en_US

Samsung Research

SSTF 2022 | Hacker's Playground

# Tutorial Guide

## RC four

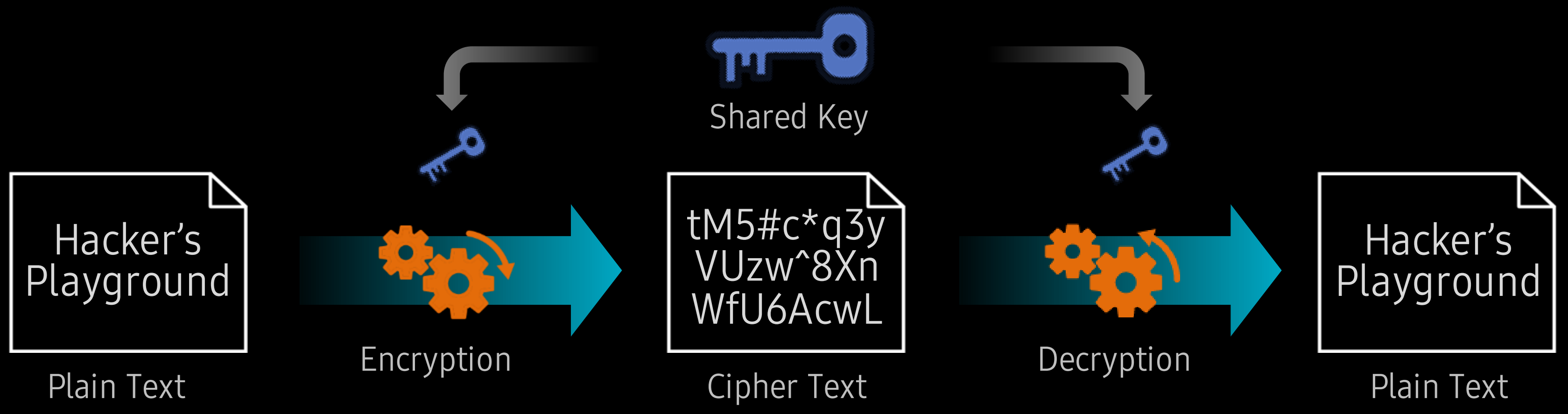
Crypto

# Two kinds of encryption

| | Symmetric Encryption | Asymmetric Encryption |
|---|---|---|
| Key | One shared key for encryption and decryption | Mathematically coupled public key and private key |
| Typical Key Size | 128~256 bits | 1024~3072 bits (for RSA) |
| Performance | High | Low, because it's a complex mathematical computation |
| Main Purpose | Data Encryption | Digital Signature/Certificate |
| Representative Algorithms | DES, AES, RC4 | RSA, DSA, ECC |

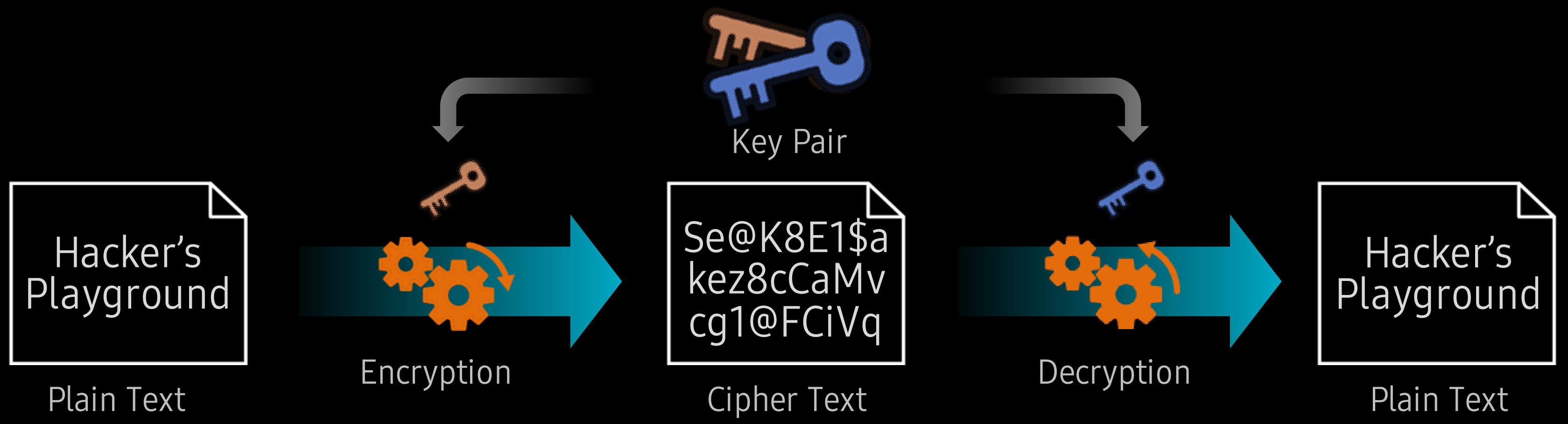
# Two kinds of encryptions

**Symmetric Encryption**




**One key** used for both encryption and decryption

**Asymmetric Encryption**



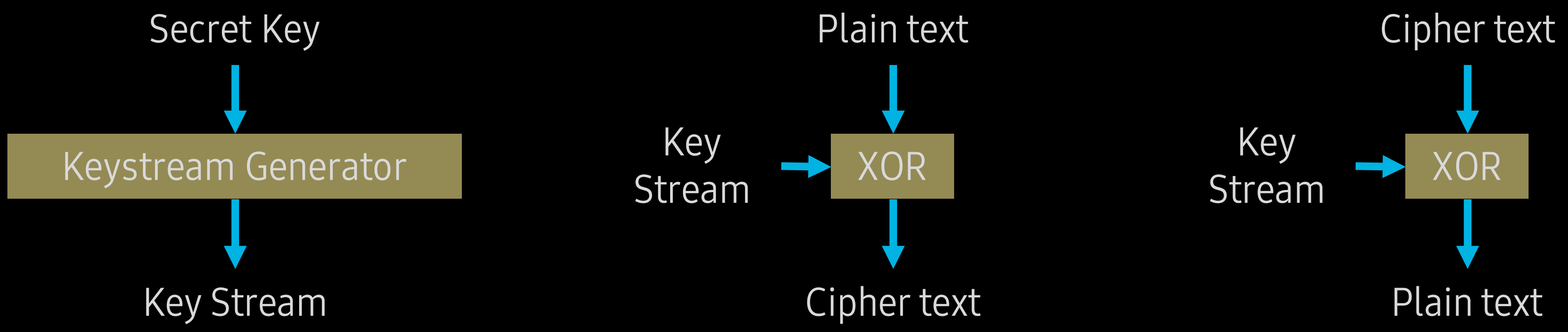

**Key Pair** consisting of encryption key and decryption key

# RC4 (a.k.a ARC4)

## A representative stream cipher

- Stream cipher is a branch of symmetric key cipher.
- XOR-based common encryption/decryption processing

## Working




Step 1. Key stream generation

Step 2-1. Encryption

Step 2-2. Decryption

Let's solve
Crypto quiz!

# Quiz #1
& solution

# Quiz #1

```
KeyStream_From_RC4 ="<y4)ky&=zuw(8*#3*<q4Quw)o+"
RC4_CipherText ="k6cv36tb1<9ogcplby#qpT"
```

Download the source code [HERE].

- Simple python code
- Can you get the plaintext?
- Try it before you see the solution.

# Solution for Quiz #1

- It’s quite simple. To decrypt the RC4 ciphertext, just XOR it with the key stream.

```python
KeyStream_From_RC4 = "<y4)ky&=zuw(8*#3*<q4Quw)o+"
RC4_CipherText = "k6cv36tb1<9ogcplby#qpT"

plaintext = ""
for k, c in zip(KeyStream_From_RC4, RC4_CipherText):
    plaintext += chr(ord(k) ^ ord(c))
print(plaintext)
```

Bytewise XORing of ciphertext and key stream

- And you did it!

```
$ python3 ex.py
WOW_XOR_KING_IS_HERE!!
$
```

# Quiz #2
& solution

# Quiz #2

```python
from Crypto.Cipher import ARC4
from binascii import hexlify
from secret import key, flag

def encrypt(data):
    return ARC4.new(key).encrypt(data)

ct = b""
for ch in flag:
    ct += encrypt(ch)

print("Ciphertext = ", hexlify(ct).decode())

'''
$ python3 challenge.py
Ciphertext = 6f47474c06086f47085c47085c404d08464d505c085b5c494f4d09
'''
```

Download the source code HERE.

- ARC4 module generates key stream and XORs with the input.
- RC4 ciphertext is given, but key is not known.
- Can you get the plaintext?
- Try it before you see the solution.
- HINT: You may need a little bit brute-forcing.

# Solution for Quiz #2

```python
from Crypto.Cipher import ARC4
from binascii import hexlify
from secret import key, flag

def encrypt(data):
    return ARC4.new(key).encrypt(data)

ct = b""
for ch in flag:
    ct += encrypt(ch)

print("Ciphertext = ", hexlify(ct).decode())

'''
$ python3 challenge.py
Ciphertext = 6f47474c06086f47085c47085c404d08464d505c085b5c494f4d09
'''
```

- **According to the source code...**
  - The flag is not encrypted at once.
  - It's split for each byte, encrypted, and put back together.
- **Each letter of the flag is XORed with the first byte of the key stream.**
  - Only one byte of the key stream is used.
  - The entire flag data can be recovered by finding the value of the one byte.

# Solution for Quiz #2

- Try every possible case.
  - 1 byte is group of 8 bits, so there can be $2^8$ = 256 cases

```python
from binascii import unhexlify

Ciphertext = unhexlify("6f47474c06086f47085c47085c404d08464d505c085b5c494f4d09")

for i in range(256):
    flag = ""
    for ch in Ciphertext:
        flag += chr(ch ^ i)
    else:
        print(i, flag)
```

Try XORing for every possible case

- We got a meaningful sentence among them.

```
38 Iaaj .Ia.za.zfk.`kvz.}zoik/
39 H``k!/H`/{`/{gj/ajw{/|{nhj.
40 Good. Go to the next stage!
41 Fnne/!Fn!un!uid!odyu!ru`fd
42 Emmf,"Em"vm"vjg"lgzv"qvceg#
```

# Let’s practice

Solve the tutorial challenge

# Challenge Definition

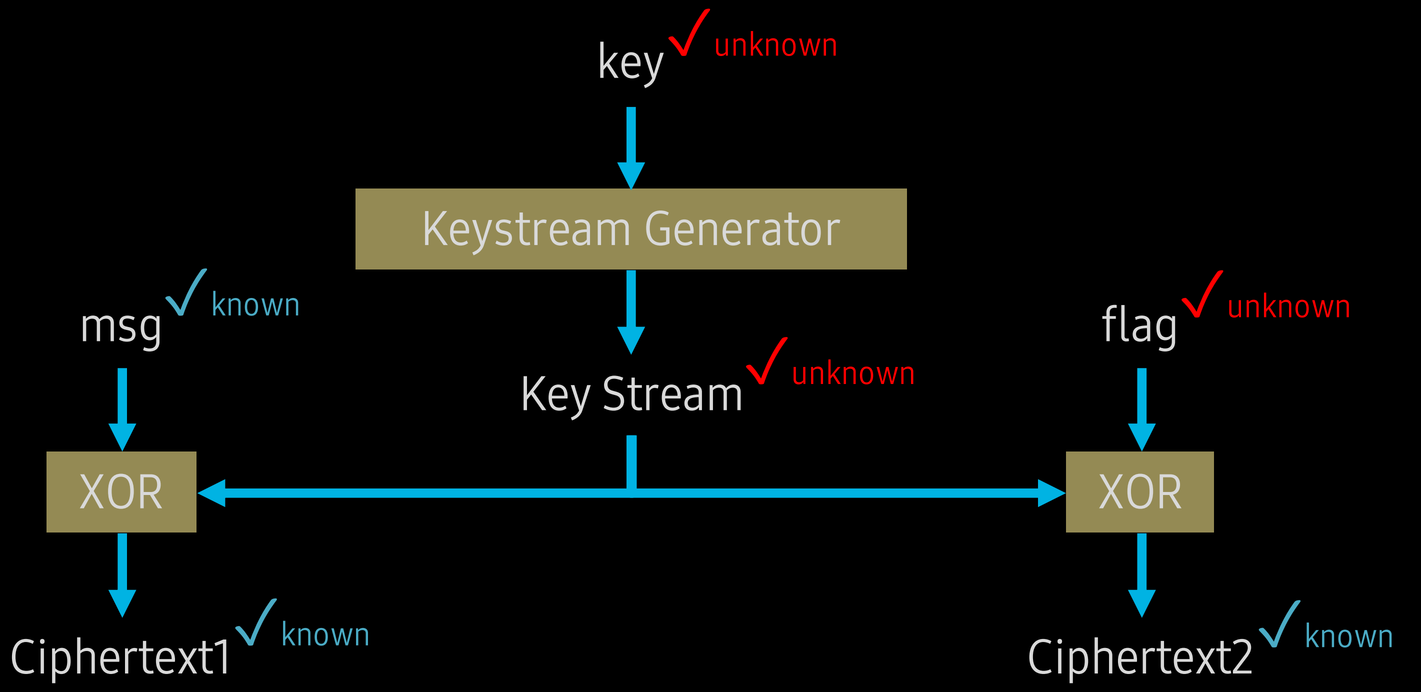
```python
from Crypto.Cipher import ARC4
from secret import key, flag
from binascii import hexlify

#RC4 encrypt function with "key" variable.
def encrypt(data):
    #check the key is long enough
    assert(len(key) > 128)

    #make RC4 instance
    cipher = ARC4.new(key)

    #We don't use the first 1024 bytes from the key stream.
    #Actually this is not important for this challenge. Just ignore.
    cipher.encrypt("0"*1024)

    #encrypt given data, and return it.
    return cipher.encrypt(data)

msg = "RC4 is a Stream Cipher, which is very simple and fast."

print (hexlify(encrypt(msg)).decode())
print (hexlify(encrypt(flag)).decode())
```

- There are
  - one **key**,
    unknown
  - two plaintexts
    **flag**: unknown
    **msg**: known
  - and two ciphertexts
    both of them are known

```
$ cat output.txt
634c3323bd82581d9e5bbfaaeb17212eebfc975b29e3f4452eefc08c09063308a35257f1831d9eb80a583b8e28c6e4d2028df5d53df8
624c5345afb3494cdd6394bbbf06043ddacad35d28ceed112bb4c8823e45332beb4160dca862d8a80a45649f7a96e9cb
```

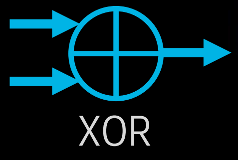
Ciphertext1
Ciphertext2

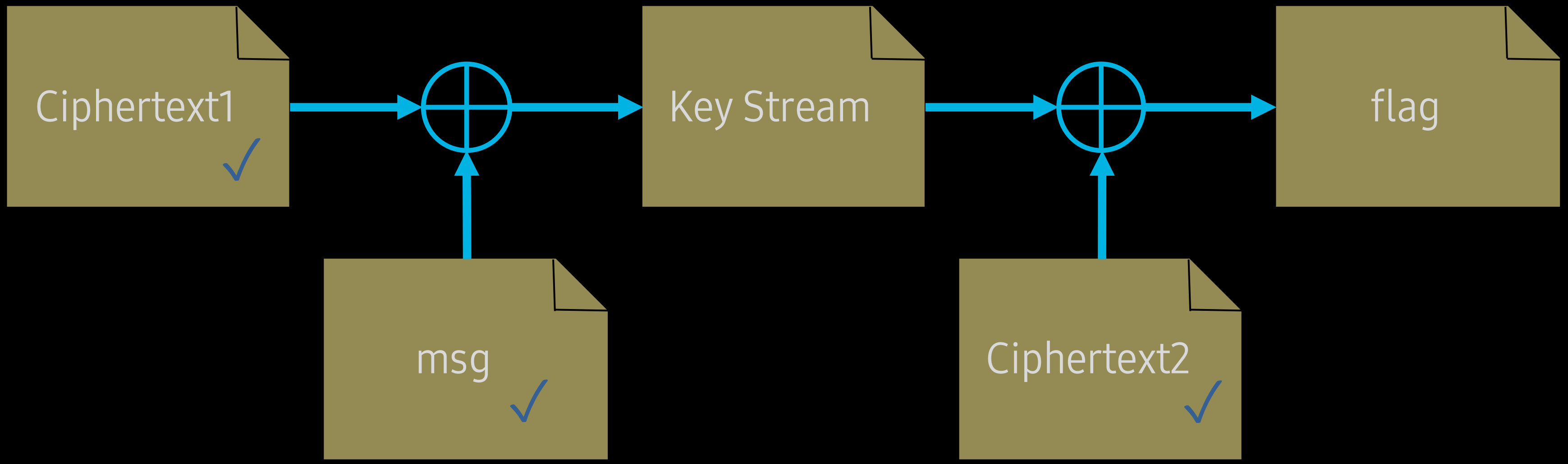
# Let’s see




- Can you see? We can find the **flag**, even without **key**!
  - Because when a⊕ b = c, a⊕ c = b.

# It's an easy logic!




- Step 1. We can recover the Key Stream from the known plaintext and ciphertext pair.
- Step 2. We can recover the Ciphertext2 because now we know the Key Stream.
- Step 3. Now we got the flag!! :)

# Does it really work?

```python
from binascii import unhexlify

ct1, ct2 = open("output.txt").read().strip().split("\n")
msg = b"RC4 is a Stream Cipher, which is very simple and fast."

ct1 = unhexlify(ct1)
ct2 = unhexlify(ct2)

l = min(len(ct1), len(ct2))
r = ""
for (c1, m, c2) in zip(ct1[:l], msg[:l], ct2[:l]):
    r += chr((c1 ^ m) ^ c2)

print (r)
```

Bytewise XORing of ct1, msg, and ct2

✔ Yes, it does!

Give it a shot!

```
$ python solve.py
SCTF{[redacted]}
```