The usage and the impact of shift-registers on the CFB mode of operation

Pietro Spadaccino 1706250

12 October 2018

1 Introduction

Block ciphers are algorithms that can encrypt or decrypt data. They split the message into fixed-length blocks of bits and transform them using operations like mix, shift and substitution. They usually rely on a symmetric key, meaning that the key used during encryption and decryption will be the same. AES is perhaps the most widely-known block cipher, successor of the old DES, and it was adopted as a standard since 2002. While it provides strong security capabilities, the direct use of the cipher on the messages is discouraged for various reasons, like the possibility for the attackers to create a table with all known ciphertext/plaintext, exploiting the fact that the encryption of two messages using the same key generates the same ciphertext. The solution is using a mode of operation that provide confidentiality on the top of the block cipher. In this paper we will analyse the CFB mode of operation, in particular its variant using a shift register.

2 CFB Mode of Operation

The CFB mode of operation is designed to make a block cipher into a self-synchronizing stream cipher, meaning that, in case of a transmission error, the receiver is able to successfully decrypt most part of the data.

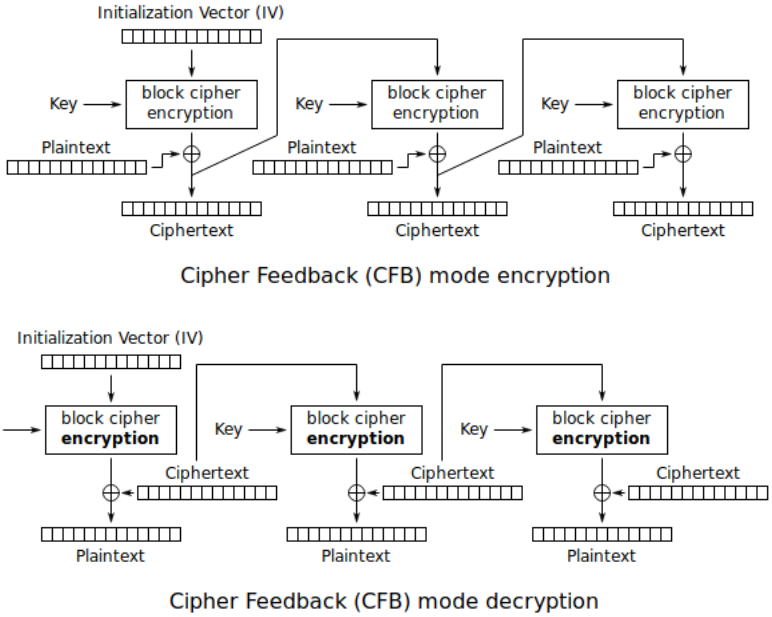

Figure 1: Scheme of CFB mode of operation

Similar to the CBC, the decryption can be done in parallel, because the blocks to be encrypted/decrypted don't depend on the blocks before. The same is not true about encryption, with every block being the XOR between the plaintext and the previous block encrypted:

$$C_i = E_k(C_{i-1}) \oplus P_i$$
$$P_i = E_k(C_{i-1}) \oplus C_i$$
$$C_0 = IV$$

A difference between CBC and CFB is that the block cipher is used only in the encrypting direction, reducing the possibilities for an attacker. Note that it's possible to use decryptors instead of encryptors, because the block transformations are conceptually the same.

A possibly unwanted feature of CFB is that the initialization vector (or the key) must be changed for every message. If that's not the case and if an attacker has access to a pair of ciphertext/plaintext $<C^1, P^1>$, he can decrypt the first block of another message C^2 without knowing IV nor the

key, exploiting the fact that the keystream K is the same:

$$C^1 = P^1 \oplus K$$
$$C^2 = P^2 \oplus K$$
$$C^1 \oplus C^2 = P^1 \oplus P^2$$
$$P^2 = C^1 \oplus C^2 \oplus P^1$$

where C^i/P^i is the first block of a ciphertext/plaintext i. Note that this attack is valid for the first block only, because the keystream is ciphertext-dependent and it will be different for the following blocks.

In CFB, an error during a decryption, like a bit flip in the ciphertext, is poorly propagated and affects only a restrained number of plaintext bits: if some bits i are flipped in the block j of the ciphertext then bits i will be flipped in the block j of the plaintext and all the block $j + 1$ will be corrupted, but the remaining blocks are decrypted without errors. However, a different type of error, like adding or subtracting a bit from the ciphertext, can compromise the decryption of all the message, shifting the ciphertext thus scrambling the keystream during the decryption. To overcome this difficulty CFB is often used with a shift register.

3 CFB with Shift Registers

Combining CFB with shift registers result in a stronger error handling in case of an error during decryption, respect to plain CFB.
The encryption/decryption scheme is similar to CFB, as shown in Figure 2, where the output of the block cipher is XORed with the plaintext or ciphertext to obtain respectively the ciphertext or plaintext, but the cipher operates on blocks of size b bits, while the message is encoded one block of size s at a time ($b > s$). After the encryption by the block cipher, only the first s bits are used in the XOR, while the others are discarded. Then the new XORed block of size s is pushed into the shift register, which will be the new input block of the cipher:

$$C_i = head(E_k(S_{i-1}), s) \oplus P_i$$
$$P_i = head(E_k(S_{i-1}), s) \oplus C_i$$
$$S_i = ((S_{i-1} << s) + C_i) mod 2^b$$
$$S_0 = IV$$

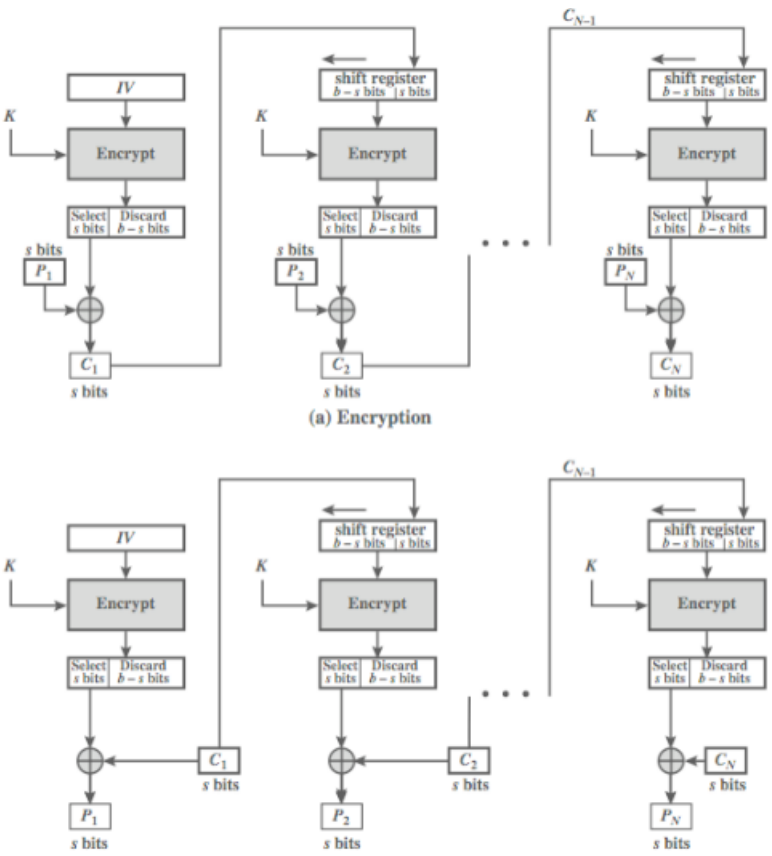

Figure 2: CFB mode of operation with shift register

where S_i is the state i of the register and $head(x, s)$ are the first s bits of x. Having to encrypt b bits for every s bits of message results in a significant overhead respect to CFB, using b/s times more the block cipher.

It shares most of the characteristics with plain CFB:

- encryption not parallelizable;
- decryption fully parallelizable;
- block cipher used only in one way;
- restrained error propagation.

Like discussed earlier, the IV must be changed for every message, or an attacker that has a pair of ciphertext/plaintext can easily decode the first block of another message. But, by using this variant of CFB, the first block that the attacker can obtain will be shorter, of size s instead of b.

The main advantage of using a shift register is that the decryption is more resistant not only to bit flips, but also to additions and subtractions from the ciphertext. Let's say that part of the ciphertext is lost because of transmission errors: while decrypting, the shift register will be missing some part of the original ciphertext and some bits are decoded incorrectly. However the shift register synchronizes itself after some correctly received blocks, reaching the same state it had during encryption, and the receiver is able to successfully decode the rest of the message.

If the error is a bit flip in the ciphertext, then then behavior is equal to plain CFB: the wrongly decoded output will have size of b bits, the same size of the block encrypted by the block cipher, plus another bit flip. After b bits correctly processed the shift register is in the correct state to continue to process the rest of the message. Instead, if some bits n are subtracted from the ciphertext, the shift register is able to synchronize itself, under the assumption:

$$n \bmod s = 0$$

Note that if this assumption cannot be done then the shift register will be shifted for every subsequent block by $n \bmod s$ compared to the state it had when encrypting. For this reason a common choice of s is $s = 1$.

Let's divide the ciphertext into m blocks of size s bits

$$ciphertext = \{C_1, C_2, ..., C_m\},$$

if n blocks, or ns bits, are lost starting from index i, then the resulting ciphertext is

$$ciphertext = \{C_1, C_2, ..., C_{i-1}, C_{i+n}, C_{i+n+1}, ..., C_m\}.$$

When the receiver decrypts the message, the resulting plaintext will have b/s blocks incorrectly decoded, and the whole will be shorter by n blocks compared to the original ciphertext:

$$plaintext = \{P_1, P_2, ..., P_{i-1}, P_i, P_{i+1}, ..., P_{i+n-2}, P_{i+n-1}, ..., P_{m-n}\}.$$

Blocks from P_1 to P_{i-1} and from P_{i+n+1} to P_{m-n} are successfully decrypted, while the remaining are not correct. An example is shown in Figure 3.

Original ciphertext	[C_1, C_2, C_3, C_4, C_5, C_6, C_7, C_8, C_9]
Received ciphertext	[C_1, C_4, C_5, C_6, C_7, C_8, C_9]

Iteration	Shift Register status	Current ciphertext block	Correct
1	[IV_1, IV_2, IV_3, IV_4]	C_1	YES
2	[IV_2, IV_3, IV_4, C_1]	C_4	NO
3	[IV_3, IV_4, C_1, C_4]	C_5	NO
4	[IV_4, C_1, C_4, C_5]	C_6	NO
5	[C_1, C_4, C_5, C_6]	C_7	NO
6	[C_4, C_5, C_6, C_7]	C_8	YES
7	[C_5, C_6, C_7, C_8]	C_9	YES

Decoded plaintext	[P_1, -, -, -, -, P_8, P_9]

Figure 3: Self-synchronization after bit loss. The shift register is formed by four ciphertext blocks ($b = 4s$), where IV_i is the i-th block of IV (the same is also valid for blocks of ciphertext C_i and plaintext P_i). In this example two blocks are lost: the decoded plaintext is shorter by 2 blocks compared to the original ciphertext and has $b/s = 4$ blocks incorrectly decrypted. Note that if the number of lost blocks (assuming contiguous) was greater than 4, the number of corrupted blocks in the plaintext would have been the same.