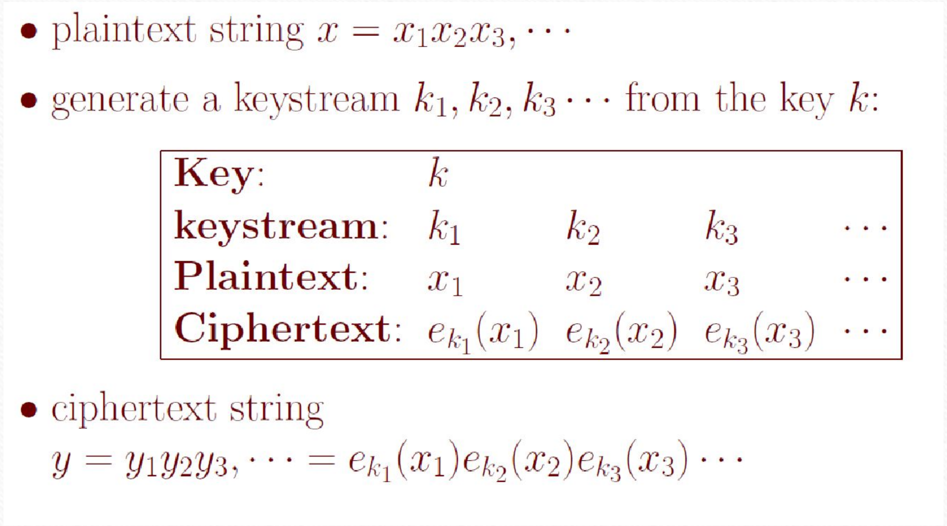
Stream Cipher

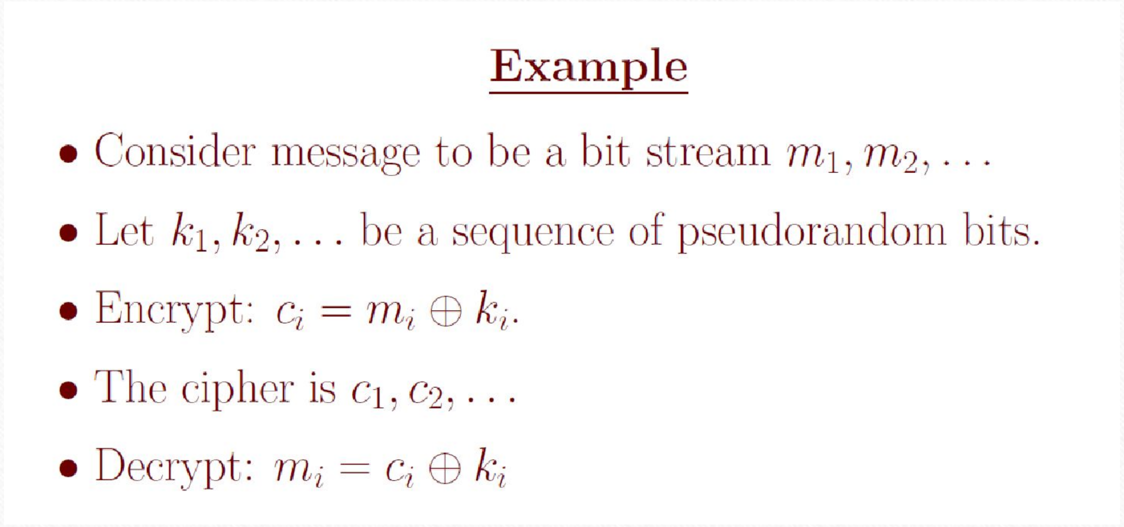


- plaintext string $x = x_1x_2x_3, \cdots$
- generate a keystream $k_1, k_2, k_3 \cdots$ from the key k:

Key:	k			
keystream:	k_1	k_2	k_3	$\cdots$
Plaintext:	x_1	x_2	x_3	$\cdots$
Ciphertext:	$e_{k_1}(x_1)$	$e_{k_2}(x_2)$	$e_{k_3}(x_3)$	$\cdots$

- ciphertext string
  $y = y_1y_2y_3, \cdots = e_{k_1}(x_1)e_{k_2}(x_2)e_{k_3}(x_3)\cdots$

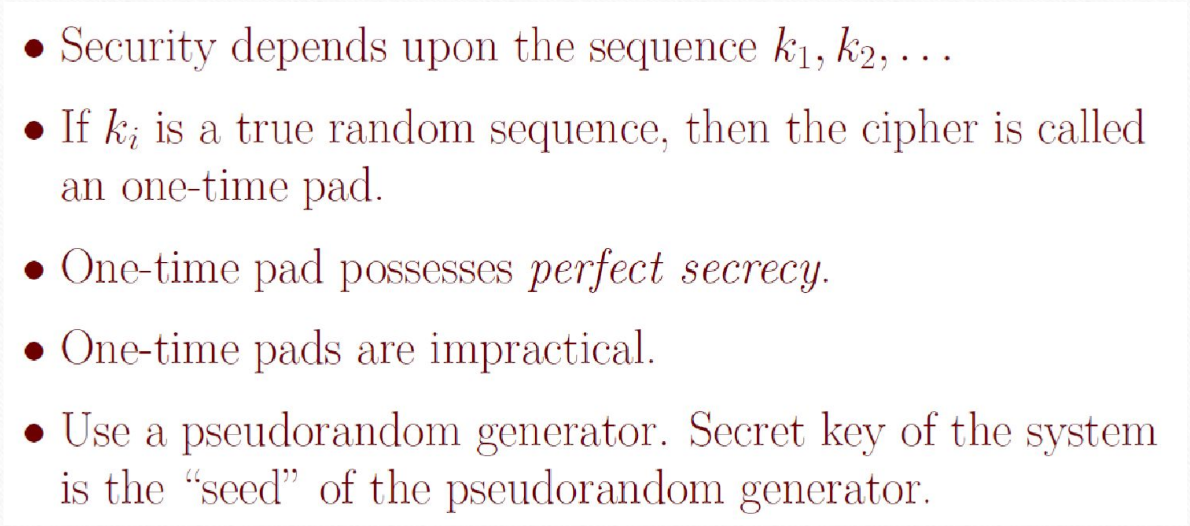
Stream Cipher



Example

- Consider message to be a bit stream $m_1, m_2, \ldots$
- Let $k_1, k_2, \ldots$ be a sequence of pseudorandom bits.
- Encrypt: $c_i = m_i \oplus k_i$.
- The cipher is $c_1, c_2, \ldots$
- Decrypt: $m_i = c_i \oplus k_i$

Stream Cipher



- Security depends upon the sequence $k_1, k_2, \ldots$
- If k_i is a true random sequence, then the cipher is called an one-time pad.
- One-time pad possesses *perfect secrecy*.
- One-time pads are impractical.
- Use a pseudorandom generator. Secret key of the system is the “seed” of the pseudorandom generator.

Stream Cipher – RC4

- One Time Pad (OTP) has perfect secrecy
  - requires key as long as plaintext
- Stream cipher approximates OTP by using PRNG
- A PRNG expands a short random seed into a long string that “looks random”
- A PRNG-based stream cipher has fundamental weaknesses
  - same stream cannot be used twice
  - highly malleable

- A proprietary cipher owned by RSA, designed by Ron Rivest in 1987.
- Became public in 1994.
- Simple and effective design.
- Variable key size, byte-oriented stream cipher.
- Widely used (web SSL/TLS, wireless WEP).

Stream Cipher – RC4

- RC4 is a variable key length stream cipher with byte-oriented operations.

- Fundamental to the RC4 algorithm is a 256 element array of 8-bit integers. It is called the **state vector** and denoted S.

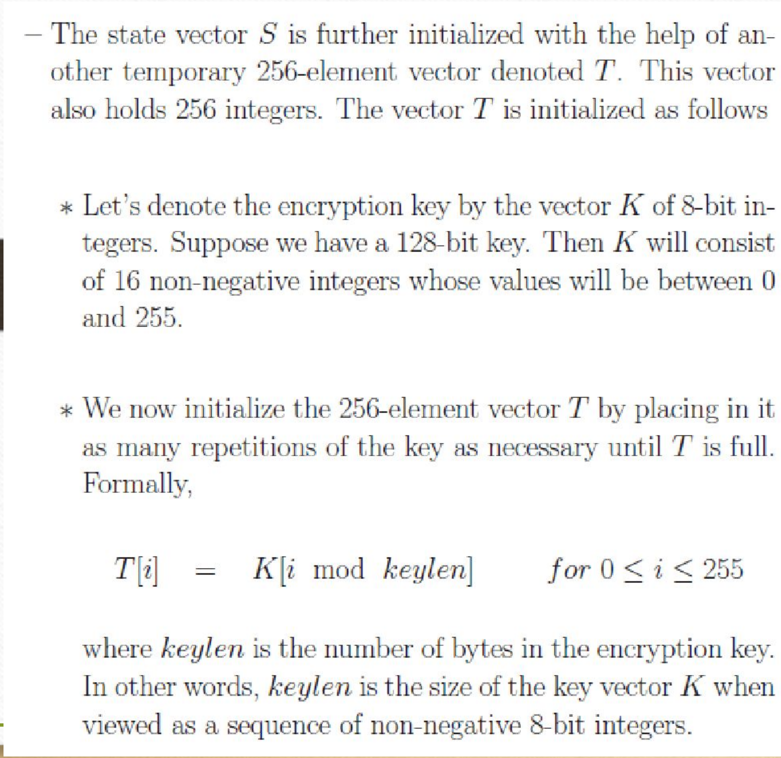
- The state vector is initialized with the encryption key. The exact initialization steps are as follows:

  – The state vector S is initialized with entries from 0 to 255 in the ascending order. That is

$$S[0] = 0x00 = 0$$

$$\begin{array}{rcccc} S[1] & = & 0x01 & = & 1 \\ S[2] & = & 0x02 & = & 2 \\ S[3] & = & 0x03 & = & 3 \\ \ldots\ldots & & & & \\ \ldots\ldots & & & & \\ S[255] & = & 0xFF & = & 255 \end{array}$$

RC4 Initialization



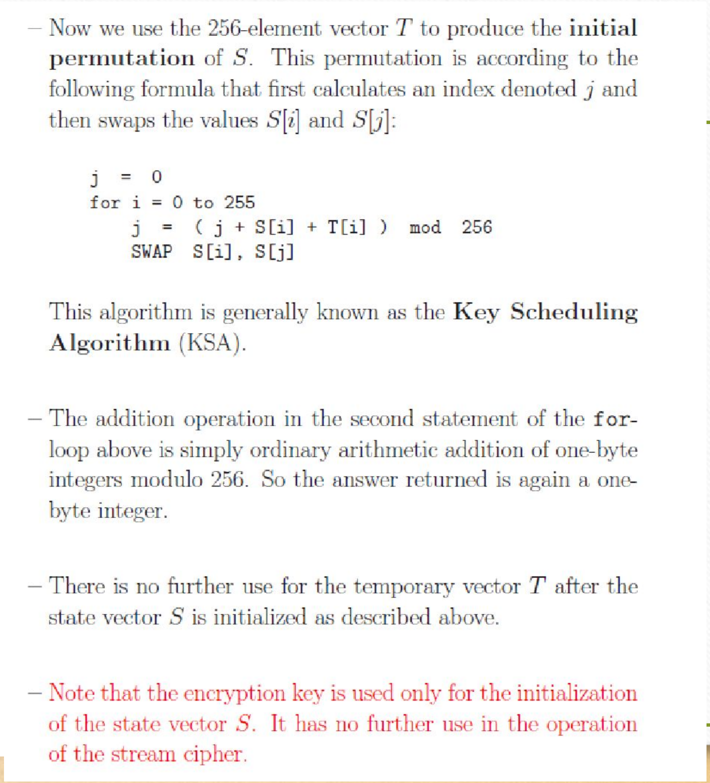
– The state vector S is further initialized with the help of another temporary 256-element vector denoted T. This vector also holds 256 integers. The vector T is initialized as follows

∗ Let's denote the encryption key by the vector K of 8-bit integers. Suppose we have a 128-bit key. Then K will consist of 16 non-negative integers whose values will be between 0 and 255.

∗ We now initialize the 256-element vector T by placing in it as many repetitions of the key as necessary until T is full. Formally,

$$T[i] \quad = \quad K[i \mod keylen] \qquad for\ 0 \leq i \leq 255$$

where *keylen* is the number of bytes in the encryption key. In other words, *keylen* is the size of the key vector K when viewed as a sequence of non-negative 8-bit integers.



– Now we use the 256-element vector T to produce the **initial permutation** of S. This permutation is according to the following formula that first calculates an index denoted j and then swaps the values $S[i]$ and $S[j]$:

```
j  =  0
for i = 0 to 255
    j  =  ( j + S[i] + T[i] )  mod  256
    SWAP  S[i], S[j]
```

This algorithm is generally known as the **Key Scheduling Algorithm** (KSA).

– The addition operation in the second statement of the `for`-loop above is simply ordinary arithmetic addition of one-byte integers modulo 256. So the answer returned is again a one-byte integer.

– There is no further use for the temporary vector T after the state vector S is initialized as described above.

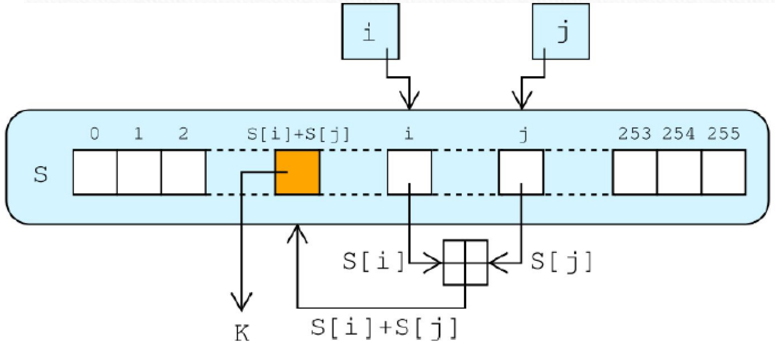
– Note that the encryption key is used only for the initialization of the state vector S. It has no further use in the operation of the stream cipher.

RC4 Encryption

- Now that the state vector S is initialized, we are ready to describe how the **pseudorandom byte stream** is generated from the state vector. Recall that when you are using a stream cipher, as each byte of the plaintext becomes available, you XOR it with a byte of the pseudorandom byte stream. The output byte is what is transmitted to the destination.

- The following procedure generates the pseudorandom byte stream from the state vector

```
i, j  =  0
while ( true )
    i  =  ( i + 1 )  mod  256
    j  =  ( j + S[i] )  mod  256
    SWAP S[i], S[j]
    k  =  ( S[i]  +  S[j] )  mod  256
    output S[k]
```



RC4 Cryptanalysis

Bias in the initial output:

Case 1: If $S[2] = 0$, $S[1] \neq 2$, then pr[2nd output = 0] = 1

Proof:

Iteration 1: initially $i=j=0$; i is incremented pointing to $S[1]$; Lets say, $S[1] = X$; j incremented $0+S[1] = X$, so points to X; Lets say, $S[X] = Y$; X and Y are swapped; first output is $S[X+Y]$

Iteration 2: i incremented to 2; j incremented to $X+S[2]$ which is X; values of X and 0 (at position 2) are swapped. So, the second output $S[X+0] = S[X] = 0$

RC4 Cryptanalysis

Case 2: Given a random initial state, the value of 2nd output is 0 with probability 2/256

Proof: when $S[2] = 0$, $pr[2^{nd} \text{ output} = 0] = 1$

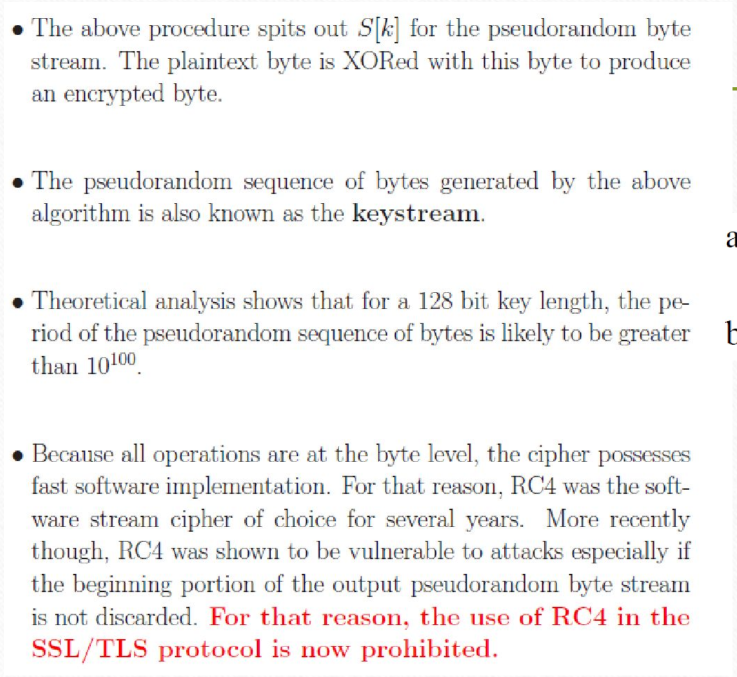
when, $S[2] \neq 0$, then the $pr[\text{output} = 0] = 1/256$

$pr[2^{nd} \text{ output} = 0]$

$= pr[2^{nd} \text{ output} = 0 | S[2] = 0] . pr[S[2] = 0] + pr[2^{nd} \text{ output} = 0 | S[2] \neq 0] . pr[S[2] \neq 0]$

$= 1. 1/256 + 1/256 . (1-1/256) = (1+1 - 1/256) . 1/256 \approx 2/256$

RC4 Conclusion



- The above procedure spits out $S[k]$ for the pseudorandom byte stream. The plaintext byte is XORed with this byte to produce an encrypted byte.
- The pseudorandom sequence of bytes generated by the above algorithm is also known as the **keystream**.
- Theoretical analysis shows that for a 128 bit key length, the period of the pseudorandom sequence of bytes is likely to be greater than 10^{100}.
- Because all operations are at the byte level, the cipher possesses fast software implementation. For that reason, RC4 was the software stream cipher of choice for several years. More recently though, RC4 was shown to be vulnerable to attacks especially if the beginning portion of the output pseudorandom byte stream is not discarded. **For that reason, the use of RC4 in the SSL/TLS protocol is now prohibited.**

a. Find the period of the following generator using seed x_0=1:

$$x_n = 5x_{n-1} \bmod 2^5$$

b. Now repeat part a with seed x_0 =2