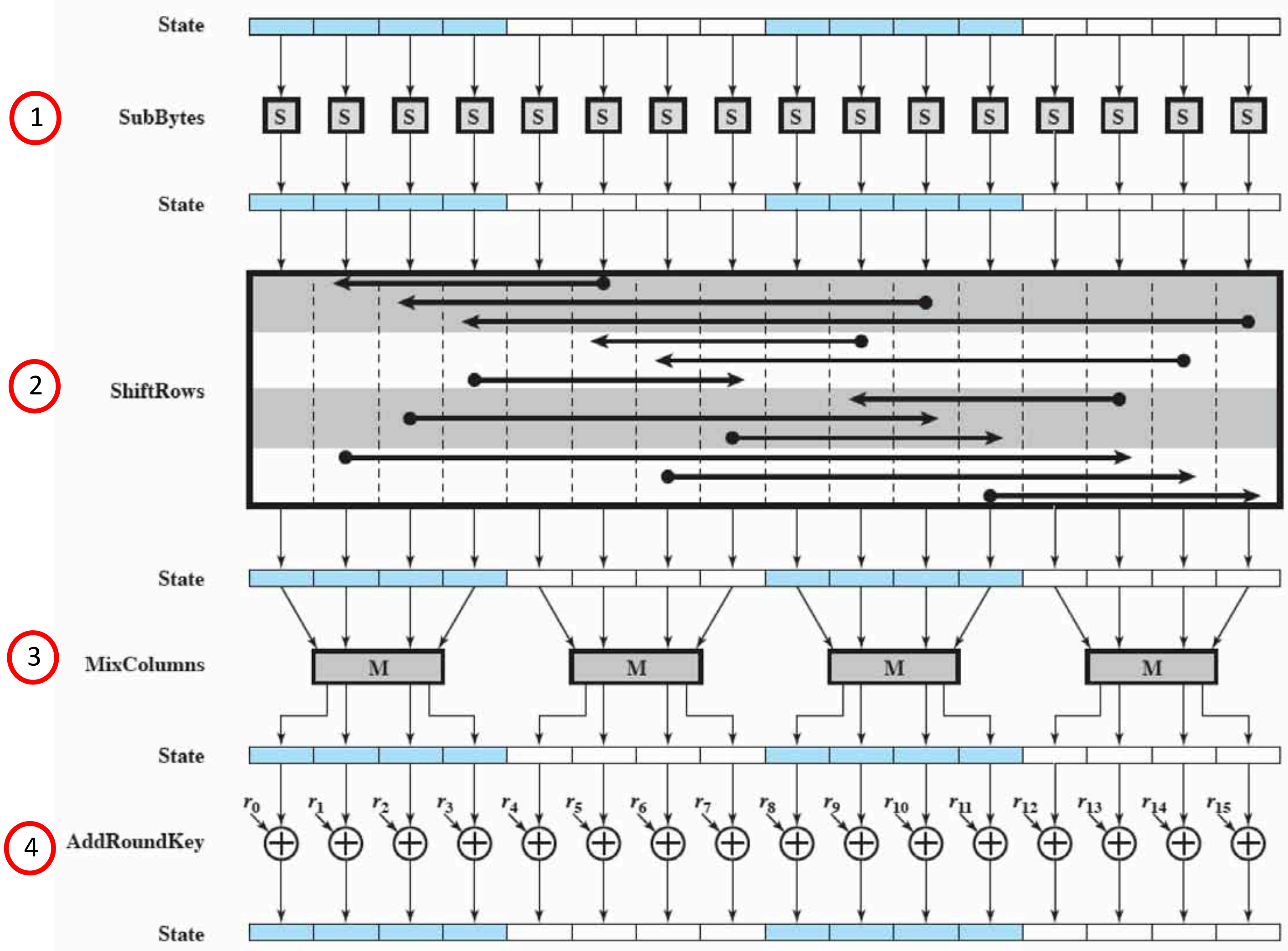
Exclusive-or (XOR) the state with a ***set*** of keys for the round, derived from the 128-bit secret key

AES Key Schedule (Expansion)

- ***The 128-bit key is first divided into four 4-byte "words" (represented as columns in diagram)***
- ***First column of round key is*** computed as XOR of previous round first column and the "T" transformation of the previous round last column.
- "T" transformation involves
  - Cyclical left shifts
  - S-box substitutions
  - XOR first byte with a "round" constant
- ***Remaining columns*** computed in order using XOR of previous round column value and the preceding column in the current round.



AES Encryption Round



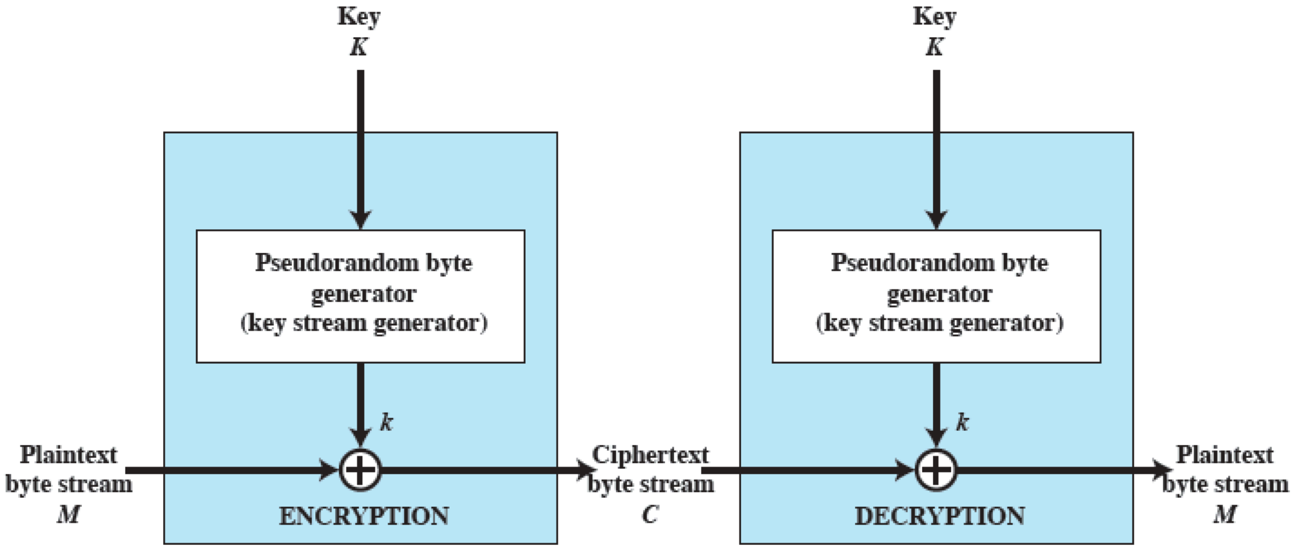
source: Fig. 20.4

Outline

- The Advanced Encryption Standard (AES)
- The RC4 Stream Cipher
- Location of Symmetric Encryption Devices
- Symmetric Key Distribution

Stream Cipher Structure

- A **stream cipher** processes one element at a time
  - element can be a bit, byte, or something larger
- Principal difference from a block cipher:
  - a stream cipher applies a *different key* for each element
  - cipher key is used by a **key stream generator** to generate the element keys
- Generic steam cipher structure:



source: Fig. 2.2(b)

Stream Cipher Operation

- Regardless of size of element, bit-wise XOR is used for both encryption and decryption
  - we have seen this previously as the binary one-time pad

- Example (given element is an 8-bit byte):

  - Encryption

	1 1 0 0 1 1 0 0	plaintext
$\oplus$	0 1 1 0 1 1 0 0	key stream
	1 0 1 0 0 0 0 0	ciphertext

  - Decryption

	1 0 1 0 0 0 0 0	ciphertext
$\oplus$	0 1 1 0 1 1 0 0	key stream
	1 1 0 0 1 1 0 0	plaintext

The RC4 Stream Cipher

- invented in 1987 by Ron Rivest (of RSA fame)
  - originally a trade secret of RSA Security, but posted the Internet in 1994
- uses a variable key size
- basic idea:
  - use a random permutation of the numbers 0 to 255 to generate the pad
  - after exhaust the first 256 bytes, generate next 256 bytes based on the previous set
  - apply the pad using XOR operator for both encryption and decryption
- this scheme essentially uses a PRNG with a very long period
  - all PRNGs repeat, eventually, since they are deterministic
  - analysis shows that the period (for repetition) is "overwhelmingly likely" to be greater than 10^{100} numbers
- used in SSL/TLS, WEP, and WPA

RC4 Operation

- Input is a key that can be from 1 to 256 bytes (i.e., from 8 to 2048 bits)
    - Processing
        1. Initialize arrays
        2. Generate initial permutation
        3. Generate key stream

- To encrypt: use bitwise XOR to combine key stream bit-by-bit with plaintext

- To decrypt: use bitwise XOR to combine key stream bit-by-bit with ciphertext

RC4: Initialize Arrays

Given: Key K expressed as a bit array

```
for( int i = 0; i < 256; i++ ) {
    S[ i ] = i;
    T[ i ] = K [ i mod K.length ]
}
```

Note: We express the algorithm in Java-style pseudocode

Example: for K = { 1, 1, 0, 1, 1, 0, 1, 1 }

```
S[i]: [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, ...
T[i]: [1, 1, 0, 1, 1, 0, 1, 1, 1, 1, 0, 1, 1, 0, 1, 1, 1, 1, 0, 1, 1, 0, ...
```

Note how key repeats, just like a Vigenere keyword

RC4: Generate Initial Permutation

- This step generates a pseudorandom permutation of the entire S array

```
j = 0;
for( int i = 0; i < 256; i++ ) {
        j = ( j + S[ i ] + T[ i ] ) mod 256
        Swap (  S[ i ],  S[ j ]  );
}
```

Swap(i, j) interchanges the values of S[i] and S[j]

Example: for S and T arrays computed before:

```
S[i]: [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, …
T[i]: [1, 1, 0, 1, 1, 0, 1, 1, 1, 1, 0, 1, 1, 0, 1, 1, 1, 1, 0, 1, 1, 0, …
```

this step modifies S to become:

```
Initial perm of S[i]: [1, 205, 155, 104, 11, 92, 20, 28, 37, 251, …
```

RC4: Generate Key Stream

- This step cycles through all the elements of S[i] and, for each S[i], swaps it with another element S[j] and uses both elements to compute the index of S, the value of which is used as the next key.
- After S[255] is reached, the proces continues, starting over at S[0]

```
i, j = 0;
while( true ) {
        i = ( i + 1 ) mod 256
        j = ( j + S[ i ] ) mod 256
        Swap (  S[ i ],  S[ j ]  );
        t = ( S[ i ] + S[ j ] ) mod 256;
        nextKey = S( t );
}
```

The value "t" is the index into S that is generated from i and j. The value of S[t] is the next key in the key stream

- Continuing our example, this produces the key stream:

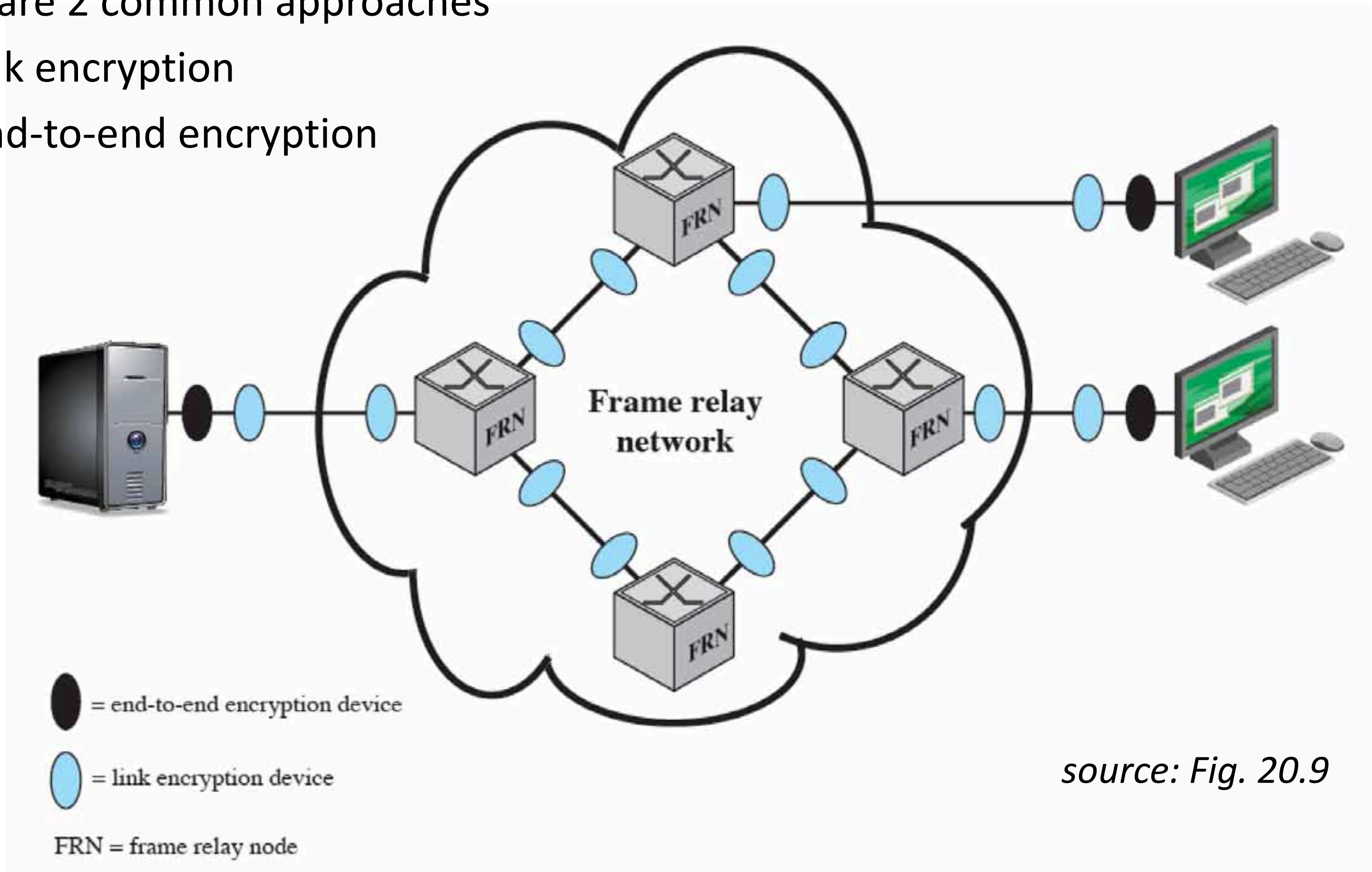
```
Key stream: 35, 211, 203, 36, 163, 148, 132, 68, 62, 194, 218, 78, 200, 119, 48, 250, 85, 235, 123, 211, 2, 12, …

Binary stream: 0010001111010011110010110010010010100011100101001000010001000100011 …
```

Outline

- The Advanced Encryption Standard (AES)
- The RC4 Stream Cipher
- Location of Symmetric Encryption Devices
- Symmetric Key Distribution

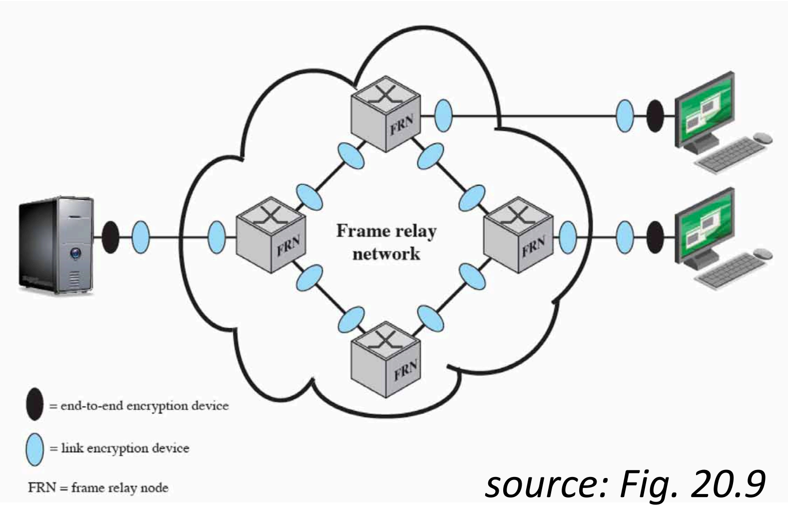
Location of Encryption

- Encryption is the most common means for securing network communications
- There are 2 common approaches
  - link encryption
  - end-to-end encryption



source: Fig. 20.9

Comparison of Methods



source: Fig. 20.9

- Link encryption
  - each packets encrypted
    - all traffic secured
  - equipment required for each link
  - traffic must be decrypted and re-encrypted at each node
    - vulnerable at each node/switch
- End-to-end encryption
  - encrypted at source, decrypted at destination
    - user data is secure
  - cannot encrypt frame header
  - vulnerable to traffic analysis
- Preferred method: Do both
  - user data secure, even at nodes/switches
  - not vulnerable to traffic analysis

Outline

- The Advanced Encryption Standard (AES)
- The RC4 Stream Cipher
- Location of Symmetric Encryption Devices
- Symmetric Key Distribution

Key Distribution Alternatives

- Security of cryptosystem depends on the key distribution method used
  - secrecy of keys
  - frequently changing keys Why?

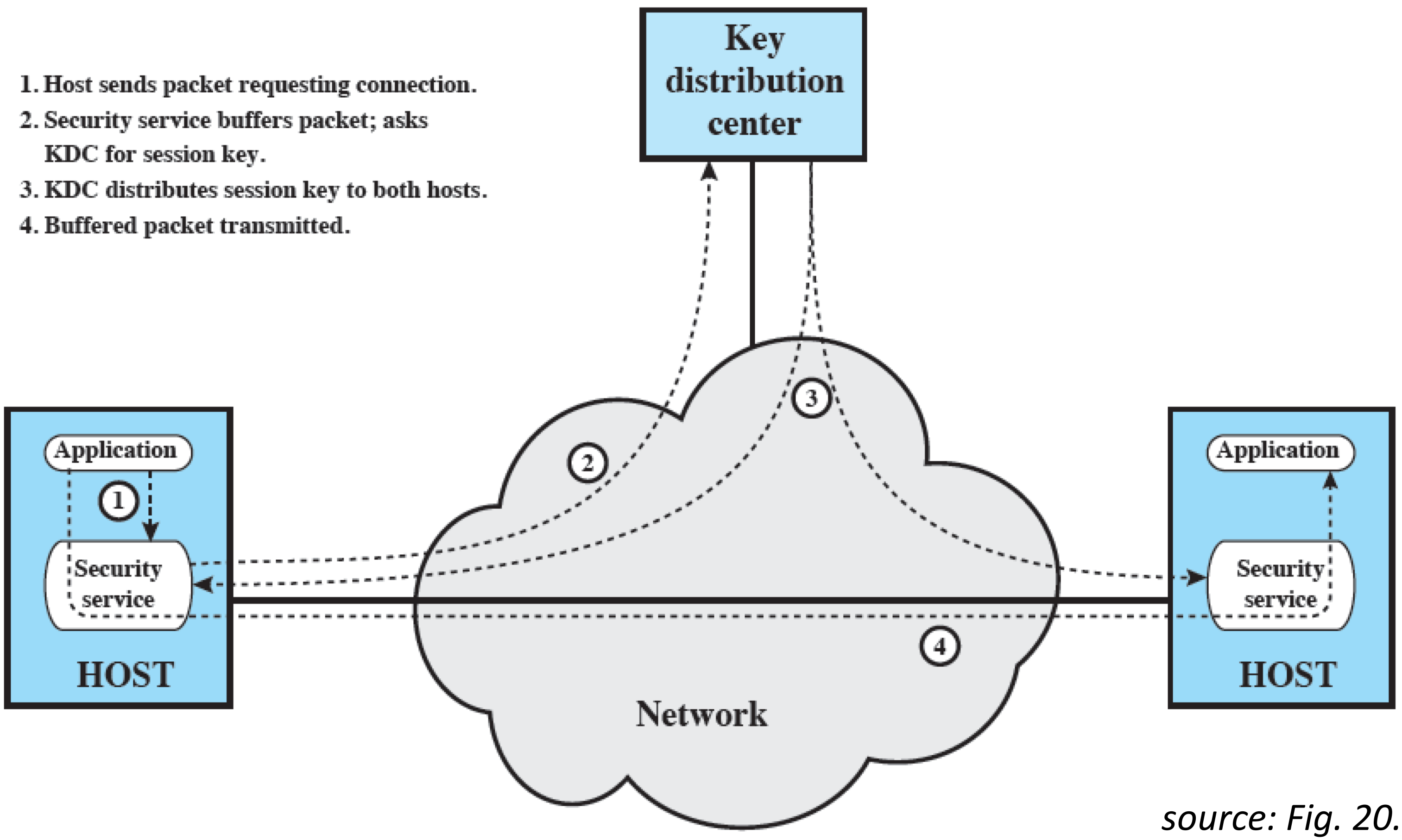
- Key distribution alternatives for secure communication between A and B:

  1. Key could be selected by A and physically delivered to B
  2. Key could be selected by a third party and physically delivered to both A and B
  3. If A and B have previously communicated, could send new key by encrypting it with prior key
  4. If A and B each have an encrypted connection to a third party C, then C could deliver a key on the encrypted links to A and B

- Option 3 is dangerous: if any key is compromised, all subsequent keys are known
- Option 4 is preferable for end-to-end encryption

Key Distribution Using a KDC

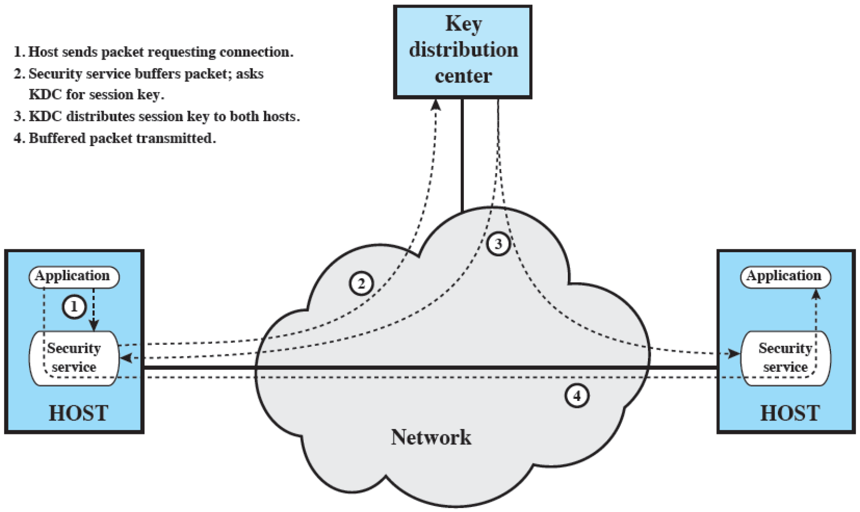
- The preferred method for distributing symmetric keys for end-to-end encryption



source: Fig. 20.10

Key Distribution Components

- **Key Distribution Center (KDC)**
  - responsible for distributing session keys
  - knows who is authorized to communicate to whom
- **Security Service Module (SSM)**
  - at each host
  - requests session key from KDC
    - using permanent key shared with KDC



source: Fig. 20.10

- **Permanent key** is used for communicating between hosts and KDC
- **Session key** is used for encrypted communications between the hosts