14 Identification

CHAPTER

In the previous chapters, two important basic mechanisms have been explained: encryption and digital signatures. In this chapter we describe a third basic technique: identification.

First, we present two examples for situations in which identification is necessary.

Example 14.0.5
Using Internet banking, Alice wants to find out how much money is left in her account. She must identify herself to the bank in order to prove that she is entitled to obtain the information.

Example 14.0.6
Bob works in a university where he uses a Unix workstation. When he comes to work, he identifies himself to his workstation in order to get access. The computer verifies Bob's identity and checks whether Bob is a legal user. If he is, access is granted to Bob. Typically, Bob proves his identity by presenting a secret password. This method of identification is not totally secure and will be discussed in Section 14.1.

Identification is required in many applications. Typically, the goal of an identification procedure is access control. Methods that permit identification are called *identification protocols*.

In an identification protocol, the *prover*, Bob, proves to the *verifier*, Alice, that it is really Bob who is communicating with Alice. Identification is a real-time problem.

In this chapter, we describe different identification protocols.

14.1 Passwords

Access to Unix or Windows NT systems is typically controlled by password systems. Each user picks his individual and secret password w. The computer stores the image $f(w)$ of the password w under a one-way function f. If the user wants access to his computer, he enters his name and password w. The computer determines $f(w)$ and compares this value with the stored value. If they are identical, then access is granted. Otherwise, the user is rejected.

Passwords are also used to control access to World Wide Web pages or to files that contain private encryption or signature keys.

The password file does not need to be kept secret since it contains only the images $f(w)$ of the passwords w and f is a one-way function. Nevertheless, password identification systems are not very secure.

A user must memorize his or her password. Therefore, many users choose the first name of their spouse or children as their password. An attacker can mount a dictionary attack. For all words w in a dictionary, he computes $f(w)$ and compares the result with the entries in the password file. If he finds an entry of the password file, he has determined the corresponding password. It is, therefore, recommended to use symbols such as \$ or # in the the passwords. Then dictionary attacks are impossible, but it is also harder to memorize the passwords. It is also possible to store the password on a smart card. Instead of typing in his password, the prover inserts his smart card into the smart card reader. The verifier reads the password from the smart card. There is no need for the user to memorize or even know the password. On the contrary, if the user does not know his password he cannot give it away.

An attacker can also tap the connection between the prover and the verifier and can learn the password. This is particularly successful if there is a great distance between the prover and the verifier; for example, if a password system is used to protect World Wide Web access. Note that the use of smart cards does not prevent this attack.

Finally, the attacker can also replace an entry $f(w)$ in the password file with the image $f(v)$ of his own password v. Then, using the password v, he can get access. Therefore, the password file must be write protected.

14.2 One-Time Passwords

Using passwords is dangerous because an attacker can learn the passwords by tapping the connection between the prover and the verifier. With one-time passwords, this attack does not work. One-time passwords are used for one identification. For the next identification, a new one-time password is used.

A simple way of implementing one-time passwords is the following. The verifier has a list $f(w_1), f(w_2), \ldots, f(w_n)$ of images of passwords $w_1, \ldots, w_n$. The prover knows this list of passwords and uses its elements for the identifications. Since the prover must store all passwords in advance, an attacker could learn some or all of them.

It is also possible that the prover and the verifier share a secret function f of an initial string w. Then the one-time passwords are $w_i = f^i(w)$, $i \geq 0$. The prover can put the current password w_i and the one-way function f on a smart card. He does not need a large password file.

14.3 Challenge-Response Identification

Password identification system protocols have the disadvantage that an attacker can learn passwords long before the actual identification. This is even true for one-time password systems.

Challenge response identification systems do not have this problem. Alice wants to identify herself to Bob in a challenge response system. Bob asks a question, the *challenge*. Alice computes the *response* using her secret key and sends it to Bob. Bob verifies the response using the same secret key or the corresponding public key.

14.3.1 Symmetric systems

We describe a simple challenge response identification system which uses a symmetric cryptosystem. We assume that the encryption key and the corresponding decryption key are the same. Alice and Bob share a secret key k. Alice wants to identify herself to Bob. Bob sends a random number r to Alice. Alice encrypts this random number by computing $c = E_k(r)$ and sends the ciphertext c to Bob. Bob decrypts the ciphertext; that is, he computes $r' = D_k(c)$ and compares the result with his chosen random number r. If $r = r'$, then he accepts the proof of identity; otherwise he rejects it.

This protocol proves that Alice knows the secret key at the time of the identification. It is not possible for Bob or an attacker to compute or obtain the correct response in advance. But since the verifier, Bob, also knows Alice's secret key, this key cannot be used for identification with another verifier since Bob can then pretend that he is Alice.

14.3.2 Public-key systems

Challenge response systems can also be based on public-key signature schemes. If Alice wants to identify herself, she obtains a random number from Bob and signs this random number with her private key. Bob verifies the signature, thereby verifying the identity of Alice.

In this protocol, Bob cannot pretend that he is Alice. He only knows Alice's public key. But it is necessary that Bob obtains the authentic public key of Alice. If the attacker, Oscar, can replace Alice's public key with his own, then he can convince Bob that he is Alice.

14.3.3 Zero-knowledge proofs

In a challenge response protocol, the prover proves that he knows a secret. If a symmetric cryptosystem is used, then the verifier also knows the secret. If a public-key signature system is used, then the verifier does not know the secret.

We now describe a *zero-knowledge* identification protocol. Again, the prover proves the knowledge of a secret, which the verifier does not know. During the protocol, the verifier learns nothing but the fact that the prover knows the secret. He gets no additional information about the secret. We say that the protocol has the *zero-knowledge property*.

The protocol that we describe is the *Fiat-Shamir identification protocol*. As in the RSA scheme, the prover, Alice, chooses two large random primes p and q. Then she chooses a random number s from $\{1, \ldots, n-1\}$ and computes $v = s^2 \bmod n$. Bob's public key is (v, n). His secret key is s, a square root of v mod n.

In a zero-knowledge protocol the prover Bob proves to the verifier Alice that he knows a square root s of v mod n. That protocol works as follows.

1. Commitment: Bob chooses $r \in \{1, 2, \ldots, n-1\}$ randomly with the uniform distribution and computes $x = r^2 \bmod n$. Bob sends x to the verifier Alice.
2. Challenge: Alice chooses $e \in \{0, 1\}$ randomly with the uniform distribution and sends it to Bob.
3. Response for $e = 0$: Upon receiving $e = 0$, Bob sends the random number r to Alice. Alice verifies that $r^2 \equiv x \bmod n$.
4. Response for $e = 1$: Upon receiving $e = 1$, Bob sends the number $y = rs \bmod n$ to Alice. Alice verifies that $y^2 \equiv xv \bmod n$.

Example 14.3.1
Let $n = 391 = 17 * 23$. Bob's secret key is $s = 123$. His public key is $(271, 391)$. In the identification protocol, Bob proves that he knows a square root of v mod n.

1. Commitment: Bob chooses the random number $r = 271$ and computes $x = r^2 \bmod n = 324$. He sends the result x to the verifier Alice.

2. Challenge: Alice chooses the random number $e = 1$ and sends it to Bob.
3. Response: Bob sends $y = rs \bmod n = 98$ to Alice.
4. Verification: Alice accepts since $220 = y^2 \equiv vx \bmod n$.

We analyze the protocol.

If Alice knows the secret, the square root s of v mod n, then she can answer both possible questions in the protocol correctly. We say that the protocol is *complete*.

If the attacker, Oscar, can compute a square root of v mod n, then he can also factor n. This was shown in Section 8.4.5. Because factoring integers is considered to be difficult, Alice's secret is secure.

But what happens if Oscar tries to impersonate Alice even though he does not know the secret? Then he cannot answer both possible questions correctly, as we will show now. Suppose that Oscar knows r and rs mod n. Then he can compute $s = rss^{-1} \bmod n$, so he knows Alice's secret. Because he does not know s, Oscar can only answer one question correctly. In fact, in order to be able to answer the challenge correctly, for a fixed e he chooses the commitment x as $x = y^2 r^{-e} \bmod n$ for some y. Nevertheless, Oscar will not be able to give the correct response for the other e. Therefore, the verifier will detect with probability $1/2$ that Oscar is not Alice. After k iterations of the protocol, the verifier will detect the fraud with probability $1 - 1/2^k$. This probability can be made arbitrarily close to 1. We say that the protocol is *correct*.

Finally we show that the Fiat-Shamir protocol has the *zero-knowledge property*. We first explain what this means.

Suppose that Alice wants to cheat. Which goals can she have? For example, she could try to obtain Bob's secret, a square root of v modulo n. Assume that for sufficiently large n such a square root cannot be computed from n and v. But perhaps, Alice could use the information that she obtains in the protocol to find the secret. Also, Alice can try to convince others that she is Bob even if she does not know Bob's secret. Many other goals for an attack are possible and it is impossible to predict them all. But we would like to be sure that Alice cannot use the information that she obtains in the protocol to mount any attack that she cannot mount without that information. This is guaranteed by the zero knowledge property. The protocol has that

property, if Alice can simulate the protocol without the knowledge of Bob's secret in such a way that the distribution on the messages in the simulated protocol cannot be distinguished from the distribution on the messages in the real protocol or in modifications in which the attacker Alice deviates from the protocol in order to obtain even more information. For example, it is possible that Alice chooses the challenge e not randomly but as a function of the commitment x. If Alice can simulate any such protocol variant without the knowledge of Bob's secret then she cannot learn anything relevant from the protocol. The protocol is zero-knowledge.

Let us analyze the message distributions and then explain how the simulation works.

In each iteration of the protocol a message triplet (x, e, y) is generated. Here x is a square modulo n in $\{0, \ldots, n-1\}$ that is chosen randomly with the uniform distribution. The challenge e is chosen according to some distribution that is selected by Alice. Also, y is a random square root of xs^e mod n in $\{0, \ldots, n-1\}$. Triplets (x, e, y) with the same distribution can be generated as follows. The simulator selects a random f in $\{0, 1\}$ with the uniform distribution. The simulator calculates $x = y^2 s^{-f} \bmod n$ where s^{-f} is the inverse of s^f modulo n. Finally, the simulator chooses a random number e in $\{0, 1\}$ with Alice's distribution. If e and f are equal, then the simulator outputs (x, e, y). Then we have $x = y^2 s^{-f} \bmod n$. Otherwise, the simulator deletes the triplet (x, e, y) and outputs nothing.

We analyze the distribution that is generated by the simulator. The probability for the simulator to output something is $1/2$. So the simulator efficiently produces an output distribution. We show that the distributions on the message triplets of the simulator and the real protocol are the same. As in the real protocol, x is a random square modulo n in $\{0, \ldots, n-1\}$, the challenge e is selected as in the real protocol and the response y is a random square root of xs^e modulo n in $\{0, 1, \ldots, n-1\}$.

For more details concerning zero knowledge, we refer the reader to [31].

14.4 Exercises

Exercise 14.4.1
Let p be a prime number, g a primitive root mod p, $a \in \{0, 1, \ldots, p-2\}$, and $A = g^a \bmod p$. Describe a zero-knowledge proof for the knowledge of the discrete logarithm a of A mod p to the base g.

Exercise 14.4.2
In the Fiat-Shamir scheme, let $n = 143$, $v = 82$, $x = 53$, and $e = 1$. Determine a valid response that proves the knowledge of a square root of v mod n.

Exercise 14.4.3 (Feige-Fiat-Shamir protocol)
The Feige-Fiat-Shamir protocol is a modification of the Fiat-Shamir protocol. In this protocol, a cheating verifier is discovered with much higher probability. A simplified version works as follows. Alice uses an RSA modulus n. She chooses random numbers $s_1, \ldots, s_k$ in $\{1, \ldots, n-1\}^k$ and computes $v_i = s_i^2 \bmod n$, $1 \le i \le k$. Her public key is $(n, v_1, \ldots, v_k)$. Her secret key is $(s_1, \ldots, s_k)$. To convince Bob of her identity, she chooses a random number $r \in \{1, \ldots, n-1\}$, computes the commitment $x = r^2 \bmod n$, and sends it to Bob. Bob chooses a random challenge $(e_1, \ldots, e_k) \in \{0, 1\}^k$ and sends it to Alice. Alice sends the response $y = r \prod_{i=1}^{k} s_i^{e_i}$ to Bob. Bob verifies that $y^2 \equiv x \prod_{i=1}^{k} v_i^{e_i} \bmod p$. Determine the probability of success for a cheating verifier in one round.

Exercise 14.4.4
Modify the scheme from Exercise 14.4.3 such that its security is based on computing discrete logarithms.

Exercise 14.4.5 (Signatures from identification)
Find a signature scheme based on the protocol from Exercise 14.4.3. The idea is to replace the challenge by the hash value $h(x \circ m)$, where m is the message to be signed and x is the commitment.

15 Secret Sharing

CHAPTER

In public-key infrastructures it is frequently useful to be able to reconstruct private keys. For example, if a user has lost his smartcard that contains his private decryption key, then he cannot decrypt any encrypted file on his computer anymore. So those encrypted files are then inaccessible for the user unless it is possible to reconstruct the decryption key. However, for security reasons it may be important that the key cannot be reconstructed by a single person. That person could abuse the knowledge of the private key. It is more secure if a group of people has to be involved in the reconstruction. In this chapter we describe *secret sharing*, a protocol that can be used to solve this problem.

15.1 The Principle

We explain what secret sharing does. Let n and t be positive integers. In an (n, t) secret sharing protocol the secret is distributed among n shareholders. Each of them has his share of the secret. If t of the shareholders collaborate, then they can reconstruct the se-