CS381-Cryptography

Lecture 5: Block Ciphers

September 22, 2014

1 Overview

1.1 What is a Block Cipher?

Most (not all) modern symmetric encryption is built on *block ciphers*, which are algorithms for encrypting fixed-length blocks of data.
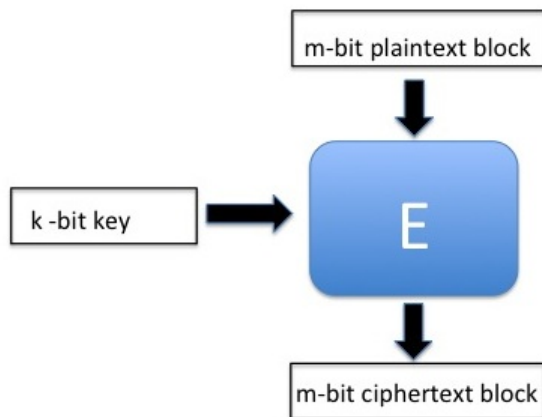

Figure 1: *Block cipher encryption of a single block of data*

Formally a block cipher is a pair of functions $E, D : \{0,1\}^k \times \{0,1\}^m \to \{0,1\}^m$. k is the key length, in bits, and m the *block size*. We require that for all $K \in \{0,1\}^k$, $P \in \{0,1\}^m$,

$$D(K, E(K, P)) = P.$$

The decryption and encryption functions for block ciphers need to be fast to compute, and easily implemented in hardware.

If you choose a key $K \in \{0, 1\}^k$, then the function

$$E_K : P \mapsto E_K(P)$$

is a permutation of $\{0, 1\}^m$. The idea is that if you select K at random, this should be for all practical purposes indistinguishable from a randomly-selected permutation of $\{0, 1\}^m$. (A randomly-selected permutation is like using the substitution cipher with a random permutation of the alphabet, although here the alphabet has 2^m letters rather than 26.) We do not have enough time or space in the universe to generate or specify a random permutation of $\{0, 1\}^m$ for, say, $m = 128$, which is a typical value. The trick is to use a short key to generate a permutation that 'looks' random.

1.2 Block Ciphers in Wide Use

They have names like DES, IDEA, Blowfish, Twofish, and AES. DES, the Data Encryption Standard, was introduced in the 1970's, and adopted as the official standard for encrypting non-classified government communications. DES has

$$k = 56, m = 64.$$

The fact that there are "only" $2^{56} \approx 7 \times 10^{16}$ keys proved to be a serious problem.

AES, the Advanced Enryption Standard, originally called the Rijndael Cipher, was adopted in 2000 as the replacement for DES. It can be used with several different key sizes, but typical values are

$$k = m = 128.$$

The textbook gives a lot of detail–more than we need to know–on the internal structure of both DES and AES.

2 Block Ciphers as Black Boxes

Here we discuss properties that are independent of the internal structure of the block cipher. We will assume that we have an 'ideal' block cipher: in other words, we will pretend that our key is a permutation of $\{0, 1\}^m$ selected uniformly at random from all $(2^m)!$ such permutations.

2.1 Security Against Brute-force Attack

2.1.1 Attack with two known plaintexts

If Eve intercepts a block obtained by encrypting with a block cipher (for example, a 16-byte AES block) she really can't decipher it, even with exhaustive search, if she knows nothing about the distribution of plaintext messages—we are no longer thinking of these things as being normal English.

What we'll show here is that if Eve knows just a few plaintext-ciphertext pairs, then exhaustive search is effective.

First imagine the following experiment. Suppose you have r randomly-selected permutations $\pi_1, \ldots, \pi_r$ of $\{1, \ldots, N\}$. Suppose you know a value of π_1, say $\pi_1(i) = j$. What is the probability that $\pi_k(i) = j$ for one of the other permutations π_k? We can obtain an upper bound on this probability by

$$\texttt{Prob}[\bigvee_{k=2}^{r} \pi_k(i) = j] \leq \sum_{k=2}^{r} \texttt{Prob}[\pi_k(i) = j]$$
$$= \frac{r-1}{n}.$$

Suppose instead that we know *two* values of π_1, say $\pi_1(i_1) = j_1$, $\pi_1(i_2) = j_2$. What is the probability that one of the other permutations π_k agrees at these two values?

$$\texttt{Prob}[\bigvee_{k=2}^{r} (\pi_k(i_1) = j_1) \wedge (\pi_k(i_2) = j_2)] \leq \sum_{k=2}^{r} \texttt{Prob}[(\pi_k(i_1) = j_1) \wedge (pi_k(i_2) = j_2)]$$
$$= \frac{r-1}{n(n-1)}.$$

What does this say about exhaustive-search attacks? For DES, if we know a single plaintext-ciphertext pair of blocks we can encrypt the plaintext under all possible keys until we find a block that matches the ciphertext. The probability that this can happen with more than one key is no more than

$$\frac{(r-1)}{n} \approx \frac{2^{56}}{2^{64}} = 2^{-8} = \frac{1}{256},$$

so this attack is likely to reveal the key. If we have two plaintext-ciphertext pairs, and search for a key that encrypts both plaintexts correctly, then the probability

that more than one key does this is no more than

$$\frac{r}{n(n-1)} \approx \frac{2^{56}}{2^{128}} = 2^{-72},$$

and for AES it is no more than

$$\frac{2^{128}}{2^{256}} = 2^{-128}.$$

So it is almost certain that there is a unique key that generates the pair of blocks.

This attack requires 2^{k+1} encryptions in the worst case, and 2^k encryptions on average. For 128-bit keys it is really impossible: we have only a negligible probability of finding the key as a result of any search that we could carry out in practice.

The 56-bit key for DES, on the other hand, is problematic. This weakness was pointed out when the standard was first proposed in the 1970's. The attack was successfully carried out in 1997, first by distributed computation by many computers working over the Internet, and then by specialized hardware that could test many keys in parallel. (The hardware was able to perform something like 80 billion encryptions per second!)

2.1.2 Multiple Encryption for Increased Key Size: Meet-in-the-middle attack

One might try to remedy the weak key length in DES by encrypting twice, using two different keys:

$$E'_{(K_1, K_2)}(P) = E_{K_2}(E_{K_1}(P)).$$

The hope is that in doing so we will have effectively doubled the key length, and thus squared the size of the key space. For DES this means a 112-bit key, which should be secure against brute-force attack.

You might well wonder at this point whether composing two short keys in this way is not equivalent to using a single short key, *i.e.,* is there a key K_3 such that

$$E(K_3, P) = E(K_1, E(K_2, P))?$$

It is known (see 'DES is not a group' in the textbook) that this does not occur in DES.

4

However, this double-encryption approach is vulnerable to a 'meet-in-the-middle attack', which renders it not much more secure than using a single key, provided one has a great deal of memory available. The idea is simple—for concreteness, we will assume a key size of $k = 56$ bits and a block size of $m = 64$ bits, which are the DES parameters.

Suppose we know two plaintext-ciphertext pairs $(P_1, C_1), (P_2, C_2)$, where

$$C_1 = E(K_2, E(K_1, P_1)), C_2 = E(K_2, E(K_1, P_2)).$$

The attack begins by encrypting the pair of blocks (P_1, P_2) under all 2^{56} keys, storing the resulting ciphertext pairs in a table, and sorting the table. Sorting a list of size n requires time proportional to $n \cdot \log_2 n$, so in this case we require 56×2^{56} steps for the sorting in addition to the 2×2^{56} encryptions of blocks. For the second step, we decrypt (C_1, C_2) under all 2^{56} keys and search for the resulting pair in the table. This requires $2 \cdot 2^{56}$ decryptions of blocks, plus 2^{56} searches of a sorted table, each of which uses about 56 probes. We know that there is at least one pair of keys (K_1, K_2) that results in a match. How likely is it that we will find a second match?

Our table contains no more than 2^{56} pairs of blocks. The probability that a randomly generated pair of blocks occurs in the table is $2^{56}/2^{128} = 2^{-72}$. Thus the probability that we will find more than one match in the decryption phase is no more than

$$2^{56}/2^{72} = 2^{-16} = 1/65536.$$

It is thus highly likely that the two known-plaintext pairs reveal the pair of keys that was used. In the unlikely event that we find more than one pair of keys that works, a third known-plaintext pair is virtually certain to resolve the issue.

The point of this attack is that it is only a few orders of magnitude more costly in terms of time that using a single 56-bit key, and thus the encryption method is far weaker than what one expects with a 112-bit key.

Triple encryption with DES is believed to be secure, and was adopted as a standard in the late 1990's.

2.2 Modes of Operation

You have a block cipher that encrypts a single block of data. How do you use it to encrypt multiple blocks?

2.2.1 The obvious (bad) answer: ECB mode

ECB stands for 'electronic code book'. It is the obvious thing, just encrypt each block independently. You've already seen this with the monoalphabetic substitution cipher. The key is a permutation of the 26 letters, a block is a single letter, and encryption is applied block by block. Much of the weakness follows from the fact that repeated blocks of plaintext show up as repeated blocks of ciphertext.
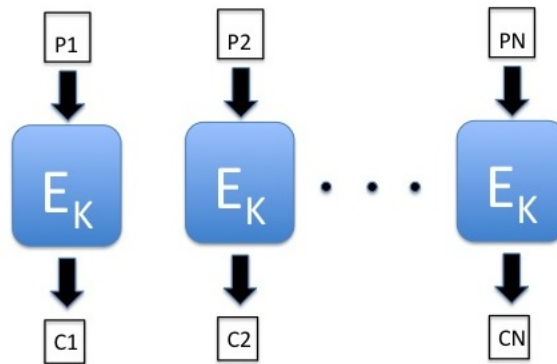

Figure 2: *ECB encryption of N blocks of data*

The same weakness is present in ECB with larger block sizes. If two blocks of ciphertext are identical, then the corresponding blocks of plaintext are also identical. Thus the ciphertext leaks information about the plaintext at the cost of very little computational effort.

You might think that in spite of this, a large block size would make repeated blocks in plaintext very rare, and that in any case, an occasional repeated block in ciphertext would not give away very much useful information. The following demonstration shows just how wrong that can be: The image on the left of Figure 3 was saved in a bit-mapped file format, so that apart from the short file header, every byte of the file is associated with one image pixel, with three bytes for each pixel. The file contents were encrypted by AES in ECB mode, and the original header restored so that the encrypted file could be displayed as a bit-mapped image.

Moral: Don't use ECB mode.

Figure 3: *ECB encryption of a bit-mapped image. Large regions of the image with a single color lead to a large number of repeated blocks.*

2.2.2 CBC Mode

CBC stands for *cipherblock chaining*. The ciphertext block generated in one step is mixed with the plaintext block of the next step prior to encryption. This leaves the problem of what to mix the first plaintext block with, and for this reason a special block called an *initialization vector* (IV) must be supplied. So we have (see Figure 4)

$$
\begin{aligned}
C_1 &= E_K(IV \oplus P_1) \\
C_{i+1} &= E_K(C_i \oplus P_{i+1}),
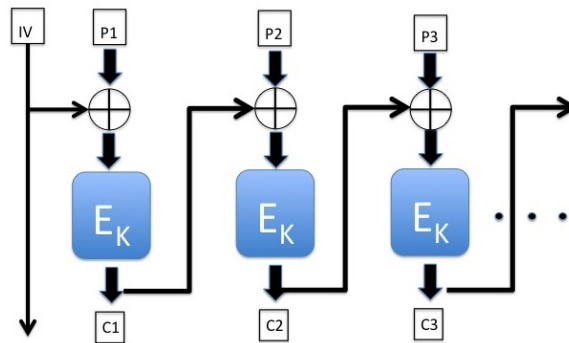\end{aligned}
$$

for $i \geq 1$.

Figure 4: *CBC encryption of multiple blocks of data*

Figure 5 shows the image experiment, this time conducted in CBC mode.

7

Figure 5: *CBC encryption of a bit-mapped image. The chaining process alters the input to each encryption block, and the resulting ciphertext looks completely random.*

To decrypt, Bob computes

$$\begin{aligned} P_1 &= IV \oplus D_K(C_1) \\ P_{i+1} &= C_i \oplus D_K(C_{i+1}), \end{aligned}$$

Bob must have IV available to obtain the first plaintext block, so the *IV must be sent unencrypted.* One consequence of this is that the ciphertext is one block longer than the plaintext message.

You might wonder why we bother with the IV at all, and not simply send $C_1 = E_K(P_1)$ as the first ciphertext block. The reason is that if we have several long messages encrypted with the same key, then a repetition of the first blocks in two of the messages will show up as a repetition of the first blocks in the received ciphertexts.

For the same reason, one should not use the same IV for two different messages. (For a related, but somewhat subtler reason, it should not be possible to predict the IV of the next message.) A sound practice is to choose a random IV for each message.

CBC has a nice error-recovery feature: Although changing one bit of a plaintext block completely changes all subsequent ciphertext blocks, a change of one bit of a ciphertext block because of a transmission error will only alter one bit in the next decrypted plaintext block, and have no effect on the subsequent ones.
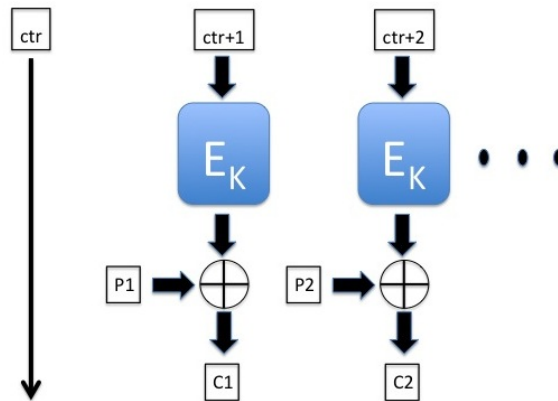
Figure 6: *Encryption of multiple blocks in CTR mode. Just as with the one-time pad, the plaintext is XORed with a keystream. In this case the keystream is generated using the block cipher applied to successive values of a counter block.*

2.2.3 CTR Mode

CTR stands for *counter*. An initial value ctr is chosen for a counter block. If we have N plaintext blocks to send, we encrypt the blocks $\text{ctr} + i$ for $i = 1, \ldots, N$, and XOR the results with the subsequent blocks of plaintext. (Figure 6). Note that we do not use the block cipher to directly encrypt the plaintext blocks; instead this is a stream cipher in which we use the block cipher to generate the keystream. As is the case with stream ciphers, decryption is the same as encryption, and you can not use the same keystream twice. For this reason, counter values should never repeat.

If the encryption function were truly random, then this method would be as secure as the one-time pad, provided that we never repeat a value of the counter fed to E_K. The probability that we repeat a value of the counter with two messages (or any small number of messages) is quite small, unless the messages are huge. For example, suppose we have two messages that are both one million blocks long, and that we use a block cipher with a 128-bit block size. We randomly initialize ctr for each of these messages. We will get a repeated value of the counter only if the initial values of ctr for the two messages are within two million of one another. The probability of this happening is no more than

$$\frac{2 \cdot 10^6}{2^{128}} < 3 \cdot 10^{-31}.$$

9

Observe that the large block size is crucial here. Security in practice depends on the extent to which the block cipher really does behave like a random function.

An advantage of CTR mode is that it is very fast and can be executed in parallel.

3 Internal Structure of Block Ciphers (especially AES)

Good block cipher design is an extremely difficult task. Both AES and triple-DES have been subject to intense scrutiny for many years, and no major security weakness has been discovered. If you need a block cipher, use one of these—don't try to design your own.

Still, it's worth talking about some of the design principles behind these block ciphers. Block ciphers typically have a *round structure:* In each round, the key, or a portion of the key, is used to derive a *round key.* Each round further alters the plaintext. Each individual round has a simple, easy-to-understand structure, and permits a rapid implementation. But the accumulated effect of all the rounds is supposed to make it effectively impossible to recover information about the plaintext block from the ciphertext block.

3.1 Substitution-Permutation Networks

What's inside a round? One general model (essentially followed in AES, but not in DES), is called a *substitution-permutation network.* This is illustrated in Figure 8. The input block of the round is first XORed with the round key. The resulting block (16 bytes in the example) is partitioned into 1-byte sub-blocks. The byte in each sub-block is replaced by a different byte, by means of a lookup table called an *S-box.* The bits of the outputs of the S-box are then permuted and the resulting output becomes the input to the next round. We call these three phases of a round AddKey, Sub, and Mix.

If we did not have the S-boxes, then regardless of the number of rounds, the ciphertext C would be $\pi(P) \oplus K'$, where K' is a block derived from the key, and $\pi(P)$ is some known permutation of the bits of P. This means that if we encrypted two different plaintext blocks P_1, P_2 to produce ciphertexts C_1, C_2, we would have $C_1 \oplus C_2 = \pi(P_1) \oplus \pi(P_2)$, revealing information about the plaintext with no knowledge of the key.

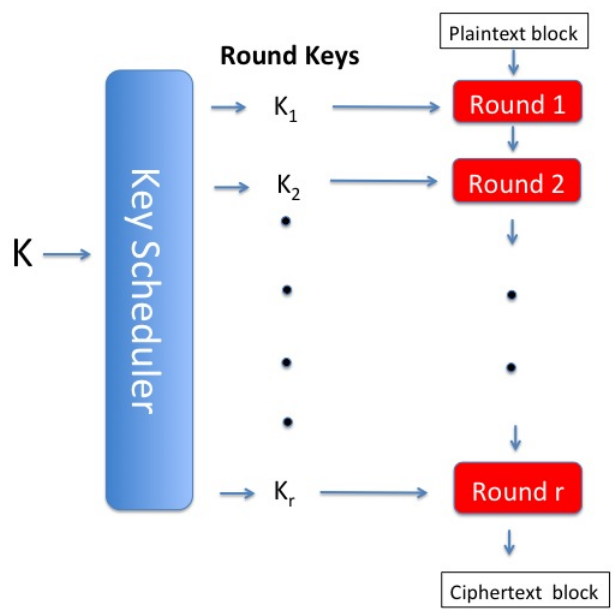For the same reason, the S-boxes must be a *nonlinear* functions of the input bytes.
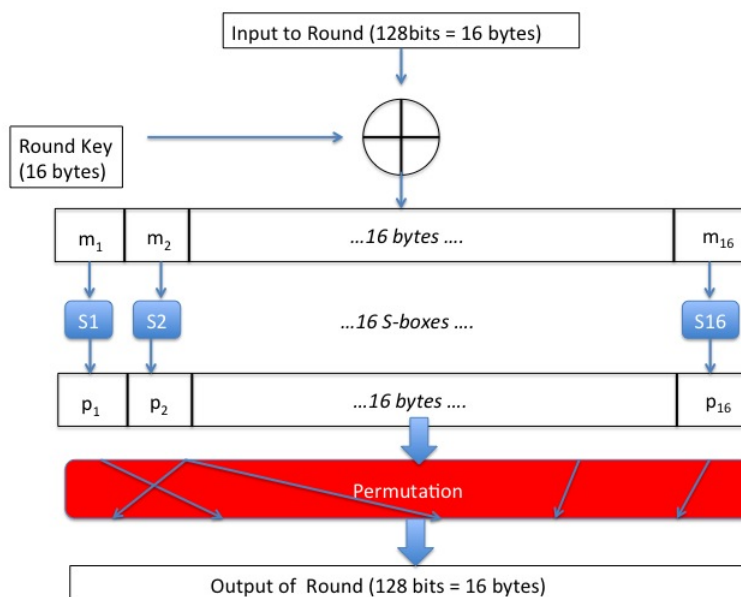
Figure 7: *Round structure of a block cipher.*

Figure 8: *One round of a substitution-permutation network. The S-boxes are lookup tables that provide a permutation of $\{0,1\}^8$. The mixing permutation permutes the bits of a 128-bit block.*

The S-boxes are designed so that changing any bit of the input byte changes at least two bits of the output byte. The mixing permutation is designed so that the output bits of a single S-box are sent to *different* sub-blocks. The effect of these two operations is the following: If we change one bit of plaintext, it will result in at least two bits difference in the output of one S-box in the first round. The mixing permutation will send these two bits to different S-boxes in the subsequent round, resulting in 4 bits of difference at the end of round 2, then 8 bits of difference at the end of round 3, etc. Of course, it cannot keep doubling like that, and there will be some collisions along the way, where output bits of two different subblocks are sent to the same subblock. But the net effect of all of this should be that changing one bit of a plaintext block will on average change about *half* the bits of the resulting ciphertext block. This is called the *avalanche effect.* As a result, local alterations to a block are diffused throughout the block. This also shows that the cipher requires a relatively large number of rounds to achieve this kind of thorough mixing.

You can see from even this brief description that the design of a round requires some intricate engineering, and is not a question of choosing something really confusing-looking at random.

AES is a variant of the substitution-permutation network idea. In AES, all the S-boxes are the same. The mixing permutation step in each round is divided into two phases: In the first phase (ShiftRows), the 16-byte state is represented as a 4×4 array of bytes, and each row of this matrix is cyclically shifted. In the second phase (MixColumns), the state is represented as a 32×4 matrix of bits, which is multiplied on the left by a particular 32×32 matrix of bits, with addition being done modulo 2. This is not precisely a permutation of the bits, so in this respect, the description differs from that of the generic substitution-permutation network. There are ten such rounds, with the last round being slightly different from the earlier ones.

An very detailed description is given in the textbook, along with some of the mathematics behind the construction of the various components.

3.2   Baby block cipher

Ignoring the admonitions not to design my own block cipher, I applied several of the design principles above to create a cipher with a 16-bit block size and key size. Obviously 16 bits is too small for any practical security, but it will enable you to perform several simple experiments on the homework assignment. No guarantees are made as to the soundness of the design, and I invite you to try other forms of
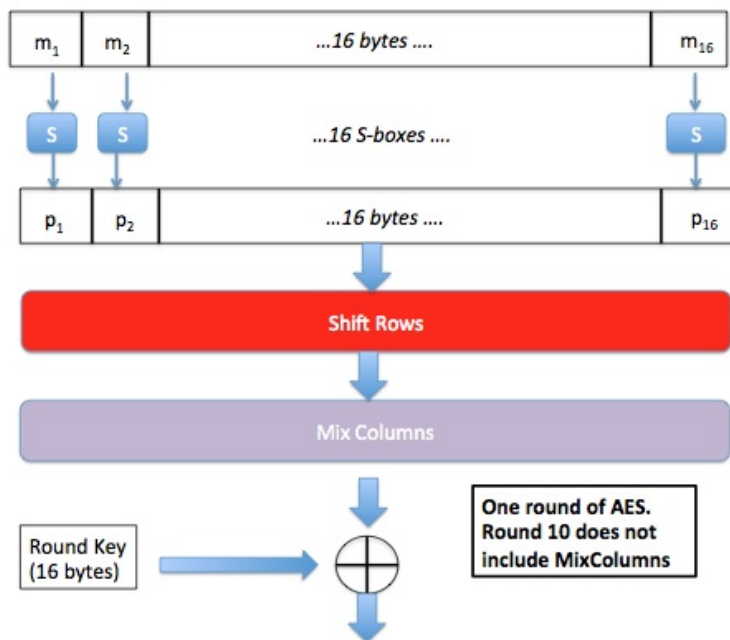
Figure 9: *One round of AES-128. All the S-boxes are the same, and the mixing permutation is replaced by a permutation of the bits followed by a linear transformation.*

cryptanalysis that will break this cipher faster than a brute-force attack.

The block cipher is constructed as a substitution permutation network. There are five rounds. The key-scheduling algorithm is as follows: Let's represent the original 16-bit key as 4 hex digits. For example, suppose it is

$$a30e.$$

This is the round key for the first round. The round keys for the next four rounds are obtained by cyclically permuting these hex digits:

$$ea30$$

$$0ea3$$

$$30ea$$

$$a30e$$

(In particular, the round keys for the first round and the last round are the same.)

Each round operates on a 16-bit *state.* At the start of the algorithm, the state is the plaintext block, and at the end, it is the ciphertext block.

The first phase of each round is the `AddKey` phase, where the round key is XORed with the state to produce the new state.

The second phase is the `Sub` phase: The state is split into four 4-bit chunks. Each chunk represents an integer in the range 0,..,15. It is replaced by another integer in the same range using the permutation.

$$[11, 13, 7, 12, 3, 6, 10, 1, 14, 0, 9, 8, 15, 4, 2, 5]$$

In other words, suppose the state is $287a$. The permutation replaces 2 by 7, 8 by 14 (e), 7 by 1, and a (10) by 9, so the new state is (in hex) 7e19. Observe that all four S-boxes are the same.

The final phase is the `Mix` phase, where the bits of the state are permuted according to the permutation

$$[5, 8, 0, 15, 11, 7, 2, 13, 14, 10, 1, 4, 12, 9, 6, 3]$$

Let's see what this does with our example above. The state at the end of the `Sub` phase of the round was $7e19 = 0111111000011001$ in binary. The new state will consist of bits 5, 8, 0, 15, etc. of this binary string, in other words.

$$1001101000111011 = 9a3b.$$

15

I have provided an implementation in Python. The default encryption function just takes the key and plaintext as arguments in the range 0,..,65535 and returns the ciphertext block as an integer in the same range. It's worth seeing what happens when you use a smaller number of rounds, so there is an optional keyword argument `numrounds`. There is another optional keyword argument `verbose` of boolean type, that lets you see how the state is transformed in each phase of each round. For example

```
encrypt(2345,24680,numrounds=2,verbose=True)
```

produces the following output:

```
plaintext
0110 0000 0110 1000
Round 1
Round Key 0000 1001 0010 1001
Add Key
0110 1001 0100 0001
Sub
1010 0000 0011 1101
Mix
0011 1011 0100 1000
Round 2
Round Key 1001 0000 1001 0010
Add Key
1010 1011 1101 1010
Sub
1001 1000 0100 1001
Mix
0011 0000 0001 1101
12317
```