Algorithm Types and Modes

Sources:

- *Applied Cryptography* by B. Schneier
- *Modern Cryptography* by Wenbo Mao

There are two basic types of symmetric algorithms:

- *Block ciphers* operate on blocks of plaintext and ciphertext (usually 128 bits).

- *Stream ciphers* operate on streams of plaintext and ciphertext one bit or byte at a time.

Block ciphers are easier to implement in software.

Stream ciphers are more suitable for hardware encryption on a digital communications channel.

A cryptographic *mode* combines the basic cipher with some feedback and simple operations.

Electronic Cookbook Mode

In *electronic cookbook mode* (*ECB*), a block of plaintext encrypts into a block of ciphertext.

You don't have to encrypt a file linearly, so ECB is useful with random-access files like databases.

If two plaintext blocks are the same, the corresponding ciphertext blocks will be identical.

If a cryptanalyst has the plaintext and ciphertext of several messages, he can begin to compile a *code book* without knowing the key.

The cryptanalyst can obtain a lot of information this way if messages contain *redundancies* such as stereotyped beginnings or endings.

Block Replay

A more serious problem with ECB is that an adversary can fool a recipient by *modifying* message blocks.

Mallory can remove, repeat, or interchange blocks.

He doesn't need to know the key or the algorithm.

Example: Mallory can intercept bank deposit messages and replace the name and account blocks with those from a deposit to his own account.

Cipher Block Chaining Mode

To prevent replay attacks, *chaining* adds a *feedback* mechanism to a block cipher.

The results of the encryption of previous blocks are fed back into the encryption of the current block.

Thus, each ciphertext block is dependent on *all* of the previous plaintext blocks.

In *cipher block chaining* (*CBC*) mode, the plaintext is XORed with the previous ciphertext block before it is encrypted.

After a plaintext block is encrypted, the resulting ciphertext is stored in a *feedback register*.

Before the next plaintext block is encrypted, it is XORed with the feedback register.

A ciphertext block is decrypted normally and is also saved in a feedback register.

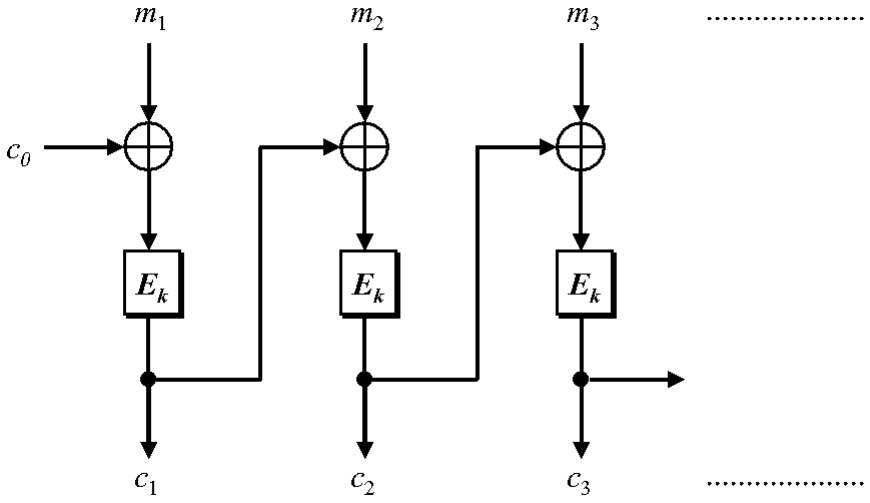
After the next block is decrypted, it is XORed with the results of the feedback register.

Mathematically, we have:

$$C_i = E_k(P_i \oplus C_{i-1})$$

$$P_i = C_{i-1} \oplus D_k(C_i)$$

Diagram [RSA Security]:



With CBC mode, two identical messages will still encrypt to the same ciphertext.

Two messages that begin the same will encrypt the same way up to the first difference.

This can be prevented by encrypting *random data* as the first block.

This block is called the *initialization vector* (*IV*).

When the user decrypts this block, he uses it to fill the feedback register and then discards it.

The IV should be unique for each message encrypted with the same key.

The IV need not be secret.

Stream Ciphers

Stream ciphers convert plaintext to ciphertext one bit or byte at a time.

A *keystream generator* outputs a stream of bits k_1, k_2, ..., k_i, ...

This keystream is XORed with a stream of plaintext bits p_1, p_2, ..., p_i, ... to produce ciphertext bits:

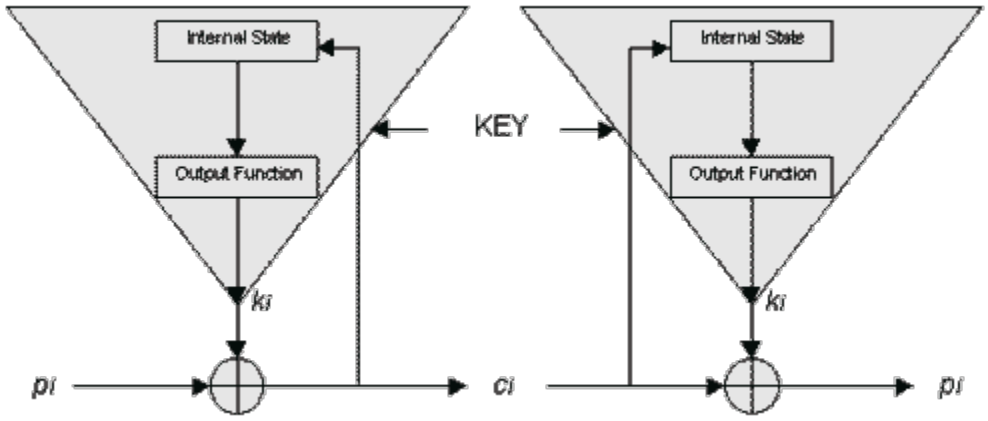
$$c_i = p_i \oplus k_i$$

During decryption, the ciphertext bits are XORed with an identical keystream to recover the plaintext bits:

$$p_i = c_i \oplus k_i$$

The keystream generator deterministically generates a pseudorandom bit stream.

Diagram of (self-synchronizing) stream cipher [ChipCenter.com]:



If a keystream generator produces the same bitstream every time it is turned on, the resulting cryptosystem will be easy to break.

If Alice ever has a ciphertext and corresponding plaintext, she can XOR them to recover the keystream.

If she has two different ciphertexts encrypted with the same keystream, she can XOR them and get two plaintext messages XORed with each other.

For these reasons, all stream ciphers have *keys*.