Attention! If the same generated key stream ***K*** is used twice in Vernam cipher for any two messages M_1 and M_2 encryption, then ***eavesdropping*** adversary can obtain data D_b which is equal to bitwise XOR between M_1 and M_2. Let ciphertexts C_1 and C_2 are obtained by the following encryption with the same generated key stream ***K***

$C_1 = M_1 \oplus K$,

$C_2 = M_2 \oplus K$,

where $\oplus$ is bitwise XOR operation.

Then eavesdropping adversary computes the following data d_b

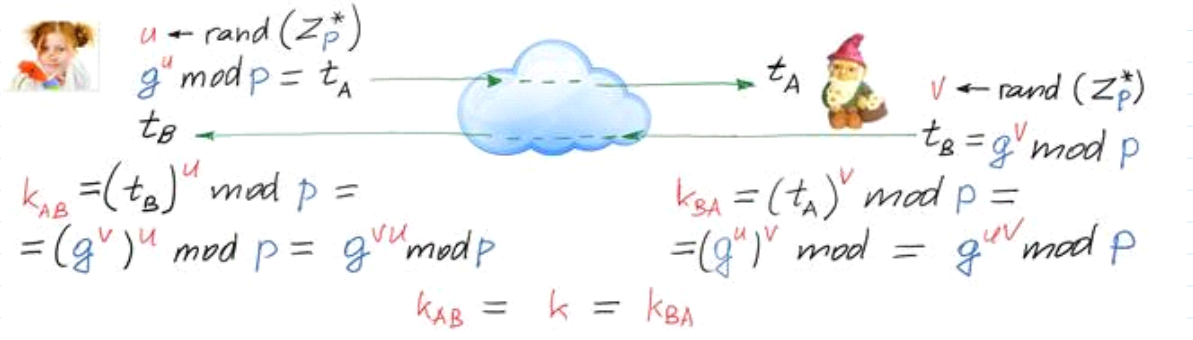
$D_b = C_1 \oplus C_2 = M_1 \oplus K \oplus M_2 \oplus K = M_1 \oplus M_2 \oplus K \oplus K = M_1 \oplus M_2 \oplus 0 = M_1 \oplus M_2$.

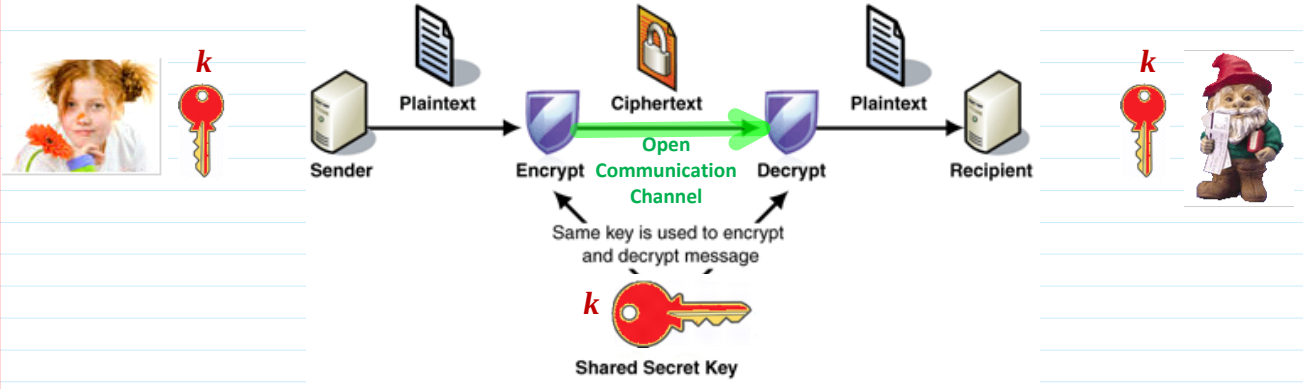
It is reckoned as a ***crucial insecurity*** since cryptanalysis of data D_b is significantly facilitated and both M_1 and M_2 can be disclosed.

Moreover, if any message of two M_1 or M_2 are revealed by some circumstances, say message M_2, then the other message M_1 becomes clear to the adversary by computing

$D_b \oplus M_2 = M_1 \oplus M_2 \oplus M_2 = M_1 \oplus 0 = M_1$.

Never use the same generated key stream ***K*** twice in Vernam cipher!






```
>> kAB = int64(195681379)
kAB = 195681379
>> k=kAB
k = 195681379
>> kb=dec2bin(k)
kb = 101 1101 0100  1 1101 1100 0110 0011

>> m1=120000
m1 = 120000
```

```
>> c1b=binaryxor(m1b,kb)
c1b = 101 1101 0100 0 0000 1000 1010 0011
>> c2b=binaryxor(m2b,kb)
c2b = 101 1101 0100 1 1101 1100 0110 1010
 >> c12b=binaryxor(c1b,c2b)
c12b=000 0000 0000 1 1101 0100 1100 1001

m2b = 1001
```

```
10111010100 0 0000 1000 1010 0011
⊕
10111010100 1 1101 1100 0110 1010

00000000000 1 1101 0100 1100 1001
                          m2b  =  1001
```

```
>> m1=120000
m1 = 120000
>> m1b=dec2bin(m1)
m1b = 1 1101 0100 1100 0000
>> m2=9
m2 = 9
>> m2b=dec2bin(m2)
m2b = 1001
```

```
m2b = 1001

Assume that Adversary spies found out
        m2 and hence m2b
>> mm1b=binaryxor(c12b,m2b)
mm1b = 1 1101 0100 1100 0000
>> mm1=bin2dec(mm1b)
mm1 = 120000
Then adversary detected the other secret
        sum m1 = 120000
```




```
% Octave realization of AES - Advanced Encryption Standard
% AES128(in,kh32,NR,fun)
% Advanced Encryption Standard symmetric cipher with key length of 128 bits
% Encryption is performed for 1 block of length 128 bits or 16 ASCII symbols
%
% in - plaintext/ciphertext of string type: maximum 16 symbols or shorter
%
% kh32 - shared secret key in hexadecimal number of length=32 (128 bits)
% kh32 can be obtained when shared decimal key k is given using commands:
%       >> k=int64(randi(2^28-1))
%       k = 160966896
%       >> kh32=dec2hex(k,32)
%       kh32 = 000000000000000000000000099828F0
%
% NR - Number of Rounds (e.g. Nr = 10)
%     The greater NR, the greater security is achieved but the speed of encryption is lower
%     The smaller NR, the lower security is of encryption but the speed of encryption is higher
%     The least number of NR is 1 and in this case security lack is evident
%
% fun - letter determining either encription: fun='e' or decryption: fun='d' functions
%
% Encryption example:
% >> in = 'Hello Bob';
% >> kh32 = '000000000000000000000000099828F0';
% >> NR = 10;
% >> Ch = AES128(in,kh32,NR,'e')
% ASCII_e =   ?1  ~mV              % ciphertext in ASCII format
% Ch = 0f9a2c08d191310fb27ed16d90f45686   % ciphertext in hexadecimal format
%
% Decryption example:
% >> Dh = AES128(Ch,kh32,NR,'d')
% Dh = 00000000000048656c6c6f7720426f62   % decrypted message in hex format
% D = Hello Bob                    % Decrypted message in ASCII format
%
```