Using block ciphers

- DES is a type of block cipher, taking 64-bit plaintexts and returning 64-bit ciphertexts. We now discuss a number of ways in which block ciphers are employed in practice.

- The natural method of taking a plaintext message of more than 64 bits and breaking it up into 64-bit blocks $P = P_1, P_2, \ldots, P_L$ for processing into ciphertext blocks $C = C_1, C_2, \ldots, C_L$, where for each $j = 1, 2, \ldots, L$, $C_j = E_K(P_j)$ is the encryption of the corresponding block using keyword $K$, is called the **electronic codebook** mode (ECB). An obvious weakness in ECB is the following: suppose Eve can capture a fair amount of ciphertext and its associated plaintext; then she can build a fairly complete codebook that allows her to decrypt any further ciphertext blocks when the key $K$ is used, or even to pose as Alice and send her own ciphertext messages to Bob. In particular, the key itself is no longer a security for the cipher system.
One way to combat the flaws of ECB is to employ some feedback in the encryption process. Rather than use the normal encryption method \( C_j = E_K(P_j) \) for blocks of plaintext, first XOR the plaintext with the previous block of ciphertext before encryption:

\[
C_j = E_K(P_j \oplus C_{j-1}).
\]

This method is called **cipher block chaining** (CBC). Choose some **initialization vector** \( C_0 \) to get the process started. It is best to choose \( C_0 \) at random (it can even be communicated from Alice to Bob without encryption), else whenever the same plaintext is encrypted, the same ciphertext is generated; this produces behavior exactly like the ECB, which is precisely what we were trying to avoid! Decryption is straightforward, since

\[
D_K(C_j) = P_j \oplus C_{j-1} \quad \Rightarrow \quad P_j = D_K(C_j) \oplus C_{j-1}.
\]

Because each ciphertext block is used to determine the next ciphertext block, an error in transmission of the block will cause errors in the decryption of later blocks, an unfortunate circumstance indeed.
A drawback of both EBC and CBC is that to produce a block of ciphertext, one requires a full block of plaintext (and in the case of CBC, a previous block of ciphertext as well). The cipher feedback mode (CFB) is a way to employ the block cipher so as to implement is as a stream cipher. Rather than break the plaintext into 64-bit blocks, we use much smaller 8-bit blocks \( P_j \) \( (j = 1, 2, 3, \ldots) \), and encrypt as follows:

\[
\begin{align*}
O_j &= L_8(E_K(X_j)) \\
C_j &= P_j \oplus O_j \\
X_{j+1} &= R_{56}(X_j) \| C_j
\end{align*}
\]

where \( L_8(X) \) denotes selection of the 8 leftmost bits of the input \( X \), \( R_{56}(X) \) denotes selection of the 56 rightmost bits of the input, and \( X \| Y \) denotes concatenation of \( X \) with \( Y \). Decryption is easy:

\[
\begin{align*}
P_j &= C_j \oplus L_8(E_K(X_j)) \\
X_{j+1} &= R_{56}(X_j) \| C_j
\end{align*}
\]

In particular, the block cipher decryption function \( D_K \) is not needed. The method is presented in the schematic below.
Notice that the encryption in CFB is not applied to the plaintext, but to the blocks of the 64-bit words $X_j$. In a sense this behaves similarly to a one-time pad method: the $X_j$ supply the key words $O_j$ that we XOR with the plaintext blocks to obtain the ciphertext blocks $C_j$. Also, note that since $C_j$ becomes the rightmost part of $X_{j+1}$, which then propagates leftward through subsequent $X$’s until it is dropped 8 rounds later, it follows that an error in transmission of a block of ciphertext corrupts at most 8 subsequent blocks of ciphertext before the system recovers.
As we have seen, both CBC and CFB allow errors in transmission to corrupt portions of the message. There is another mode of operation called **output feedback** (OFB) that employs a streaming effect and is free of this error propagation flaw. As with CFB, we break the plaintext into 8-bit blocks $P_j$ ($j = 1, 2, 3, ...$) and set a random 64-bit initialization vector $X_1$. We encrypt $X_j$ using key $K$, extract its leftmost 8-bit block $O_j$, XOR it with $P_j$ to obtain the ciphertext block $C_j$. But instead of appending $C_j$ to the rightmost 56 bits of $X_j$ to obtain $X_{j+1}$, we append $O_j$ instead.
The general procedure is formulated as

\[
O_j = L_8 (E_K (X_j))
\]

\[
X_{j+1} = R_{56} (X_j) \parallel O_j
\]

\[
C_j = P_j \oplus O_j
\]

This technique requires encryption only of subblocks of the \( X_j \). Thus, the \( O_j \) can be determined long in advance, which speeds considerably the \( C_j = P_j \oplus O_j \) computations.

Moreover, errors in transmission of a block of ciphertext do not corrupt any further blocks of the ciphertext. Nonetheless, there is still a flaw here: if Eve captures both \( P_j \) and \( C_j \), she can compute \( P_j \oplus C_j = O_j \), which gives her the opportunity to craft her own plaintext \( P'_j \) and ciphertext \( C'_j = P'_j \oplus O_j \) to send surreptitiously to Bob.

- Finally, the **counter** mode (CTR) works just like OFB except that \( X_{j+1} \) is determined from \( X_j \) not by appending a block to the rightmost bits, but simply by adding 1 (binary addition, not XOR), with the additional stipulation that if \( X_j = 111\cdots1 \), then we take \( X_{j+1} = 000\cdots0 \). The schematic now becomes
This method shares many features with OFB: the $X_j$ can be computed ahead of time, and error transmission propagation is not an issue, but in addition, as no encryption is required to find $X_{j+1}$ from $X_j$, the $O_j$ can be computed in parallel, vastly decreasing the time required to execute the process.