Problems

12.1 In Figure 12.4, it is assumed that an array of 80 64-bit words is available to store the values of W_t, so that they can be precomputed at the beginning of the processing of a block. Now assume that space is at a premium. As an alternative, consider the use of a 16-word circular buffer that is initially loaded with W_0 through W_{15}. Design an algorithm that, for each step t, computes the required input value W_t.

12.2 For SHA-512, show the equations for the values of W_{16}, W_{17}, W_{18}, W_{19}.

12.3 Suppose $a_1\ a_2\ a_3\ a_4$ are the 4 bytes in a 32-bit word. Each a_i can be viewed as an integer in the range 0 to 255, represented in binary. In a big-endian architecture, this word represents the integer

$$a_1 2^{24} + a_2 2^{16} + a_3 2^8 + a_4$$

In a little-endian architecture, this word represents the integer

$$a_4 2^{24} + a_3 2^{16} + a_2 2^8 + a_1$$

a. Some hash functions, such as MD5, assume a little-endian architecture. It is important that the message digest be independent of the underlying architecture. Therefore, to perform the modulo 2 addition operation of MD5 or RIPEMD-160 on a big-endian architecture, an adjustment must be made. Suppose $X = x_1\ x_2\ x_3\ x_4$ and $Y = y_1\ y_2\ y_3\ y_4$. Show how the MD5 addition operation $(X + Y)$ would be carried out on a big-endian machine.

b. SHA assumes a big-endian architecture. Show how the operation $(X + Y)$ for SHA would be carried out on a little-endian machine.

12.4 This problem introduces a hash function similar in spirit to SHA that operates on letters instead of binary data. It is called the *toy tetragraph hash* (tth).[5] Given a message consisting of a sequence of letters, tth produces a hash value consisting of four letters. First, tth divides the message into blocks of 16 letters, ignoring spaces, punctuation, and capitalization. If the message length is not divisible by 16, it is padded out with nulls. A four-number running total is maintained that starts out with the value (0, 0, 0, 0); this is input to the compression function for processing the first block. The compression function consists of two rounds. **Round 1:** Get the next block of text and arrange it as a row-wise 4 × 4 block of text and convert it to numbers (A = 0, B = 1, etc.). For example, for the block ABCDEFGHIJKLMNOP, we have:

A	B	C	D
E	F	G	H
I	J	K	L
M	N	O	P

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

[5]I thank William K. Mason, of the magazine staff of *The Cryptogram*, for providing this example.

Then, add each column mod 26 and add the result to the running total, mod 26. In this example, the running total is (24, 2, 6, 10). **Round 2:** Using the matrix from round 1, rotate the first row left by 1, second row left by 2, third row left by 3 , and reverse the order of the fourth row. In our example:

B	C	D	A
G	H	E	F
L	I	J	K
P	O	N	M

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

Now, add each column mod 26 and add the result to the running total. The new running total is (5, 7, 9, 11). This running total is now the input into the first round of the compression function for the next block of text. After the final block is processed, convert the final running total to letters. For example, if the message is ABCDEFGHIJKLMNOP, then the hash is FHJL.

a. Draw figures comparable to Figures 12.1 and 12.2 to depict the overall tth logic and the compression function logic.

b. Calculate the hash function for the 48-letter message "I leave twenty million dollars to my friendly cousin Bill."

c. To demonstrate the weakness of tth, find a 48-letter block that produces the same hash as that just derived. *Hint:* Use lots of A's.

12.5 Develop a table similar to Table 4.8 for GF(2^8) with $m(x) = x^8 + x^4 + x^3 + x^2 + 1$.

12.6 Show the E and E^{-1} mini-boxes in Table 12.3 in the traditional S-box square matrix format, such as that of Table 5.4.

12.7 Verify that Figure 12.9 is a valid implementation of the S-box shown in Table 12.3a. Do this by showing the calculations involved for three input values: 00, 55, 1E.

12.8 Provide a Boolean expression that defines the S-box functionality of Figure 12.9.

12.9 Whirlpool makes use of the construction $H_i = E(H_{i-1}, M_i) \oplus H_{i-1} \oplus M_i$. Another construction that was shown by Preneel to be secure is $H_i = E(H_{i-1}, M_i) \oplus M_i$. Now notice that the key schedule for Whirlpool resembles encryption of the cipher key under a pseudo-key defined by the round constants, so that the core of the hashing process could be formally viewed as two interacting encryption lines. Consider the encryption $E(H_{i-1}, M_i)$. We could write the final round key for this block as $K_{10} = E(RC, H_{i-1})$. Now show that the two hash constructions are essentially equivalent because of the way that the key schedule is defined.

12.10 At the beginning of Section 12.4, it was noted that given the CBC MAC of a one-block message X, say $T = \text{MAC}(K, X)$, the adversary immediately knows the CBC MAC for the two-block message $X \| (X \oplus T)$ since this is once again T. Justify this statement.

12.11 In this problem, we demonstrate that for CMAC, a variant that XORs the second key after applying the final encryption doesn't work. Let us consider this for the case of the message being an integer multiple of the block size. Then the variant can be expressed as $\text{VMAC}(K, M) = \text{CBC}(K, M) \oplus K_1$. Now suppose an adversary is able to ask for the MACs of three messages: the message $\mathbf{0} = 0^n$, where n is the cipher block size; the message $\mathbf{1} = 1^n$, and the message $\mathbf{1}\|\mathbf{0}$. As a result of these three queries the adversary gets $T_0 = \text{CBC}(K, \mathbf{0}) \oplus K_1$; $T_1 = \text{CBC}(K, \mathbf{1}) \oplus K_1$, and $T_2 = \text{CBC}(K, [\text{CBC}(K, \mathbf{1})]) \oplus K_1$. Show that the adversary can compute the correct MAC for the (unqueried) message $\mathbf{0}\|(T_0 \oplus T_1)$.

12.12 In the discussion of subkey generation in CMAC, it states that the block cipher is applied to the block that consists entirely of 0 bits. The first subkey is derived from the resulting string by a left shift of one bit, and, conditionally, by XORing a constant that depends on the block size. The second subkey is derived in the same manner from the first subkey.

a. What constants are needed for block sizes of 64 and 128 bits?

b. Explain how the left shift and XOR accomplishes the desired result.