CHAPTER 11

Message Integrity and Message Authentication

(Solution to Odd-Numbered Problems)

Review Questions

1. Message integrity guarantees that the message has not been changed. Message authentication guarantees that the sender of the message is authentic.

3. The *second preimage resistance* ensures that a message cannot easily be forged. In other words, given a specific message and its digest, it is impossible to create another message with the same digest.

5. The Random Oracle Model is an ideal mathematical model for a hash function. A function based on this model behaves completely randomly. The digest can be thought of as a random variable uniformly distributed between 0 and $N - 1$ in which $N = 2^n$. Attacks on hash functions can be analyzed working with this random variable.

7. The following briefly states the four birthday problems:

- **Problem 1:** What is the minimum number, *k*, of students in a classroom such that it is *likely* that at least one student has a predefined birthday?
- **Problem 2:** What is the minimum number, *k*, of students in a classroom such that it is *likely* that at least one student has the same birthday as the student selected by the professor? This problem can be generalized as follows.
- **Problem 3:** What is the minimum number, *k*, of students in a classroom such that it is *likely* that at least two students have the same birthday?
- **Problem 4:** We have two classes, each with *k* students. What is the minimum value of *k* so that it is *likely* that at least one student from the first classroom has the same birthday as a student from the second classroom?

9. A *modification detection code* (*MDC*) is a message digest that can prove the integrity of the message. A *message authentication code* (*MAC*) ensures the integrity of the message and the data origin authentication. The difference between an MDC and a MAC is that the second includes a secret between Alice and Bob.

Exercises

11. Imagine Alice uses the oracle. She give her message M to send to the oracle and obtain the digest d. Alice then sends the message and digest to Bob. To verify that the message has come from Alice, Bob needs to give the received message M to the same oracle and obtain the digest d_1. If d_1 is not equal (all the time with d), Bob cannot verify that the message has been sent by Alice. This means that the oracle needs to record the message obtained from Alice and the digest given to her to be able to create the same digest when Bob ask for calculation of the digest.

13. This the first birthday problem, in which "on average" can be interpreted as "with probability P = 1/2 and $N = 30$". We have

$$k \approx 0.69 \times N \approx 21$$

15. This the first birthday problem, in which "on average" can be interpreted as "with probability P = 1/2 and $N = 2007 - 1950 = 57$". We have

$$k \approx 0.69 \times N \approx 40.$$

17. We solve each case separately.

a. This the third birthday problem with N = 365 and $k = 6$. We let the probability of two family member having the same birthday be P and no family member having the same birthday be Q. According to Table 11.3 in the text, we have

$$\text{P} \approx 1 - e^{-k(k-1)/2N} \approx 0.04 \qquad \text{Q} = 1 - \text{P} \approx 0.96$$

b. This the third birthday problem with N = 30 and $k = 6$. We let the probability of two family member born on the same day of the month be P and no family member born on the same day of the month be Q. According to Table 11.3 in the text, we have

$$\text{P} \approx 1 - e^{-k(k-1)/2N} \approx 0.04 \qquad \text{Q} = 1 - \text{P} \approx 0.96$$

c. This the first birthday problem with N = 30 and $k = 6$. We can solve this problem directly or using the approximation given in Table 11.3 in the text. In the direct solution, the probability of one particular member is born on the first day of the month is 1/30; the probability that any member of the family is born at the first day of the month is 6/30.

$$\text{P} = 6 \times (1/30) = 0.20 \qquad \text{P} \approx 1 - e^{-k/N} \approx 0.19$$

d. This the same as the third birthday problem because "three" can be thought of "at least two". Using the approximation in the third birthday problem we can have

$$P \approx 1 - e^{-k(k-1)/2N} \approx 0.35$$

19. This the third birthday problem with N = 9999 and k = 100. We let the probability of two students having the same four digits in their social security number be P. According to Table 11.3 in the text, we have

$$P \approx 1 - e^{-k(k-1)/2N} \approx 0.74$$

21. The pigeons can be distributed in the pigeonholes in any order. The principle does not define any order.

23. The probability of failure, Q, with a list of digest $k = 1.18 \times 2^{n/2}$, is 50 percent. This means that if Eve needs to be sure that she is successful, she needs to repeat the algorithm m times so that $Q = (1/2)^m = 0$. Solving this equation, gives us m = infinity, which means that Eve can never be hundred present sure that she can find the answer. However, if Eve repeats the algorithm many time, the probability of failure reduces tremendously. For example, if she repeat the algorithm 1000 times, $Q = 1 / 2^{1024}$, which is a very small number.

25. Although the problem does not mention the substitution process, we substitute the number x with the $(x + 1)$th prime number (the first prime number is 2). The following shows how the digest is calculated (using the table of prime numbers in Appendix H. The method is very insecure because the possible number of digests N = 100.

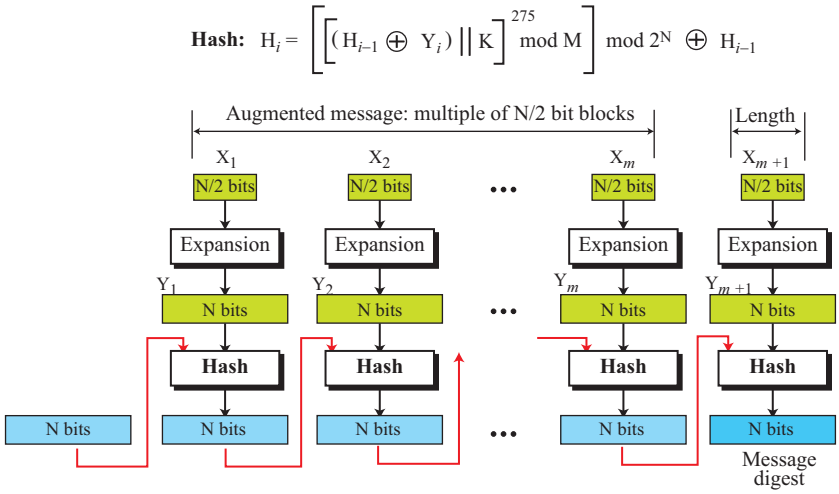
Initial value of the digest:			d = 00
Character: H (07)	→	8th prime: 19	d = (00 + 19) mod 100 = 19
Character: E (04)	→	5th prime: 11	d = (19 + 11) mod 100 = 30
Character: H (11)	→	12th prime: 37	d = (30 + 37) mod 100 = 67
Character: H (11)	→	12th prime: 37	d = (67 + 37) mod 100 = 04
Character: H (14)	→	15th prime: 47	d = (04 + 47) mod 100 = 51

27. Figure S11.27 shows the diagram for this hash algorithm. We assume that the message is already augmented and the last block (length block) is already added. We have $m + 1$ blocks, each of N/2 bits. Note that << means shift left and >> means shift right. The || operator means the OR operation.

```
MASH (x[0 ... m + 1], K, H_init , p, q)
{
        H[0] ← H_init
        M ← p × q
        for (i = 1 to m + 1)
        {
                if (i < m + 1)
                        Y[i] ← Expand (x[i], (1111)_2)
                else
```

```
                    Y[i] ← Expand (x[i], (1010)2)
                T[i] ← ((H[i − 1] ⊕ Y[i]) || K) 275 mod M
                G[i] ← T[i] mod 2N
                H[i] ← H[i − 1] ⊕ G[i]
        }
        return H[m + 1]
}
Expand (x, const)
{
        u ← (const << 4)                    // u is a const with four 0's
        y ← (000… 0)                        // y is made of N zeros
        bytecount ← (N/ 2) /4
        for (i = 1 to bytecount)
        {
                t ← x || (F)16              // extract the four rightmost bits
                u ← u || t
                y ← y || u << (i −1) × 8
                x ← x >> 4
        }
        return y
}
```

Figure S11.27 *Diagram for Exercise 27*



29. Given the value of P (less than 1), the following algorithm finds the value of *k* for the second general birthday problem:

```
BirthdaySecond (P, N)
{
        k ← 1
        while (true)
        {
                Q ← (1 – 1 / N) k –1
                if ((1 – Q) = P)
                        return k
                k ← k + 1
        }
}
```

31. Given the value of P (less than 1) and N, the following algorithm finds the value of k for the fourth general birthday problem:

```
BirthdayFourth (P, N)
{
        k ← 1
        while (true)
        {
                Q ← (1 – 1 / N) k × k
                if ((1 – Q) = P)
                        return k
                k ← k + 1
        }
}
```

33. Given the value of n, N, k, K (key), and a message of N blocks, the following algorithm calculates the CMAC. Note that we have not shown how to calculate the value of k, but it is not difficult to write another algorithm to do so.

```
CMAC (n, N, k, K, M[1 … N])
{
        C[1] ← EK (M[1])
        i ← 2
        while (i < N)
        {
                C[i] ← EK (C[i – 1] ⊕ M[i])
                i ← i + 1
        }
        C[N] ← EK (C[N – 1] ⊕ M[N] ⊕ k)
        h ← SelectLeft (n, C[N])
        return h
}
```