defend against timing attacks, AES should be implemented in a way that the execution time remains constant, irrespective of the cache architecture.

Other side channel attacks on AES target hardware implementations, such as those on a ***field-programmable gate array*** (FPGA). For example, ***fault attacks*** induce hardware error conditions during the execution of the algorithm and compare the resulting corrupted ciphertext with the correct ciphertext from a regular execution of the algorithm.

1.7 Modes of Operation

There are several ways to use a ***block cipher***, such as AES, that operate on fixed-length blocks. The different ways such an encryption algorithm can be used are known as its ***modes of operation***. In this section, we discuss several of the most commonly used modes of operation for block ciphers. The general scenario is that we have a sequence of blocks, B_1, B_2, B_3, and so on, to encrypt, all with the same key, K, using a block cipher algorithm, like AES.

Electronic Codebook (ECB) Mode

The simplest of encryption modes for a block cipher encrypts each block, B_i, independently. That is, this mode, which is known as ***electronic codebook mode*** (***ECB***) mode, involves encrypting the block, B_i, according to the following formula:

$$C_i = E_K(B_i),$$

were E_K denotes the block encryption algorithm using key K. Likewise, decryption is by the following formula:

$$B_i = D_K(C_i),$$

where D_K denotes the block decryption algorithm using key K.

This mode has the advantage of simplicity, of course. In addition, it can tolerate the loss of a block, such as might occur if the blocks are being sent as packets over a network. This resilience to block loss comes from the fact that decrypting the ciphertext for a block, B_i, does not depend in any way on the block, B_{i-1}.

The disadvantage of using this mode, however, is that, if our encryption algorithm is completely deterministic, like AES, so that each plaintext has a unique associated ciphertext, then the ECB mode may reveal patterns that might appear in the stream of blocks. In this case, identical blocks will have identical encryptions in ECB mode. For example, in a large image file, blocks of the image that are the same color, and are therefore identical, will be encrypted in the same way. This disadvantage of ECB mode allows an encryption of a sequence of blocks sometimes to reveal a surprising amount of information, as illustrated in Figure 6.

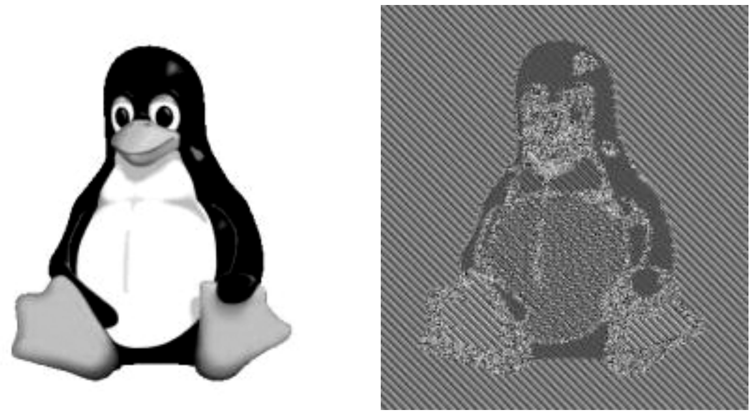

Figure 6: How ECB mode can leave identifiable patterns in a sequence of blocks: (a) An image of Tux the penguin, the Linux mascot. (b) An encryption of the Tux image using ECB mode. (The image in (a) is by Larry Ewing, lewing@isc.tamu.edu, using The Gimp; the image in (b) is by Dr. Juzam. Both are used with permission via attribution.) (a) Created by Larry Ewing (lewing@isc.tamu.edu) using The Gimp. Used with permission via attribution. (b) Created by Dr. Juzam. Used with permission via attribution.

Cipher-Block Chaining (CBC) Mode

An encryption mode that avoids the revelation of patterns in a sequence of blocks is the ***cipher-block chaining mode*** (***CBC***). In this mode of operation, the first plaintext block, B_1, is exclusive-ored with an ***initialization vector***, C_0, prior to being encrypted, and each subsequent plaintext block is exclusive-ored with the previous ciphertext block prior to being encrypted. That is, setting C_0 to the initialization vector, then

$$C_i = E_K(B_i \oplus C_{i-1}).$$

Decryption is handled in reverse,

$$B_i = D_K(C_i) \oplus C_{i-1},$$

where we use the same initialization vector, C_0, since exclusive-or is a self-inverting function.

This mode of operation has the advantage that if identical blocks appear at different places in the input sequence, then they are very likely to have different encryptions in the output sequence. So it is difficult to determine patterns in an encryption that is done using CBC mode, which corrects a disadvantage of ECB mode.

CBC mode does not allow the encryption of the blocks in a sequence to be done independently. That is, the sequence of blocks must be encrypted sequentially, with the encryption of block $i-1$ completing before the encryption of block i can begin.

Decryption, on the other hand, can proceed in parallel if all the ciphertext blocks are available. This asymmetry is due to the fact that both the encryption and decryption of block i uses the ciphertext block $i-1$. This block is available during encryption only through a sequential process. But all the encryptions are available for decryption; hence, the decryption can be done in parallel.

In addition, this property implies that the decryption process can tolerate the loss of a ciphertext block. For if block C_i is lost, it implies that decryption of blocks i and $i+1$ are lost. But decryption of block $i+2$ can still be done, since it relies only on C_{i+1} and C_{i+2}.

Cipher Feedback (CFB) Mode

The ***cipher feedback mode*** (***CFB***) for block encryption algorithms is similar to that of the CBC mode. Like the CBC, the encryption for block B_i involves the encryption, C_{i-1}, of the previous block. The encryption begins with an initialization vector, C_0. It computes the encryption of the ith block as

$$C_i = E_K(C_{i-1}) \oplus B_i.$$

That is, the ith block is encrypted by first encrypting the previous ciphertext block and then exclusive-oring that with the ith plaintext block. Decryption is done similarly, as follows:

$$B_i = E_K(C_{i-1}) \oplus C_i.$$

That is, decryption of the ith ciphertext block also involves the encryption of the $(i-1)$st ciphertext block. The decryption algorithm for the block cipher is actually never used in this mode. Depending on the details of the block cipher, this property could allow decryption to proceed faster by using the CFB mode than by using the CBC mode.

Output Feedback (OFB) Mode

In the ***output feedback mode*** (***OFB***), a sequence of blocks is encrypted much as in the one-time pad, but with a sequence of blocks that are generated with the block cipher. The encryption algorithm begins with an initialization vector, V_0. It then generates a sequence of vectors,

$$V_i = E_K(V_{i-1}).$$

Given this sequence of pad vectors, we perform block encryptions as follows:

$$C_i = V_i \oplus B_i.$$

Likewise, we perform block decryptions as follows:

$$B_i = V_i \oplus C_i.$$

Thus, this mode of operation can tolerate block losses, and it can be performed in parallel, both for encryption and decryption, provided the sequence of pad vectors has already been computed.

Counter (CTR) Mode

In ***counter mode*** (***CTR***), every step of encryption and decryption can be done in parallel. This mode is similar to the OFB in that we perform encryption through an exclusive-or with a generated pad. In fact, the method is essentially that mentioned in Section 1.4. We start with a random seed, s, and compute the ith offset vector according to the formula

$$V_i = E_K(s + i - 1),$$

so the first pad is an encryption of the seed, the second is an encryption of $s + 1$, the third is an encryption of $s + 2$, and so on. Encryption is performed as in the OFB mode, but with these generated vectors,

$$C_i = V_i \oplus B_i.$$

Likewise, we perform block decryptions as follows:

$$B_i = V_i \oplus C_i.$$

In this case, the generation of the pad vectors, as well as encryptions and decryptions, can all be done in parallel. This mode is also able to recover from dropped blocks.