Chapter 9

Simple Authentication Protocols

"I quite agree with you," said the Duchess; "and the moral of that is—
'Be what you would seem to be'—or,
if you'd like it put more simply—'Never imagine yourself not to be
otherwise than what it might appear to others that what you were
or might have been was not otherwise than what you
had been would have appeared to them to be otherwise.' "
— Lewis Carroll, *Alice in Wonderland*

Seek simplicity, and distrust it.
— Alfred North Whitehead

9.1 Introduction

Protocols are the rules that are followed in some particular interaction. For example, there is a protocol that you follow if you want to ask a question in class, and it goes something like this:

1. You raise your hand.
2. The teacher calls on you.
3. You ask your question.
4. The teacher says, "I don't know."[1]

There are a vast number of human protocols, some of which can be very intricate, with numerous special cases to consider.

[1] Well, at least that's the way it works in your oblivious author's classes.

In the context of networking, protocols are the rules followed in networked communication systems. Examples of formal networking protocols include HTTP, FTP, TCP, UDP, PPP, and there are many, many more. In fact, the study of networks is largely the study of networking protocols.

Security protocols are the communication rules followed in security applications. In Chapter 10 we'll look closely at several real-world security protocols including SSH, SSL, IPSec, WEP, and Kerberos. In this chapter, we'll consider simplified authentication protocols so that we can better understand the fundamental security issues involved in the design of such protocols. If you want to delve a little deeper than the material presented in this chapter, the paper [3] has a discussion of some security protocol design principles.

In Chapter 7, we discussed methods that are used, primarily, to authenticate humans to a machines. In this chapter, we'll discuss authentication protocols. Although it might seem that these two authentication topics must be closely related, in fact, they are almost completely different. Here, we'll deal with the security issues related to the messages that are sent over a network to authenticate the participants. We'll see examples of well-known types of attacks on protocols and we'll show how to prevent these attacks. Note that our examples and analysis are informal and intuitive. The advantage of this approach is that we can cover all of the basic concepts quickly and with minimal background, but the price we pay is that some rigor is sacrificed.

Protocols can be subtle—often, a seemingly innocuous change makes a significant difference. Security protocols are particularly subtle, since the attacker can actively intervene in the process in a variety of ways. As an indication of the challenges inherent in security protocols, many well-known security protocols—including WEP, GSM, and even IPSec—have significant security issues. And even if the protocol itself is not flawed, a particular implementation can be.

Obviously, a security protocol must meet some specified security requirements. But we also want protocols to be efficient, both in computational cost and bandwidth usage. An ideal security protocol would not be too fragile, that is, the protocol would function correctly even when an attacker actively tries to break it. In addition, a security protocol should continue to work even if the environment in which it's deployed changes. Of course, it's impossible to protect against every possible eventuality, but protocol developers can try to anticipate likely changes in the environment and build in protections. Some of the most serious security challenges today are due to the fact that protocols are being used in environments for which they were not initially designed. For example, many Internet protocols were designed for a friendly, academic environment, which is about as far from the reality of the modern Internet as possible. Ease of use and ease of implementation are also desirable features of security protocols. Obviously, it's going to be difficult to design an ideal protocol.

9.2 Simple Security Protocols

The first security protocol that we consider is a protocol that could be used for entry into a secure facility, such as the National Security Agency. Employees are given a badge that they must wear at all times when in the secure facility. To enter the building, the badge is inserted into a card reader and the employee must provide a PIN number. The secure entry protocol can be described as follows.

1. Insert badge into reader.
2. Enter PIN.
3. Is the PIN correct?
   - Yes: Enter the building.
   - No: Get shot by a security guard.[2]

When you withdraw money from an ATM machine, the protocol is virtually identical to the secure entry protocol above, but without the violent ending:

1. Insert ATM card into reader
2. Enter PIN
3. Is the PIN correct?
   - Yes: Conduct your transactions
   - No: Machine eats your ATM card

The military has a need for many specialized security protocols. One example is an *identify friend or foe* (IFF) protocol. These protocols are designed to help prevent friendly-fire incidents—where soldiers accidentally attack soldiers on their own side—while not seriously hampering the fight against the enemy.

A simple example of an IFF protocol appears in Figure 9.1. This protocol was reportedly used by the South African Air Force, or SAAF, when fighting in Angola in the mid-1970s [14]. The South Africans were fighting Angola for control of Namibia (known as Southwest Africa at the time). The Angolan side was flying Soviet MiG aircraft, piloted by Cubans.[3]

[2]Of course, this is an exaggeration—you get three tries before being shot by the security guard.

[3]This was one of the hot wars that erupted during the Cold War. Early in the war, the South Africans were amazed by the skill of the "Angolan" pilots. They eventually realized the pilots were actually Cuban when satellite photos revealed baseball diamonds.

The IFF protocol in Figure 9.1 works as follows. When the SAAF radar detects an aircraft approaching its base, a random number, or *challenge*, N is sent to the aircraft. All SAAF aircraft have access to a key K that they use to encrypt the challenge, $E(N, K)$, which is computed and sent back to the radar station. Time is of the essence, so all of this happens automatically, without human intervention. Since enemy aircraft do not know K, they cannot send back the required response. It would seem that this protocol gives the radar station a simple way to determine whether an approaching aircraft is a friend (let it land) or foe (shoot it down).

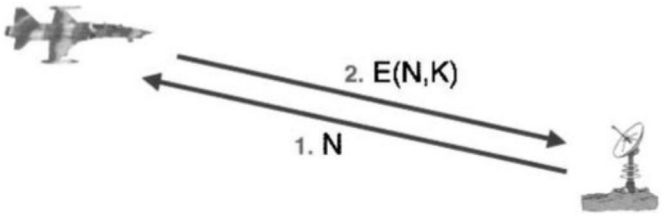


Figure 9.1: Identify Friend or Foe (IFF)

Unfortunately for those manning the radar station, there is a clever attack on the IFF system in Figure 9.1. Anderson has dubbed this attack the MiG-in-the-middle [14], which is a pun on man-in-the-middle. The scenario for the attack, which is illustrated in Figure 9.2, is as follows. While an SAAF Impala fighter is flying a mission over Angola, a Cuban-piloted MiG aircraft (the foe of the SAAF) loiters just outside of the range of the SAAF radar. When the Impala fighter is within range of a Cuban radar station in Angola, the MiG is told to move within range of the SAAF radar. As specified by the protocol, the SAAF radar sends the challenge N to the MiG. To avoid being shot down, the MiG needs to respond with $E(N, K)$, and quickly. Because the MiG does not know the key K, its situation appears hopeless. However, the MiG can forward the challenge N to its radar station in Angola, which, in turn, forwards the challenge to the SAAF Impala. The Impala fighter—not realizing that it has received the challenge from an enemy radar site—responds with $E(N, K)$. At this point, the Cuban radar relays the response $E(N, K)$ to the MiG, which can then provide it to the SAAF radar. Assuming this all happens fast enough, the SAAF radar will signal that the MiG is a friend, with disastrous consequences for the SAAF radar station and its operators.

Although it nicely illustrates an interesting security failure, it seems that this MiG-in-the-middle attack never actually occurred [15]. In any case, this is our first illustration of a security protocol failure, but it certainly won't be the last.

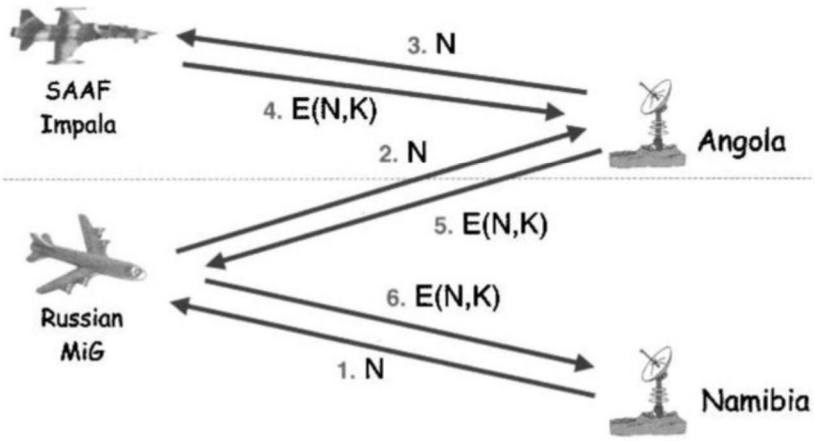


Figure 9.2: MiG-in-the-Middle

9.3 Authentication Protocols

"I can't explain myself, I'm afraid, Sir," said Alice,
"because I'm not myself you see."
— Lewis Carroll, *Alice in Wonderland*

Suppose that Alice must prove to Bob that she's Alice, where Alice and Bob are communicating over a network. Keep in mind that Alice can be a human or a machine, and ditto for Bob. In fact, in this networked scenario, Alice and Bob will almost invariably be machines, which has important implications that we'll consider in a moment.

In many cases, it's sufficient for Alice to prove her identity to Bob, without Bob proving his identity to Alice. But sometimes *mutual authentication* is necessary, that is, Bob must also prove his identity to Alice. It seems obvious that if Alice can prove her identity to Bob, then precisely the same protocol can be used in the other direction for Bob to prove his identity to Alice. We'll see that, in security protocols, the obvious approach is not always secure.

In addition to authentication, a *session key* is inevitably required. A session key is a symmetric key that will be used to protect the confidentiality and/or integrity of the current session, provided the authentication succeeds. Initially, we'll ignore the session key so that we can concentrate on authentication.

In certain situations, there may be other requirements placed on a security protocol. For example, we might require that the protocol use public keys, or symmetric keys, or hash functions. In addition, some situations might call for

a protocol that provides anonymity or plausible deniability (discussed below) or other not-so-obvious features.

We've previously considered the security issues associated with authentication on standalone computer systems. While such authentication presents its own set of challenges (hashing, salting, etc.), from the protocol perspective, it's straightforward. In contrast, authentication over a network requires very careful attention to protocol issues. When a network is involved, numerous attacks are available to Trudy that are generally not a concern on a standalone computer. When messages are sent over a network, Trudy can passively observe the messages and she can conduct various active attacks such as replaying old messages, inserting, deleting, or changing messages. In this book, we haven't previously encountered anything comparable to these types of attacks.

Our first attempt at authentication over a network is the protocol in Figure 9.3. This three-message protocol requires that Alice (the client) first initiate contact with Bob (the server) and state her identity. Then Bob asks for proof of Alice's identity, and Alice responds with her password. Finally, Bob uses Alice's password to authenticate Alice.

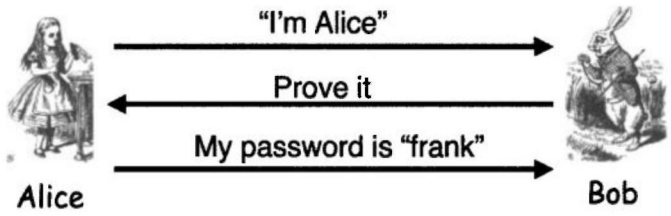


Figure 9.3: Simple Authentication

Although the protocol in Figure 9.3 is certainly simple, it has some major flaws. For one thing, if Trudy is able to observe the messages that are sent, she can later replay the messages to convince Bob that she is Alice, as illustrated in Figure 9.4. Since we are assuming these messages are sent over a network, this replay attack is a serious threat.

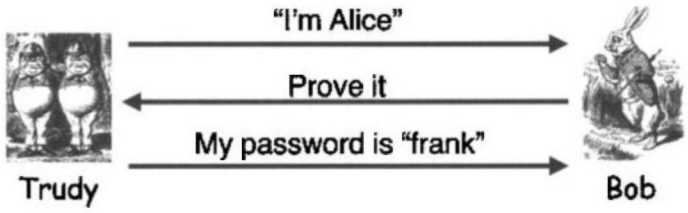


Figure 9.4: Replay Attack

Another issue with the too-simple authentication in Figure 9.3 is that Alice's password is sent in the clear. If Trudy observes the password when it is sent from Alice's computer, then Trudy knows Alice's password. This is even worse than a replay attack since Trudy can then pose as Alice on any site where Alice has reused this particular password. Another password issue with this protocol is that Bob must know Alice's password before he can authenticate her.

This simple authentication protocol is also inefficient, since the same effect could be accomplished in a single message from Alice to Bob. So, this protocol is a loser in every respect. Finally, note that the protocol in Figure 9.3 does not attempt to provide mutual authentication, which may be required in some cases.

For our next attempt at an authentication protocol, consider Figure 9.5. This protocol solves some of the problems of our previous simple authentication protocol. In this new-and-improved version, a passive observer, Trudy, will not learn Alice's password and Bob no longer needs to know Alice's password—although he must know the hash of Alice's password.

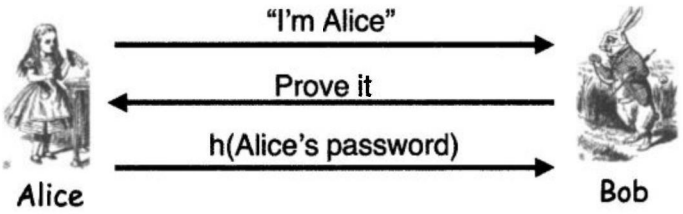


Figure 9.5: Simple Authentication with a Hash

The major flaw in the protocol of Figure 9.5 is that it's still subject to a replay attack, where Trudy records Alice's messages and later replays them to Bob. In this way, Trudy could be authenticated as Alice, without knowledge of Alice's password.

To authenticate Alice, Bob will need to employ a *challenge-response* mechanism. That is, Bob will send a challenge to Alice, and the response from Alice must be something that only Alice can provide and that Bob can verify. To prevent a replay attack, Bob can incorporate a "number used once," or *nonce*, in the challenge. That is, Bob will send a unique challenge each time, and the challenge will be used to compute the appropriate response. Bob can thereby distinguish the current response from a replay of a previous response. In other words, the nonce is used to ensure the freshness of the response. This approach to authentication with replay prevention is illustrated in Figure 9.6.

First, we'll design an authentication protocol using Alice's password. A password is something only Alice should know and Bob can verify—assuming that Bob knows Alice's password, that is.

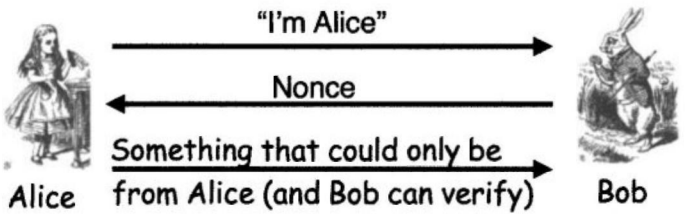


Figure 9.6: Generic Authentication

Our first serious attempt at an authentication protocol that is resistant to replay appears is Figure 9.7. In this protocol, the nonce sent from Bob to Alice is the challenge. Alice must respond with the hash of her password together with the nonce, which, assuming Alice's password is secure, serves to prove that the response was generated by Alice. Note that the nonce proves that the response is fresh and not a replay.

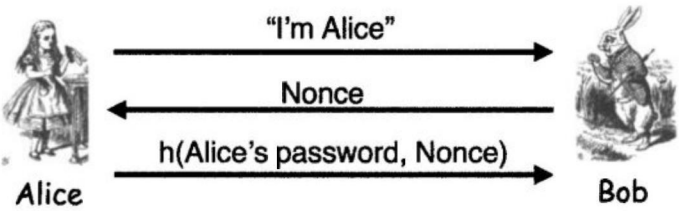


Figure 9.7: Challenge-Response

One problem with the protocol in Figure 9.7 is that Bob must know Alice's password. Furthermore, Alice and Bob typically represent machines rather than users, so it makes no sense to use passwords. After all, passwords are little more than a crutch used by humans because we are incapable of remembering keys. That is, passwords are about the closest thing to a key that humans can remember. So, if Alice and Bob are actually machines, they should be using keys instead of passwords.

9.3.1 Authentication Using Symmetric Keys

Having liberated ourselves from passwords, let's design a secure authentication protocol based on symmetric key cryptography. Recall that our notation for encrypting is $C = E(P, K)$ where P is plaintext, K is the key, and C is the ciphertext, while the notation for decrypting is $P = D(C, K)$. When discussing protocols, we are primarily concerned with attacks on protocols, not attacks on the cryptography used in protocols. Consequently, in this chapter we'll assume that the underlying cryptography is secure.

Suppose that Alice and Bob share the symmetric key K_{AB}. As in symmetric cryptography, we assume that nobody else has access to K_{AB}. Alice will authenticate herself to Bob by proving that she knows the key, without revealing the key to Trudy. In addition, the protocol must provide protection against a replay attack.

Our first symmetric key authentication protocol appears in Figure 9.8. This protocol is analogous to our previous password-based challenge-response protocol, but instead of hashing a nonce with a password, we've encrypted the nonce R with the shared symmetric key K_{AB}.

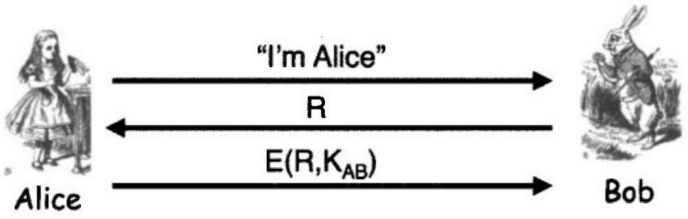


Figure 9.8: Symmetric Key Authentication Protocol

The symmetric key authentication protocol in Figure 9.8 allows Bob to authenticate Alice, since Alice can encrypt R with K_{AB}, Trudy cannot, and Bob can verify that the encryption was done correctly—Bob knows K_{AB}. This protocol prevents a replay attack, thanks to the nonce R, which ensures that each response is fresh. The protocol lacks mutual authentication, so our next task will be to develop a mutual authentication protocol based on symmetric keys.

Our first attempt at mutual authentication appears in Figure 9.9. This protocol is certainly efficient, and it does use symmetric key cryptography, but it has an obvious flaw. The third message in this protocol is simply a replay of the second, and consequently it proves nothing about the sender, be it Alice or Trudy.

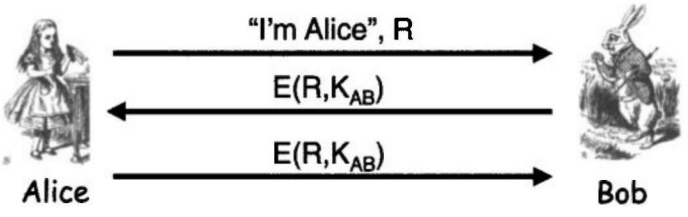


Figure 9.9: Mutual Authentication?

A more plausible approach to mutual authentication would be to use the secure authentication protocol in Figure 9.8 and repeat the process twice,

once for Bob to authenticate Alice and once more for Alice to authenticate Bob. We've illustrated this approach in Figure 9.10, where we've combined some messages for the sake of efficiency.

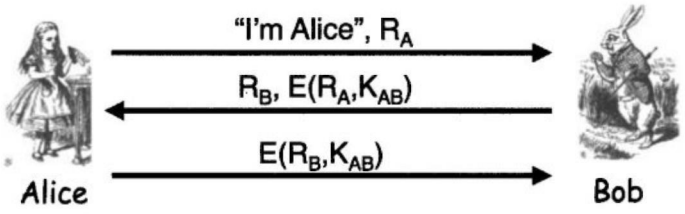


Figure 9.10: Secure Mutual Authentication?

Perhaps surprisingly, the protocol in Figure 9.10 is insecure—it is subject to an attack that is analogous to the MiG-in-the-middle attack discussed previously. In this attack, which is illustrated in Figure 9.11, Trudy initiates a conversation with Bob by claiming to be Alice and sends a challenge R_A to Bob. Following the protocol, Bob encrypts the challenge R_A and sends it, along with his challenge R_B, to Trudy. At this point Trudy appears to be stuck, since she doesn't know the key K_{AB}, and therefore she can't respond appropriately to Bob's challenge. However, Trudy cleverly opens a new connection to Bob where she again claims to be Alice and this time sends Bob his own "random" challenge R_B. Bob, following the protocol, responds with $E(R_B, K_{AB})$, which Trudy can now use to complete the first connection. Trudy can leave the second connection to time out, since she has—in the first connection—convinced Bob that she is Alice.

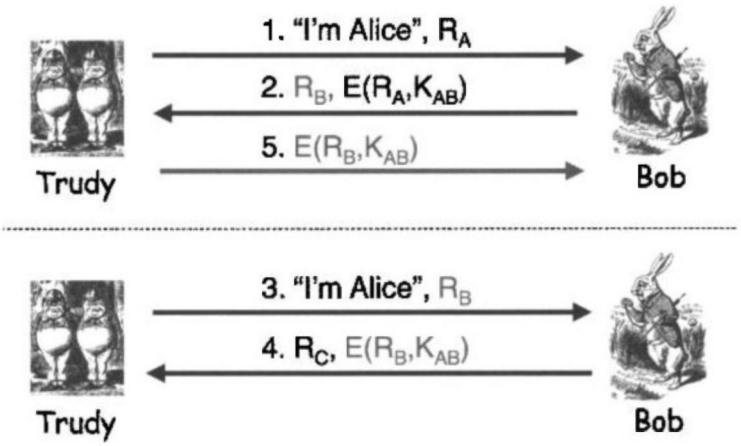


Figure 9.11: Trudy's Attack

The conclusion is that a non-mutual authentication protocol may not be secure for mutual authentication. Another conclusion is that protocols (and attacks on protocols) can be subtle. Yet another conclusion is that "obvious" changes to protocols can cause unexpected security problems.

In Figure 9.12, we've made a couple of minor changes to the insecure mutual authentication protocol of Figure 9.10. In particular, we've encrypted the user's identity together with the nonce. This change is sufficient to prevent Trudy's previous attack since she cannot use a response from Bob for the third message—Bob will realize that he encrypted it himself.

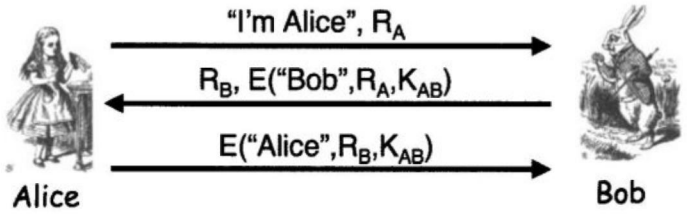


Figure 9.12: Strong Mutual Authentication Protocol

One lesson here is that it's a bad idea to have the two sides in a protocol do exactly the same thing, since this might open the door to an attack. Another lesson is that small changes to a protocol can result in big changes in its security.

9.3.2 Authentication Using Public Keys

In the previous section we devised a secure mutual authentication protocol using symmetric keys. Can we accomplish the same thing using public key cryptography? First, recall our public key notation. Encrypting a message M with Alice's public key is denoted $C = \{M\}_{\text{Alice}}$ while decrypt C with Alice's private key, and thereby recovering the plaintext M, is denoted $M = [C]_{\text{Alice}}$. Signing is also a private key operation. Of course, encryption and decryption are inverse operation, as are signing and signature verification, that is

$$[\{M\}_{\text{Alice}}]_{\text{Alice}} = M \quad \text{and} \quad \{[M]_{\text{Alice}}\}_{\text{Alice}} = M.$$

It's always important to remember that in public key cryptography, anybody can do public key operations, while only Alice can use her private key.[4]

Our first attempt at authentication using public key cryptography appears in Figure 9.13. This protocol allows Bob to authenticate Alice, since only Alice can do the private key operation that is necessary to reply with R in the third message. Also, assuming that the nonce R is chosen (by Bob) at

[4]Repeat to yourself 100 times: The public key is public.

random, a replay attack is not feasible. That is, Trudy cannot replay R from a previous iteration of the protocol, since the random challenge will almost certainly not be the same in a subsequent iteration.

However, if Alice uses the same key pair to encrypt as she uses for authentication, then there is a potential problem with the protocol in Figure 9.13. Suppose Trudy has previously intercepted a message encrypted with Alice's public key, say, $C = \{M\}_{\text{Alice}}$. Then Trudy can pose as Bob and send C to Alice in message two, and Alice will decrypt it and send the plaintext back to Trudy. From Trudy's perspective, it doesn't get any better than that. The moral of the story is that you should not use the same key pair for signing as you use for encryption.

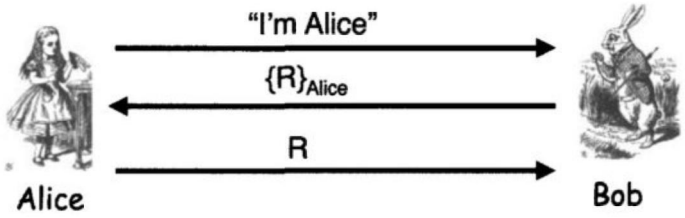


Figure 9.13: Authentication with Public Key Encryption

The authentication protocol in Figure 9.13 uses public key encryption. Is it possible to accomplish the same feat using digital signatures? In fact, it is, as illustrated in Figure 9.14.

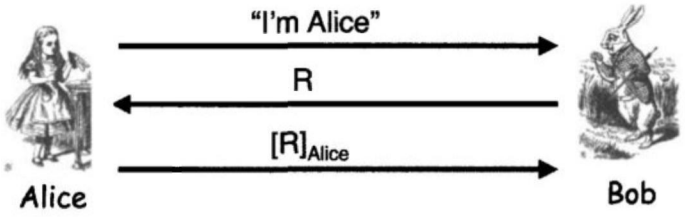


Figure 9.14: Authentication via Digital Signature

The protocol in Figure 9.14 has similar security issues as the public key encryption protocol in Figure 9.13. In Figure 9.14, if Trudy can pose as Bob, she can get Alice to sign anything. Again, the solution to this problem is to always use different key pairs for signing and encryption. Finally, note that, from Alice's perspective, the protocols in Figures 9.13 and 9.14 are identical, since in both cases she applies her private key to whatever shows up in message two.

9.3.3 Session Keys

Along with authentication, we invariably require a session key. Even when a symmetric key is used for authentication, we want to use a distinct session keys to encrypt data within each connection. The purpose of a session key is to limit the amount of data encrypted with any one particular key, and it also serves to limit the damage if one session key is compromised. A session key is used to provide confidentiality or integrity protection (or both) to the messages.

We want to establish the session key as part of the authentication protocol. That is, when the authentication is complete, we will also have securely established a shared symmetric key. Therefore, when analyzing an authentication protocol, we need to consider attacks on the authentication itself, as well as attacks on the session key.

Our next goal is to design an authentication protocol that also provides a shared symmetric key. It looks to be straightforward to include a session key in our secure public key authentication protocol. Such a protocol appears in Figure 9.15.

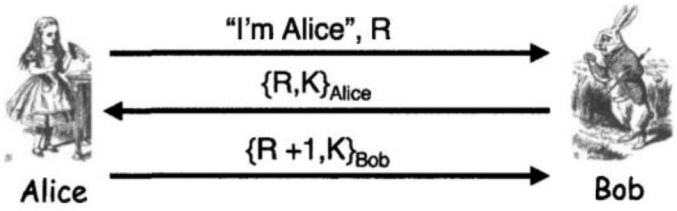


Figure 9.15: Authentication and a Session Key

One possible concern with the protocol of Figure 9.15 is that it does not provide for mutual authentication—only Alice is authenticated.[5] But before we tackle that issue, can we modify the protocol in Figure 9.15 so that it uses digital signatures instead of public key encryption? This also seems straightforward, and the result appears in Figure 9.16.

However, there is a fatal flaw in the protocol of Figure 9.16. Since the key is signed, anybody can use Bob's (or Alice's) public key and find the session key K. A session key that is public knowledge is definitely not secure. But before we dismiss this protocol entirely, note that it does provide mutual authentication, whereas the public key encryption protocol in Figure 9.15 does not. Can we combine these protocols so as to achieve both mutual

[5]One strange thing about this protocol is that the key K acts as Bob's challenge to Alice and the nonce R is useless. But there is a method to the madness, which will become clear shortly.

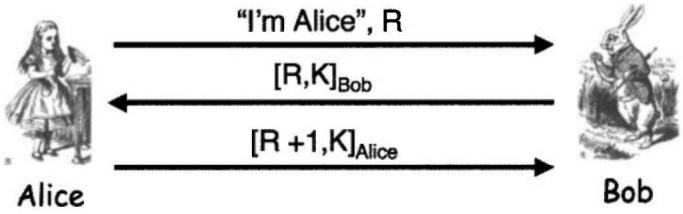


Figure 9.16: Signature-Based Authentication and Session Key

authentication and a secure session key? The answer is yes, and there are a couple of ways to do so.

Suppose that, instead of signing or encrypting the messages, we sign and encrypt the messages. Figure 9.17 illustrates such a sign and encrypt protocol. This appears to provide the desired secure mutual authentication and a secure session key.

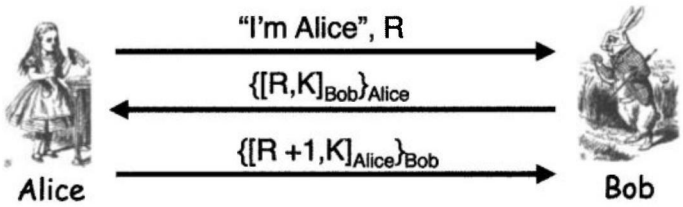


Figure 9.17: Mutual Authentication and Session Key

Since the protocol in Figure 9.17 provides mutual authentication and a session key using sign and encrypt, surely encrypt and sign must work, too. An encrypt and sign protocol appears in Figure 9.18.

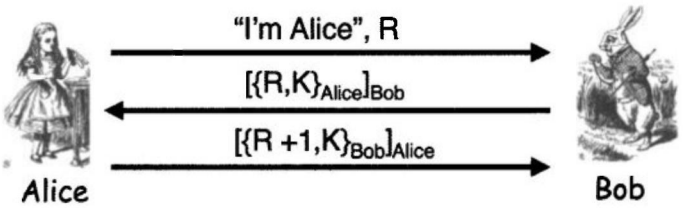


Figure 9.18: Encrypt and Sign Mutual Authentication

Note that the values $\{R, K\}_{\text{Alice}}$ and $\{R+1, K\}_{\text{Bob}}$ in Figure 9.18 are available to anyone who has access to Alice's or Bob's public keys (which, by assumption, is anybody who wants them). Since this is not the case in Figure 9.17, it might seem that sign and encrypt somehow reveals less

information than encrypt and sign. However, it appears that an attacker must break the public key encryption to recover K in either case and, if so, there is no security difference between the two. Recall that when analyzing protocols, we assume all crypto is strong, so breaking the encryption is not an option for Trudy.

9.3.4 Perfect Forward Secrecy

Now that we have conquered mutual authentication and session key establishment (using public keys), we turn our attention to *perfect forward secrecy*, or PFS. What is PFS? Rather than answer directly, we'll look at an example that illustrates what PFS is not. Suppose that Alice encrypts a message with a shared symmetric key K_{AB} and sends the resulting ciphertext to Bob. Trudy can't break the cipher to recover the key, so out of desperation she simply records all of the messages encrypted with the key K_{AB}. Now suppose that at some point in the future Trudy manages to get access to Alice's computer, where she finds the key K_{AB}. Then Trudy can decrypt the recorded ciphertext messages. While such an attack may seem unlikely, the problem is potentially significant since, once Trudy has recorded the ciphertext, the encryption key remains a vulnerability into the future. To avoid this problem, Alice and Bob must both destroy all traces of K_{AB} once they have finished using it. This might not be as easy as it seems, particularly if K_{AB} is a long-term key that Alice and Bob will need to use in the future. Furthermore, even if Alice is careful and properly manages her keys, she would have to rely on Bob to do the same (and vice versa).

PFS makes such an attack impossible. That is, even if Trudy records all ciphertext messages and she later recovers all long-term secrets (symmetric keys and/or private keys), she cannot decrypt the recorded messages. While it might seem that this is an impossibility, it is not only possible, but actually fairly easy to achieve in practice.

Suppose Bob and Alice share a long-term symmetric key K_{AB}. Then if they want PFS, they definitely can't use K_{AB} as their encryption key. Instead, Alice and Bob must agree on a session key K_S and forget K_S after it's no longer needed, i.e., after the current session ends. So, as in our previous protocols, Alice and Bob must find a way to agree on a session key K_S, by using their long-term symmetric key K_{AB}. However, for PFS we have the added condition that if Trudy later finds K_{AB}, she cannot determine K_S, even if she recorded all of the messages exchanged by Alice and Bob.

Suppose that Alice generates a session key K_S and sends $E(K_S, K_{AB})$ to Bob, that is, Alice simply encrypts the session key and sends it to Bob. If we are not concerned with PFS, this would be a sensible way to establish a session key in conjunction with an authentication protocol. However, this approach, which is illustrated in Figure 9.19, does not provide PFS. If Trudy records

all of the messages and later recovers K_{AB}, she can decrypt $E(K_S, K_{AB})$ to recover the session key K_S, which she can then use to decrypt the recorded ciphertext messages. This is precisely the attack that PFS is supposed to prevent.

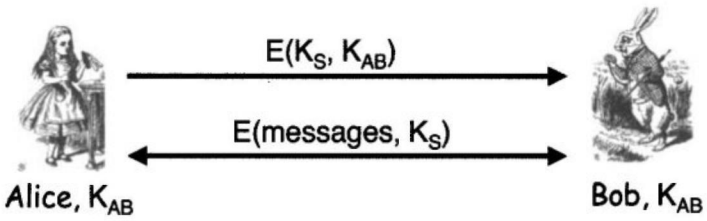


Figure 9.19: Naïve Attempt at PFS

There are actually several ways to achieve PFS, but the most elegant approach is to use an *ephemeral Diffie-Hellman* key exchange. As a reminder, the standard Diffie-Hellman key exchange protocol appears in Figure 9.20. In this protocol, g and p are public, Alice chooses her secret exponent a and Bob chooses his secret exponent b. Then Alice sends $g^a \bmod p$ to Bob and Bob sends $g^b \bmod p$ to Alice. Alice and Bob can each compute the shared secret $g^{ab} \bmod p$. Recall that the crucial weakness with Diffie-Hellman is that it is subject to a man-in-the-middle attack, as discussed in Section 4.4 of Chapter 4.

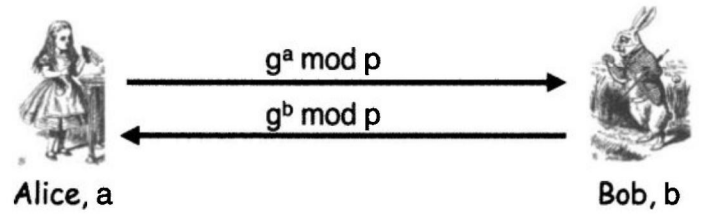


Figure 9.20: Diffie-Hellman

If we are to use Diffie-Hellman for PFS,[6] we must prevent the man-in-the-middle attack, and, of course, we must somehow assure PFS. The aforementioned ephemeral Diffie-Hellman can accomplish both. To prevent the MiM attack, Alice and Bob can use their shared symmetric key K_{AB} to encrypt the Diffie-Hellman exchange. Then to get PFS, all that is required is that, once Alice has computed the shared session key $K_S = g^{ab} \bmod p$, she must forget

[6]Your acronym-loving author was tempted to call this protocol DH4PFS or maybe EDH4PFS but, for once, he showed restraint.

her secret exponent a and, similarly, Bob must forget his secret exponent b. This protocol is illustrated in Figure 9.21.

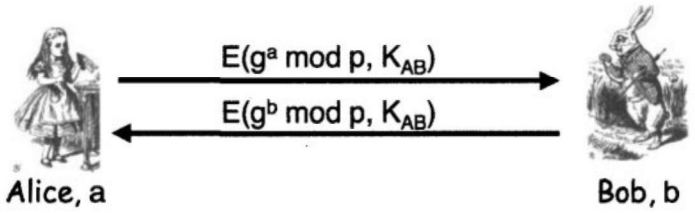


Figure 9.21: Ephemeral Diffie-Hellman for PFS

One interesting feature of the PFS protocol in Figure 9.21 is that once Alice and Bob have forgotten their respective secret exponents, even they can't reconstruct the session key K_S. If Alice and Bob can't recover the session key, certainly Trudy can be no better off. If Trudy records the conversation in Figure 9.21 and later is able to find K_{AB}, she will not be able to recover the session key K_S unless she can break Diffie-Hellman. Assuming the underlying crypto is strong, we have satisfied our requirements for PFS.

9.3.5 Mutual Authentication, Session Key, and PFS

Now let's put it all together and design a mutual authentication protocol that establishes a session key with PFS. The protocol in Figure 9.22, which is a slightly modified form of the encrypt and sign protocol from Figure 9.18, appears to fill the bill. It is a good exercise to give convincing arguments that Alice is actually authenticated (explaining exactly where and how that happens and why Bob is convinced he's talking to Alice), that Bob is authenticated, that the session key is secure, that PFS is provided, and that there are no obvious attacks.

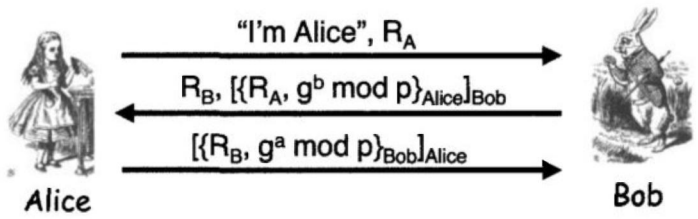


Figure 9.22: Mutual Authentication, Session Key and PFS

Now that we've developed a protocol that satisfies all of our security requirements, we can turn our attention to questions of efficiency. That is, we'll try to reduce the number of messages in the protocol or increase the

efficiency in some other way, such as by reducing the number of public key operations.

9.3.6 Timestamps

A *timestamp* T is a time value, typically expressed in milliseconds. With some care, a timestamp can be used in place of a nonce, since a current timestamp ensures freshness. The benefit of a timestamp is that we don't need to waste any messages exchanging nonces, assuming that the current time is known to both Alice and Bob. Timestamps are used in many real-world security protocols, such as Kerberos, which we discuss in the next chapter.

Along with the potential benefit of increased efficiency, timestamps create some security issues as well.[7] For one thing, the use of timestamps implies that time is a security-critical parameter. For example, if Trudy can attack Alice's system clock (or whatever Alice relies on for the current time), she may cause Alice's authentication to fail. A related problem is that we can't rely on clocks to be perfectly synchronized, so we must allow for some *clock skew*, that is, we must accept any timestamp that is close to the current time. In general, this can open a small window of opportunity for Trudy to conduct a replay attack—if she acts within the allowed clock skew a replay will be accepted. It is possible to close this window completely, but the solution puts an additional burden on the server (see Problem 27). In any case, we would like to minimize the clock skew without causing excessive failures due to time inconsistencies between Alice and Bob.

To illustrate the benefit of a timestamp, consider the authentication protocol in Figure 9.23. This protocol is essentially the timestamp version of the sign and encrypt protocol in Figure 9.17. Note that by using a timestamp, we're able to reduce the number of messages by a third.

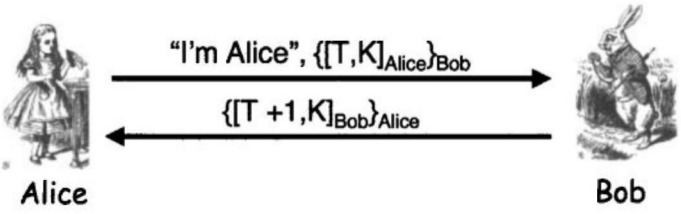


Figure 9.23: Authentication Using a Timestamp

The authentication protocol in Figure 9.23 uses a timestamp together with sign and encrypt and it appears to be secure. So it would seem obvious that the timestamp version of encrypt and sign must also be secure. This protocol is illustrated in Figure 9.24.

[7]This is yet another example of the "no free lunch" principle.

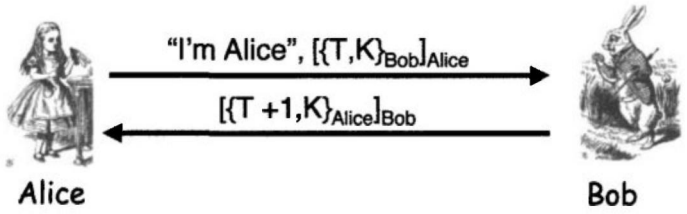


Figure 9.24: Encrypt and Sign Using a Timestamp

Unfortunately, with protocols, the obvious is not always correct. In fact, the protocol in Figure 9.24 is subject to attack. Trudy can recover $\{T, K\}_{\text{Bob}}$ by applying Alice's public key. Then Trudy can open a connection to Bob and send $\{T, K\}_{\text{Bob}}$ in message one, as illustrated in Figure 9.25. Following the protocol, Bob will then send the key K to Trudy in a form that Trudy can decrypt. This is not good, since K is the session key shared by Alice and Bob.

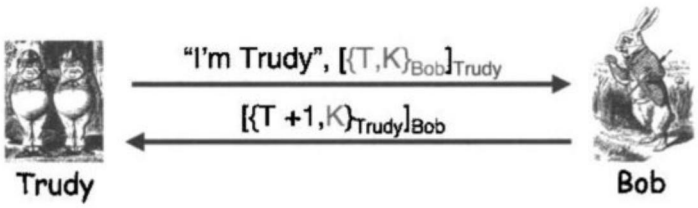


Figure 9.25: Trudy's Attack on Encrypt and Sign

The attack in Figure 9.25 shows that our encrypt and sign protocol is not secure when we use a timestamp. But our sign and encrypt protocol is secure when a timestamp is used. In addition, the nonce versions of both sign and encrypt as well as encrypt and sign are secure (see Figures 9.17 and 9.18). These examples nicely illustrate that, when it comes to security protocols, we should never take anything for granted.

Is the flawed protocol in Figure 9.24 fixable? In fact, there are several minor modifications that will make this protocol secure. For example, there's no reason to return the key K in the second message, since Alice already knows K and the only purpose of this message is to authenticate Bob. The timestamp in message two is sufficient to authenticate Bob. This secure version of the protocol is illustrated in Figure 9.26 (see also Problem 21).

In the next chapter, we'll discuss several well-known, real-world security protocols. These protocols use the concepts that we've presented in this chapter. But before moving on to the real world of Chapter 10, we briefly look at a couple of additional protocol topics. First, we'll consider a weak

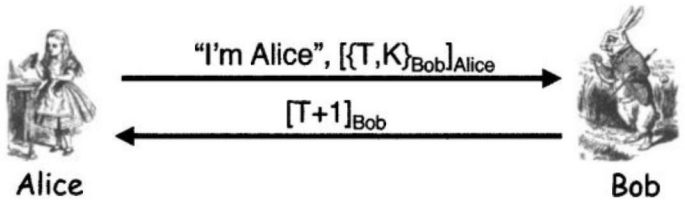


Figure 9.26: Secure Encrypt and Sign with a Timestamp

form of authentication that relies on TCP which, unfortunately, is sometimes used in practice. Finally, we discuss the Fiat-Shamir zero knowledge protocol. We'll encounter Fiat-Shamir again in the final chapter.

9.4 Authentication and TCP

In this section we'll take a quick look at how TCP is sometimes used for authentication. TCP was not designed to be used in this manner and, not surprisingly, this authentication method is not secure. But it does illustrate some interesting network security issues.

There is an undeniable temptation to use the IP address in a TCP connection for authentication.[8] If we could make this work, then we wouldn't need any of those troublesome keys or pesky authentication protocols.

Below, we'll give an example of TCP-based authentication and we illustrate an attack on the scheme. But first we briefly review the TCP three-way handshake, which is illustrated in Figure 9.27. The first message is a synchronization request, or SYN, whereas the second message, which acknowledges the synchronization request, is a SYN-ACK, and the third message—which can also contain data—acknowledges the previous message, and is simply known as an ACK.

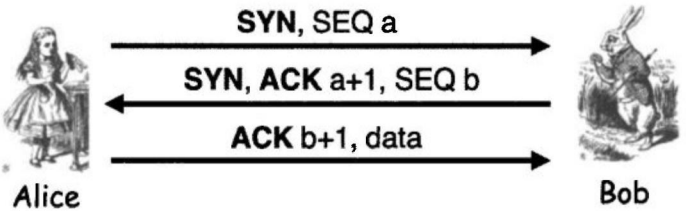


Figure 9.27: TCP 3-Way Handshake

[8] As we'll see in the next chapter, the IPSec protocol relies on the IP address for user identity in one of its modes. So, even people who should know better cannot always resist the temptation.

Suppose that Bob decides to rely on the completed three-way handshake to verify that he is connected to a specific IP address, which he knows belongs to Alice. Then, in effect, he is using the TCP connection to authenticate Alice. Since Bob sends the SYN-ACK to Alice's IP address, it's tempting to assume that the corresponding ACK must have come from Alice. In particular, if Bob verifies that ACK $b+1$ appears in message three, he has some reason to believe that Alice, at her known IP address, has received message two and responded, since message two contains SEQ b and nobody else should know b. An underlying assumption here is that Trudy can't see the SYN-ACK packet—otherwise she would know b and she could easily forge the ACK. Clearly, this is not a strong form of authentication. However, as a practical matter, it might actually be difficult for Trudy to intercept the message containing b. So, if Trudy cannot see b, is the protocol secure?

Even if Trudy cannot see the initial SEQ number b, she might be able to make a reasonable guess. If so, the attack scenario illustrated in Figure 9.28 may be feasible. In this attack, Trudy first sends an ordinary SYN packet to Bob, who responds with a SYN-ACK. Trudy examines the SEQ value b_1 in this SYN-ACK packet. Suppose that Trudy can use b_1 to predict Bob's next initial SEQ value b_2.[9] Then Trudy can send a packet to Bob with the source IP address forged to be Alice's IP address. Bob will send the SYN-ACK to Alice's IP address which, by assumption, Trudy can't see. But, if Trudy can guess b_2, she can complete the three-way handshake by sending ACK b_2+1 to Bob. As a result, Bob will believe that data received from Trudy on this particular TCP connection actually came from Alice.

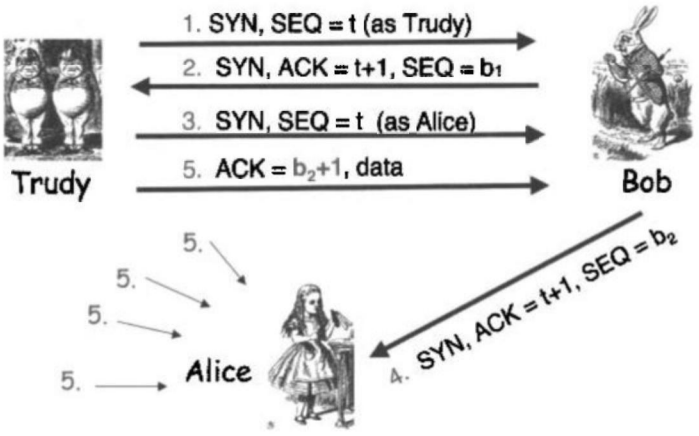


Figure 9.28: TCP "Authentication" Attack

[9]In practice, Trudy could send many SYN packets to Bob, trying to diagnose his initial sequence number generation scheme before actually attempting to guess a value.

Note that Bob always responds to Alice's IP address and, by assumption, Trudy cannot see his responses. But Bob will accept data from Trudy, thinking it came from Alice, as long as the connection remains active. However, when the data sent by Bob to Alice's IP address reaches Alice, Alice will terminate the connection since she has not completed the three-way handshake. To prevent this from happening, Trudy could mount a denial of service attack on Alice by sending enough messages so that Bob's messages can't get through—or, even if they do get through, Alice can't respond. This denial of service is illustrated Figure 9.28. Of course, if Alice happens to be offline, Trudy could conduct the attack without having to do this denial of service on Alice.

This attack is well known, and as a result initial SEQ numbers are supposed to be generated at random. So, how random are initial SEQ numbers? Surprisingly, they're often not very random at all. For example, Figure 9.29 provides a visual comparison of random initial SEQ numbers versus the highly biased initial SEQ numbers generated under an early version of Mac OS X. The Mac OS X numbers are biased enough that the attack in Figure 9.28 would have a reasonable chance of success. Many other vendors fail to generate random initial SEQ numbers, as can be seen from the fascinating pictures at [335].

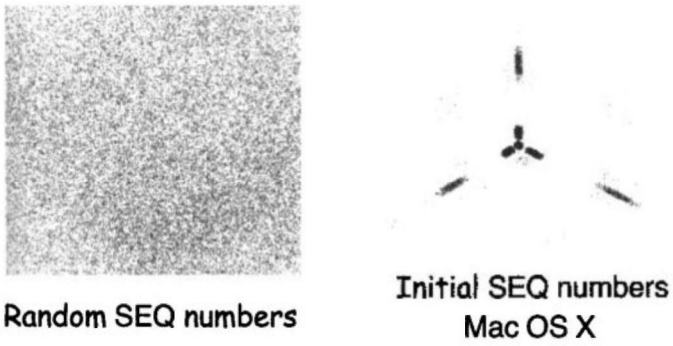


Figure 9.29: Plots of Initial SEQ Numbers (Courtesy of Michal Zalewski [335])

Even if initial SEQ numbers are random, it's a bad idea to rely on a TCP connection for authentication. A much better approach would be to employ a secure authentication protocol after the three-way handshake has completed. Even a simple password scheme would be far superior to relying on TCP. But, as often occurs in security, the TCP authentication method is sometimes used in practice simply because it's there, it's convenient, and it doesn't annoy users—not because it's secure.

9.5 Zero Knowledge Proofs

In this section we'll discuss a fascinating authentication scheme developed by Fiege, Fiat, and Shamir [111] (yes, that Shamir), but usually known simply as Fiat-Shamir. We'll mention this method again in Chapter 13 when we discuss Microsoft's trusted operating system.

In a *zero knowledge proof*,[10] or ZKP, Alice wants to prove to Bob that she knows a secret without revealing any information about the secret—neither Trudy nor Bob can learn anything about the secret. Bob must be able to verify that Alice knows the secret, even though he gains no information about the secret. On the face of it, this sounds impossible. However, there is an interactive probabilistic process whereby Bob can verify that Alice knows a secret to an arbitrarily high probability. This is an example of an interactive proof system.

Before describing such a protocol, we first consider Bob's Cave,[11] which appears in Figure 9.30. Suppose that Alice claims to know the secret phrase ("open sarsaparilla"[12]) that opens the door between R and S in Figure 9.30. Can Alice convince Bob that she knows the secret phrase without revealing any information about it?

Figure 9.30: Bob's Cave

Consider the following protocol. Alice enters Bob's Cave and flips a coin to decide whether to position herself at point R or S. Bob then enters the cave and proceeds to point Q. Suppose that Alice happens to be positioned at point R. This situation is illustrated in Figure 9.31.

Then Bob flips a coin to randomly select one side or the other and asks Alice to appear from that side. With the situation as in Figure 9.31, if Bob happens to select side R, then Alice would appear at side R whether she knows the secret phrase or not. But if Bob happens to choose side S, then Alice can only appear on side S if she knows the secret phrase that opens

[10]Not to be confused with a "zero knowledge Prof."

[11]Traditionally, Ali Baba's Cave is used here.

[12]Traditionally, the secret phrase is "open says me," which sounds a lot like "open sesame." In the cartoon world, "open sesame" somehow became "open sarsaparilla" [242].

Figure 9.31: Bob's Cave Protocol

the door between R and S. In other words, if Alice doesn't know the secret phrase, the probability that she can trick Bob into believing that she does is $\frac{1}{2}$. This does not seem particularly useful, but if the protocol is repeated n times, then the probability that Alice can trick Bob every time is only $(\frac{1}{2})^n$. So, Alice and Bob will repeat the protocol n times and Alice must pass every time before Bob will believe she knows the secret phrase.

Note that if Alice (or Trudy) does not know the secret phrase, there is always a chance that she can trick Bob into believing that she does. However, Bob can make this probability as small as he desires by choosing n appropriately. For example, with $n = 20$ there is less than a 1 in 1,000,000 chance that "Alice" would convince Bob that she knows the phrase when she does not. Also, Bob learns nothing about the secret phrase in this protocol. Finally, it is critical that Bob randomly chooses the side where he asks Alice to appear—if Bob's choice is predictable, then Alice (or Trudy) would have a better chance of tricking Bob and thereby breaking the protocol.

While Bob's Cave indicates that zero knowledge proofs are possible in principle, cave-based protocols are not particularly popular. Can we achieve the same effect without the cave? The answer is yes, thanks to the Fiat-Shamir protocol.

Fiat-Shamir relies on the fact that finding a square root modulo N is as difficult as factoring. Suppose $N = pq$, where p and q are prime. Alice knows a secret S, which, of course, she must keep secret. The numbers N and $v = S^2 \bmod N$ are public. Alice must convince Bob that she knows S without revealing any information about S.

The Fiat-Shamir protocol, which is illustrated in Figure 9.32, works as follows. Alice randomly selects a value r, and she computes $x = r^2 \bmod N$. In message one, Alice sends x to Bob. In message two, Bob chooses a random value $e \in \{0, 1\}$, which he sends to Alice who, in turn, then computes the quantity $y = rS^e \bmod N$. In the third message Alice sends y to Bob. Finally, Bob needs to verify that

$$y^2 = xv^e \bmod N,$$

which, if everyone has followed the protocol, holds true since

$$y^2 = r^2 S^{2e} = r^2 (S^2)^e = x v^e \bmod N. \tag{9.1}$$

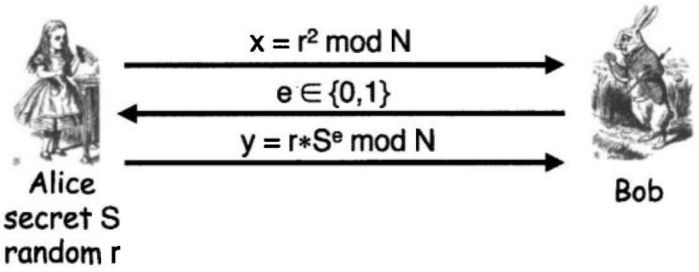


Figure 9.32: Fiat-Shamir Protocol

In message two, Bob sends either $e = 0$ or $e = 1$. Let's consider these cases separately. If Bob sends $e = 1$, then Alice responds with $y = r \cdot S \bmod N$ in the third message, and equation (9.1) becomes

$$y^2 = r^2 \cdot S^2 = r^2 \cdot (S^2) = x \cdot v \bmod N.$$

Note that in this case, Alice must know the secret S.

On the other hand, if Bob sends $e = 0$ in message two, then Alice responds in the third message with $y = r \bmod N$ and equation (9.1) becomes

$$y^2 = r^2 = x \bmod N.$$

Note that in this case, Alice does not need to know the secret S. This may seem strange, but it's roughly equivalent to the situation in Bob's Cave where Alice did not need to open the secret passage to come out on the correct side. Regardless, it is tempting to have Bob always send $e = 1$. However, we'll see in a moment that that this would not be wise.

The first message in the Fiat-Shamir protocol is the *commitment* phase, since Alice commits to her choice of r by sending $x = r^2 \bmod N$ to Bob. That is, Alice cannot change her mind (she is committed to r), but she has not revealed r, since finding modular square roots is hard. The second message is the *challenge* phase—Bob is challenging Alice to provide the correct response. The third message is the *response* phase, since Alice must respond with the correct value. Bob then verifies the response using equation (9.1). These phases correspond to the three steps in Bob's Cave protocol in Figure 9.31, above.

The mathematics behind the Fiat-Shamir protocol works, that is, assuming everyone follows the protocol, Bob can verify $y^2 = x v^e \bmod N$ from the

information he receives. But this does not establish the security of the protocol. To do so, we must determine whether an attacker, Trudy, can convince Bob that she knows Alice's secret S and thereby convince Bob that she is Alice.

Suppose Trudy expects Bob to send the challenge $e = 0$ in message two. Then Trudy can send $x = r^2 \bmod N$ in message one and $y = r \bmod N$ in message three. That is, Trudy simply follows the protocol in this case, since she does not need to know the secret S.

On the other hand, if Trudy expects Bob to send $e = 1$, then she can send $x = r^2 v^{-1} \bmod N$ in message one and $y = r \bmod N$ in message three. Following the protocol, Bob will compute $y^2 = r^2$ and $xv^e = r^2 v^{-1} v = r^2$ and he will find that equation (9.1) holds. Bob therefore accepts the result as valid.

The conclusion here is that Bob must choose $e \in \{0, 1\}$ at random (as specified by the protocol). If so, then Trudy can only trick Bob with probability $\frac{1}{2}$, and, as with Bob's Cave, after n iterations, the probability that Trudy can fool Bob is only $(\frac{1}{2})^n$.

So, Fiat-Shamir requires that Bob's challenge $e \in \{0, 1\}$ be unpredictable. In addition, Alice must generate a random r at each iteration of the protocol or her secret S will be revealed (see Problem 40 at the end of this chapter).

Is the Fiat-Shamir protocol really zero knowledge? That is, can Bob—or anyone else—learn anything about Alice's secret S? Recall that v and N are public, where $v = S^2 \bmod N$. In addition, Bob sees $r^2 \bmod N$ in message one, and, assuming $e = 1$, Bob sees $rS \bmod N$ in message three. If Bob can find r from $r^2 \bmod N$, then he can find S. But finding modular square roots is computationally infeasible. If Bob were somehow able to find such square roots, he could obtain S directly from the public value v without bothering with the protocol at all. While this is not a rigorous proof that Fiat-Shamir is zero knowledge, it does indicate that there is nothing obvious in the protocol itself that helps Bob (or anyone else) to determine Alice's secret S.

Is there an security benefit of Fiat-Shamir, or is it just fun and games for mathematicians? If public keys are used for authentication, then each side must know the other side's public key. At the start of the protocol, typically Alice would not know Bob's public key, and vice versa. So, in many public key-based protocols Bob sends his certificate to Alice. But the certificate identifies Bob, and consequently this exchange would tell Trudy that Bob is a party to the transaction. In other words, public keys make it hard for the participants to remain anonymous.

A potential advantage of zero knowledge proofs is that they allow for authentication with anonymity. In Fiat-Shamir, both sides must know the public value v, but there is nothing in v that identifies Alice, and there is nothing in the messages that are passed that must identify Alice. This is an advantage that has led Microsoft to include support for zero knowledge

proofs in its Next Generation Secure Computing Base, or NGSCB, which we'll discuss in Chapter 13. The bottom line is that Fiat-Shamir does have some potential practical utility.

9.6 The Best Authentication Protocol?

In general there is no "best" authentication protocol. What is best for a particular situation will depend on many factors. At a minimum, we need to consider the following questions.

- What is the sensitivity of the application?
- How much delay is tolerable?
- Do we want to deal with time as a security critical parameter?
- What type of crypto is supported—public key, symmetric key, or hash functions?
- Is mutual authentication required?
- Is a session key required?
- Is perfect forward secrecy desired?
- Is anonymity a concern?

In the next chapter, we'll see that there are additional issues that can influence our choice of protocol.

9.7 Summary

In this chapter we discussed several different ways to authenticate and establish a session key over an insecure network. We can accomplish these feats using symmetric keys, public keys, or hash functions (with symmetric keys). We also learned how to achieve perfect forward secrecy, and we considered the benefits (and potential drawbacks) of using timestamps.

Along the way, we came across many security pitfalls. You should now have some appreciation for the subtle issues that can arise with security protocols. This will be useful in the next chapter where we look closely at several real-world security protocols. We'll see that, despite extensive development effort by lots of smart people, such protocols are not immune to some of the security flaws highlighted in this chapter.

9.8 Problems

1. Modify the authentication protocol in Figure 9.12 so that it uses a hash function instead of symmetric key encryption. The resulting protocol must be secure.

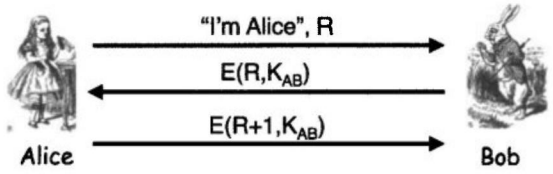
2. The insecure protocol in Figure 9.24 was modified in Figure 9.26 to be secure. Find two other distinct ways to slightly modify the protocol in Figure 9.24 so that the resulting protocol is secure. Your protocols must use a timestamp and "encrypt and sign."

3. We want to design a secure mutual authentication protocol based on a shared symmetric key. We also want to establish a session key, and we want perfect forward secrecy.

   a. Design such a protocol that uses three messages.

   b. Design such a protocol that uses two messages.

4. Consider the following mutual authentication protocol, where K_{AB} is a shared symmetric key.



   Give two different attacks that Trudy can use to convince Bob that she is Alice.

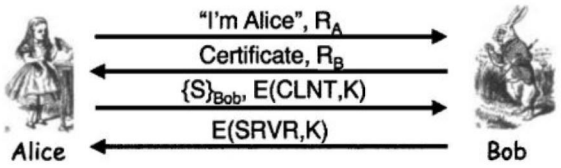
5. Consider the attack on TCP authentication illustrated in Figure 9.28. Suppose that Trudy cannot guess the initial sequence number b_2 exactly. Instead, Trudy can only narrow b_2 down to one of, say, 1,000 possible values. How can Trudy conduct an attack so that she is likely to succeed?

6. Timestamps can be used in place of nonces in security protocols.

   a. What is the primary advantage of using timestamps?

   b. What is the primary disadvantage of using timestamps?

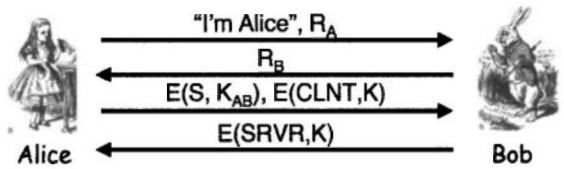
7. Consider the following protocol, where CLNT and SRVR are constants, and the session key is $K = h(S, R_A, R_B)$.



a. Does Alice authenticate Bob? Justify your answer.

b. Does Bob authenticate Alice? Justify your answer.

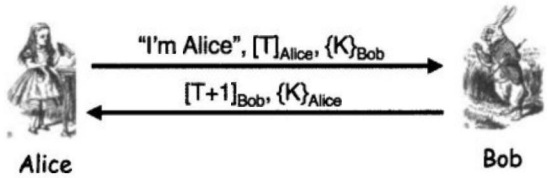
8. Consider the following protocol, where K_{AB} is a shared symmetric key, CLNT and SRVR are constants, and $K = h(S, R_A, R_B)$ is the session key.



a. Does Alice authenticate Bob? Justify your answer.

b. Does Bob authenticate Alice? Justify your answer.

9. The following two-message protocol is designed for mutual authentication and to establish a session key K. Here, T is a timestamp.



This protocol is insecure. Illustrate a successful attack by Trudy.

10. Suppose R is a random challenge sent in the clear from Alice to Bob and K is a symmetric key known only to Alice and Bob. Which of the following are secure session keys and which are not? Justify your answers.

a. $R \oplus K$

b. $E(R, K)$

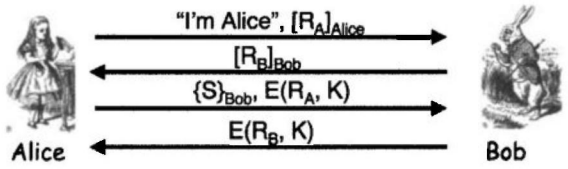
c. $E(K, R)$

d. $h(K, R)$

e. $h(R, K)$

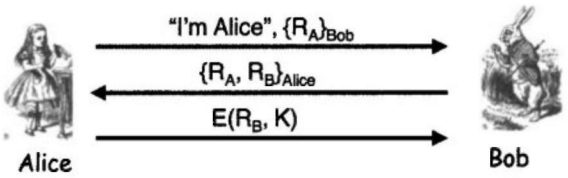
11. Design a secure two-message authentication protocol that provides mutual authentication and establishes a session key K. Assume that Alice and Bob know each other's public keys beforehand. Does your protocol protect the anonymity of Alice and Bob from a passive attacker (i.e., an attacker who can only observe messages sent between Alice and Bob)? If not, modify your protocol so that it does provide anonymity.

12. For some particular security protocol, suppose that Trudy can construct messages that appear to any observer (including Alice and/or Bob) to be valid messages between Alice and Bob. Then the protocol is said to provide *plausible deniability*. The idea here is that Alice and Bob can (plausibly) argue that any conversation they had using the protocol never actually occurred—it could have been faked by Trudy. Consider the following protocol, where $K = h(R_A, R_B, S)$.



Does this protocol provide plausible deniability? If so, why? If not, slightly modify the protocol so that it does, while still providing mutual authentication and a secure session key.

13. Consider the following protocol where $K = h(R_A, R_B)$.

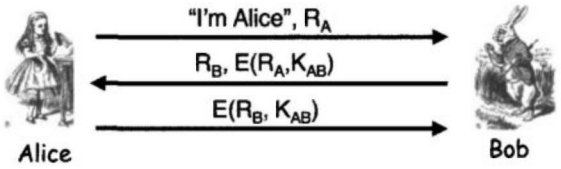


Does this protocol provide for plausible deniability (see Problem 12)? If so, why? If not, slightly modify the protocol so that it does, while still providing mutual authentication and a secure session key.

14. Design a mutual authentication protocol that employs digital signatures for authentication and provides plausible deniability (see Problem 12).

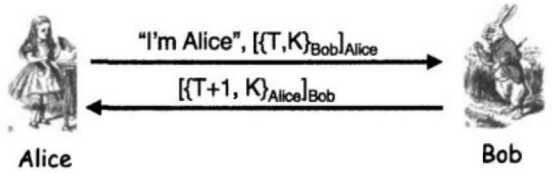
15. Is plausible deniability (see Problem 12) a feature or a security flaw? Explain.

16. The following mutual authentication protocol is based on a shared symmetric key K_{AB}.



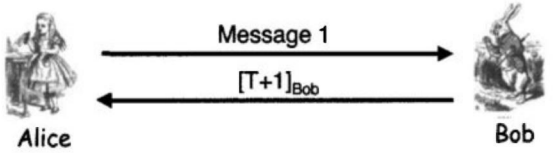
Show that Trudy can attack the protocol to convince Bob that she is Alice, where, as usual, we assume that the cryptography is secure. Modify the protocol to prevent such an attack by Trudy.

17. Consider the following mutual authentication and key establishment protocol, which employs a timestamp T and public key cryptography.



Show that Trudy can attack the protocol to discover the key K where, as usual, we assume that the cryptography is secure. Modify the protocol to prevent such an attack by Trudy.

18. Consider the following mutual authentication and key establishment protocol, which uses a timestamp T and public key cryptography.



For each of the following cases, explain whether or not the resulting protocol provides an effective means for secure mutual authentication and a secure session key K. Ignore replay attacks based solely on the clock skew.

a. Message 1: $\{[T, K]_{\text{Alice}}\}_{\text{Bob}}$

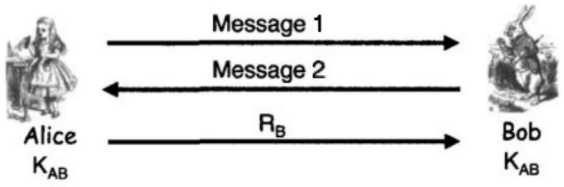
b. Message 1: $\{\text{"Alice"}, [T, K]_{\text{Alice}}\}_{\text{Bob}}$

c. Message 1: $\text{"Alice"}, \{[T, K]_{\text{Alice}}\}_{\text{Bob}}$

d. Message 1: T, $\text{"Alice"}, \{[K]_{\text{Alice}}\}_{\text{Bob}}$

e. Message 1: $\text{"Alice"}, \{[T]_{\text{Alice}}\}_{\text{Bob}}$ and let $K = h(T)$

19. Consider the following three-message mutual authentication and key establishment protocol, which is based on a shared symmetric key K_{AB}.



For each of the following cases, briefly explain whether or not the resulting protocol provides an effective means for secure mutual authentication and a secure session key K.

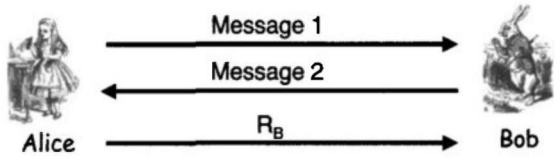
a. Message 1: $E(\text{"Alice"}, K, R_A, K_{AB})$, Message 2: R_A, $E(R_B, K_{AB})$

b. Message 1: "Alice", $E(K, R_A, K_{AB})$, Message 2: R_A, $E(R_B, K)$

c. Message 1: "Alice", $E(K, R_A, K_{AB})$, Message 2: R_A, $E(R_B, K_{AB})$

d. Message 1: "Alice", R_A, Message 2: $E(K, R_A, R_B, K_{AB})$

20. Consider the following three-message mutual authentication and key establishment protocol, which is based on public key cryptography.



For each of the following cases, briefly explain whether or not the resulting protocol provides an effective means for secure mutual authentication and a secure session key K.

a. Message 1: $\{\text{"Alice"}, K, R_A\}_{\text{Bob}}$, Message 2: R_A, R_B

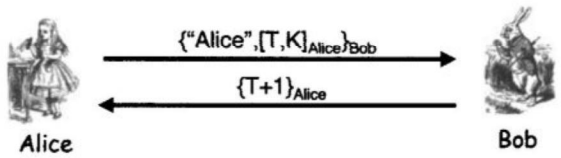
b. Message 1: "Alice", $\{K, R_A\}_{\text{Bob}}$, Message 2: R_A, $\{R_B\}_{\text{Alice}}$

c. Message 1: "Alice", $\{K\}_{\text{Bob}}$, $[R_A]_{\text{Alice}}$, Message 2: R_A, $[R_B]_{\text{Bob}}$

d. Message 1: R_A, $\{\text{"Alice"}, K\}_{\text{Bob}}$, Message 2: $[R_A]_{\text{Bob}}$, $\{R_B\}_{\text{Alice}}$

e. Message 1: $\{\text{"Alice"}, K, R_A, R_B\}_{\text{Bob}}$, Message 2: R_A, $\{R_B\}_{\text{Alice}}$

21. Consider the following mutual authentication and key establishment protocol (it may be instructive to compare this protocol to the protocol in Figure 9.26).

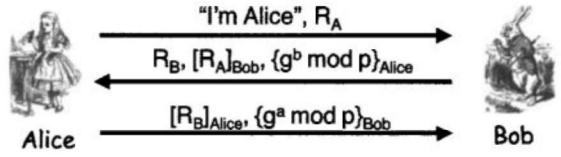


Suppose that Trudy pretends to be Bob. Further, suppose that Trudy can guess the value of T to within 5 minutes, and the resolution of T is to the nearest millisecond.

a. What is the probability that Trudy can send a correct response in message two, causing Alice to erroneously authenticate Trudy as Bob?

b. Give two distinct modifications to the protocol, each of which make Trudy's attack more difficult, if not impossible.

22. Consider the following mutual authentication and key establishment protocol, where the session key is given by $K = g^{ab} \bmod p$.



Suppose that Alice attempts to initiate a connection with Bob using this protocol.

a. Show that Trudy can attack the protocol so that both of the following will occur.

   i. Alice and Bob authenticate each other.

   ii. Trudy knows Alice's session key.

   Hint: Consider a man-in-the-middle attack.

b. Is this attack of any use to Trudy?

23. For each of the following cases, design a mutual authentication and key establishment protocol that uses public key cryptography and minimizes the number of messages.

a. Use a timestamp to authenticate Alice and a nonce to authenticate Bob.

b. Use a nonce to authenticate Alice and a timestamp to authenticate Bob.

24. Suppose we replace the third message of the protocol in Figure 9.22 with

$$\{R_B\}_{\text{Bob}}, \quad g^a \bmod p.$$

a. How can Trudy convince Bob that she is Alice, that is, how can Trudy break the authentication?

b. Can Trudy convince Bob that she is Alice and also determine the session key that Bob will use?

25. Suppose we replace the second message of the protocol in Figure 9.22 with

$$R_B, \ [R_A]_{\text{Bob}}, \quad g^b \bmod p,$$

and we replace the third message with

$$[R_B]_{\text{Alice}}, \quad g^a \bmod p.$$

a. Can Trudy convince Bob that she is Alice, that is, can Trudy break the authentication?

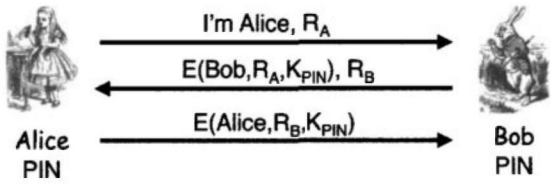
b. Can Trudy determine the session key that Alice and Bob will use?

26. In the text, it is claimed that the protocol in Figure 9.18 is secure, while the similar protocol in Figure 9.24 is not. Why does the attack on the latter protocol not succeed against the former?

27. A timestamp-based protocol may be subject to a replay attack, provided that Trudy can act within the clock skew. Reducing the acceptable clock skew might make the attack more difficult, but it will not prevent the attack unless the skew is zero, which is impractical. Assuming a non-zero clock skew, what can Bob, the server, do to prevent attacks based on the clock skew?

28. Modify the identify friend or foe (IFF) protocol discussed at the beginning of the chapter so that it's no longer susceptible to the MiG-in-the-middle attack.

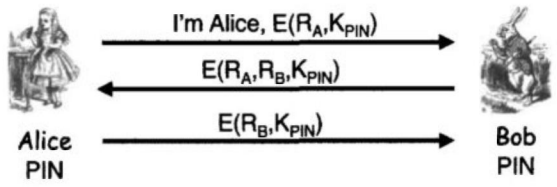
29. Consider the authentication protocol below, which is based on knowledge of a shared 4-digit PIN number. Here, $K_{\text{PIN}} = h(\text{PIN}, R_A, R_B)$.



a. Suppose that Trudy passively observes one iteration of the protocol. Can she determine the 4-digit PIN number? Justify your answer.

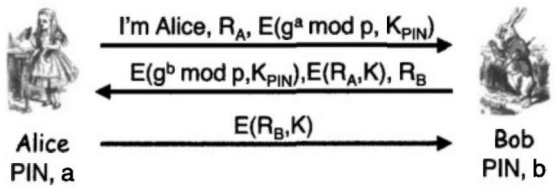
b. Suppose that the PIN number is replaced by a 256-bit shared symmetric key. Is the protocol secure? Why or why not?

30. Consider the authentication protocol below, which is based on knowledge of a shared 4-digit PIN number. Here, $K_{\text{PIN}} = h(\text{PIN})$.



Suppose that Trudy passively observes one iteration of the protocol. Can she then determine the 4-digit PIN? Justify your answer.

31. Consider the authentication protocol below, which is based on knowledge of a shared 4-digit PIN number and uses Diffie-Hellman. Here, $K_{\text{PIN}} = h(\text{PIN})$ and $K = g^{ab} \bmod p$.



a. Suppose that Trudy passively observes one iteration of the protocol. Can she then determine the 4-digit PIN number? Justify your answer.

b. Suppose that Trudy can actively attack the protocol. Can she determine the 4-digit PIN? Explain.

32. Describe a way to provide perfect forward secrecy that does not use Diffie-Hellman.

33. Can you achieve an effect similar to perfect forward secrecy (as described in this chapter) using only symmetric key cryptography? If so, give such a protocol and, if not, why not?

34. Design a zero knowledge protocol analogy that uses Bob's Cave and only requires one iteration for Bob to determine with certainty whether or not Alice knows the secret phrase.

35. The analogy between Bob's Cave and the Fiat-Shamir protocol is not entirely accurate. In the Fiat-Shamir protocol, Bob knows which value of e will force Alice to use the secret value S, assuming Alice follows the protocol. That is, if Bob chooses $e = 1$, then Alice must use the secret value S to construct the correct response in message three, but if Bob chooses $e = 0$, then Alice does not use S. As noted in the text, Bob must choose e at random to prevent Trudy from breaking the protocol. In the Bob's Cave analogy, Bob does not know whether Alice was required to use the secret phrase or not (again, assuming that Alice follows the protocol).

    a. Modify the cave analogy so that Bob knows whether Alice used the secret phrase or not, assuming that Bob is not allowed to see which side Alice actually chooses. Bob's New-and-Improved Cave protocol must still resist an attack by someone who does not know the secret phrase.

    b. Does your new cave analogy differ from the Fiat-Shamir protocol in any significant way?

36. Suppose that in the Fiat-Shamir protocol in Figure 9.32 we have $N = 63$ and $v = 43$. Recall that Bob accepts an iteration of the protocol if he verifies that $y^2 = x \cdot v^e \bmod N$.

    a. In the first iteration of the protocol, Alice sends $x = 37$ in message one, Bob sends $e = 1$ in message two, and Alice sends $y = 4$ in message three. Does Bob accept this iteration of the protocol? Why or why not?

    b. In the second iteration of the protocol, Alice sends $x = 37$, Bob sends $e = 0$, and Alice sends $y = 10$. Does Bob accept this iteration of the protocol? Why or why not?

    c. Find Alice's secret value S. Hint: $10^{-1} = 19 \bmod 63$.

37. Suppose that in the Fiat-Shamir protocol in Figure 9.32 we have $N = 77$ and $v = 53$.

a. Suppose that Alice sends $x = 15$ in message one, Bob sends $e = 1$ in message two, and Alice sends $y = 5$ in message three. Show that Bob accepts this iteration of the protocol.

b. Suppose Trudy knows in advance that Bob will select $e = 1$ in message two. If Trudy selects $r = 10$, what can she send for x in message one and y in message three so that Bob accepts this iteration of the protocol? Using your answer, show that Bob actually accepts this iteration. Hint: $53^{-1} = 16 \bmod 77$.

38. Suppose that in the Fiat-Shamir protocol in Figure 9.32 we have $N = 55$ and Alice's secret is $S = 9$.

a. What is v?

b. If Alice chooses $r = 10$, what does Alice send in the first message?

c. Suppose Alice chooses $r = 10$ and Bob sends $e = 0$ in message two. What does Alice send in the third message?

d. Suppose Alice chooses $r = 10$ and Bob sends $e = 1$ in message two. What does Alice send in the third message?

39. Consider the Fiat-Shamir protocol in Figure 9.32. Suppose that the public values are $N = 55$ and $v = 5$. Suppose Alice sends $x = 4$ in the first message, Bob sends $e = 1$ in the second message, and Alice sends $y = 30$ in message three. Show that Bob will verify Alice's response in this case. Can you find Alice's secret S?

40. In the Fiat-Shamir protocol in Figure 9.32, suppose that Alice gets lazy and she decides to use the same "random" r for each iteration.

a. Show that Bob can determine Alice's secret S.

b. Why is this a security concern?

41. Suppose that in the Fiat-Shamir protocol, as illustrated in Figure 9.32, we have $N = 27{,}331$ and $v = 7339$.

a. In the first iteration, Alice sends $x = 21{,}684$ in message one, Bob sends $e = 0$ in message two, and Alice sends $y = 657$ in the third message. Show that Bob verifies Alice's response in this case.

b. At the next iteration, Alice again sends $x = 21{,}684$ in message one, but Bob sends $e = 1$ in message two, and Alice responds with $y = 26{,}938$ in message three. Show that Bob again verifies Alice's response.

c. Determine Alice's secret S. Hint: $657^{-1} = 208 \bmod 27{,}331$.