Information Security: Principles and Practice, Second Edition
by Mark Stamp

Chapter 10

Real-World Security Protocols

The wire protocol guys don't worry about security because that's really a network protocol problem. The network protocol guys don't worry about it because, really, it's an application problem. The application guys don't worry about it because, after all, they can just use the IP address and trust the network.
— Marcus J. Ranum

In the real world, nothing happens at the right place at the right time. It is the job of journalists and historians to correct that.
— Mark Twain

10.1 Introduction

In this chapter, we'll discuss several widely used real-world security protocols. First on the agenda is the Secure Shell, or SSH, which is used for a variety of purposes. Next, we consider the Secure Socket Layer, or SSL, which is currently the most widely used security protocol for Internet transactions. The third protocol that we'll consider in detail is IPSec, which is a complex protocol with some significant security issues. Then we will discuss Kerberos, a popular authentication protocol based on symmetric key cryptography and timestamps.

We conclude the chapter with two wireless protocols, WEP and GSM. WEP is a seriously flawed security protocols, and we'll consider several well-known attacks. The final protocol we'll cover is GSM, which is used to secure mobile communications. The GSM protocol is provides an interesting case study due to the large number and wide variety of known attacks.

10.2 SSH

The Secure Shell, SSH, creates a secure tunnel which can be used to secure otherwise insecure commands. For example, in UNIX, the `rlogin` command is used for a remote login, that is, to log into a remote machine over a network. Such a login typically requires a password and `rlogin` simply sends the password in the clear, which might be observed by a snooping Trudy. By first establishing an SSH session, any inherently insecure command such as `rlogin` will be secure. That is, an SSH session provides confidentiality and integrity protection, thereby eliminating Trudy's ability to obtain passwords and other confidential information that would otherwise be sent unprotected.

SSH authentication can be based on public keys, digital certificates, or passwords. Here, we give a slightly simplified version of SSH using digital certificates.[1] The other authentication options are covered in various homework problems at the end of this chapter.

SSH is illustrated in Figure 10.1, using the following notation:

$$\begin{aligned}
\text{certificate}_A &= \text{Alice's certificate} \\
\text{certificate}_B &= \text{Bob's certificate} \\
\text{CP} &= \text{crypto proposed} \\
\text{CS} &= \text{crypto selected} \\
H &= h(\text{Alice}, \text{Bob}, \text{CP}, \text{CS}, R_A, R_B, g^a \bmod p, g^b \bmod p, g^{ab} \bmod p) \\
S_B &= [H]_{\text{Bob}} \\
K &= g^{ab} \bmod p \\
S_A &= [H, \text{Alice}, \text{certificate}_A]_{\text{Alice}}
\end{aligned}$$

As usual, h is a cryptographic hash function.

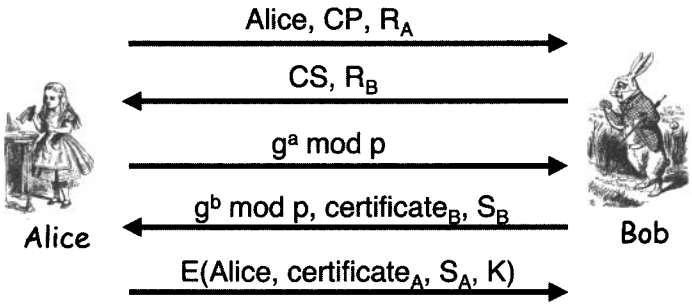

Figure 10.1: Simplified SSH

[1] In our simplified version, a few parameters have been omitted and a couple of bookkeeping messages have been eliminated.

In the first message in Figure 10.1, Alice identifies herself and she sends information regarding the crypto parameters that she prefers (crypto algorithms, key lengths, etc.), along with her nonce, R_A. In message two, Bob selects from Alice's crypto parameters and returns his selections, along with his nonce, R_B. In message three, Alice sends her Diffie-Hellman value, and in message four, Bob responds with his Diffie-Hellman value, his certificate, and S_B, which consists of a signed hash value. At this point, Alice is able to compute the key K, and in the final message, she sends an encrypted block that contains her identity, her certificate, and her signed value S_A.

In Figure 10.1, the signatures are intended to provide mutual authentication. Note that the nonce R_A is Alice's challenge to Bob, and S_B is Bob's response. That is, the nonce R_A provides replay protection, and only Bob can give the correct response since a signature is required (assuming, of course, that his private key has not been compromised). A similar argument shows that Alice is authenticated in the final message. So, SSH provides mutual authentication. The security of SSH authentication, the security of the key K, and some other quirks of SSH are considered further in the homework problems at the end of this chapter.

10.3 SSL

The mythical "socket layer" lives between the application layer and the transport layer in the Internet protocol stack, as illustrated in Figure 10.2. In practice, SSL most often deals with Web browsing, in which case the application layer protocol is HTTP and the transport layer protocol is TCP.

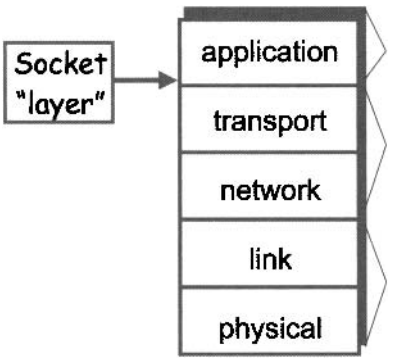

Figure 10.2: Socket Layer

SSL is the protocol of choice for the vast majority of secure transactions over the Internet. For example, suppose that you want to buy a book at

amazon.com. Before you provide your credit card information, you want to be sure you are dealing with Amazon, that is, you must authenticate Amazon. Generally, Amazon doesn't care who you are, as long as you have money. As a result, the authentication need not be mutual.

After you are satisfied that you are dealing with Amazon, you will provide private information, such as your credit card number, your address, and so on. You probably want this information protected in transit—in most cases, you want both confidentiality (to protect your privacy) and integrity protection (to assure the transaction is received correctly).

The general idea behind SSL is illustrated in Figure 10.3. In this protocol, Alice (the client) informs Bob (the server) that she wants to conduct a secure transaction. Bob responds with his certificate. Alice then needs to verify the signature on the certificate. Assuming the signature verifies, Alice will be confident that she has Bob's certificate, although she cannot yet be certain that she's talking to Bob. Then Alice will encrypt a symmetric key K_{AB} with Bob's public key and send the encrypted key to Bob. This symmetric key is used to encrypt and integrity protect subsequent communications.

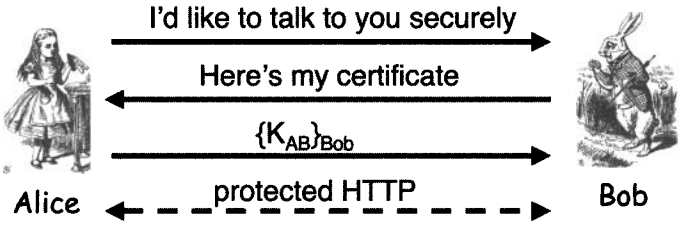

Figure 10.3: Too-Simple Protocol

The protocol in Figure 10.3 is not useful as it stands. For one thing, Bob is not explicitly authenticated and the only way Alice could possibly know she is talking to Bob is by checking to see that the encrypted data decrypts correctly. This is not a desirable situation in any security protocol. Also note that Alice is not authenticated to Bob at all, but in most cases, this is reasonable for transactions on the Internet.

In Figure 10.4, we've given a reasonably complete view of the basic SSL protocol. In this protocol,

$$
\begin{aligned}
S &= \text{the pre-master secret} \\
K &= h(S, R_A, R_B) \\
\text{msgs} &= \text{shorthand for "all previous messages"} \\
\text{CLNT} &= \text{literal string} \\
\text{SRVR} &= \text{literal string}
\end{aligned}
$$

where h is a secure hash function. The actual SSL protocol is more complex than what appears in Figure 10.4 but this simplified version is sufficient for our purposes. The complete SSL specification can be found at [271].

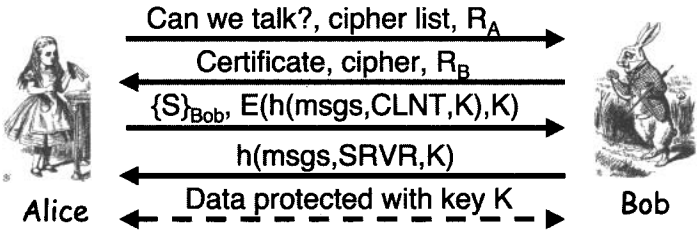

Figure 10.4: Simplified SSL

Next, we briefly discuss each message in the simplified SSL protocol given in Figure 10.4. In the first message, Alice informs Bob that she would like to establish an SSL connection, and she passes a list of ciphers that she supports, along with a nonce R_A. In the second message, Bob responds with his certificate, he selects one of the ciphers from the cipher list that Alice sent in message one, and he sends a nonce R_B.

In the third message, Alice sends the so-called pre-master secret S, which she randomly generated, along with a hash that is encrypted with the key K. In this hash, "msgs" includes all previous messages and CLNT is a literal string.[2] The hash is used as an integrity check to verify that the previous messages have been received correctly.

In the fourth message, Bob responds with a similar hash. By computing this hash herself, Alice can verify that Bob received the messages correctly, and she can authenticate Bob, since only Bob could have decrypted S, which is required to generate the key K. At this point, Alice has authenticated Bob, and Alice and Bob have established a shared session key K, which they can use to encrypt and integrity protect subsequent messages.

In reality, more than one key is derived from the hash $h(S, R_A, R_B)$. In fact, the following six quantities are generated from this hash.

- Two encryption keys, one for messages sent from the client to server, and one for messages sent from the server to the client.
- Two integrity keys, used in the same way as the encryption keys.
- Two initialization vectors (IVs), one for the client and one for the server.

[2] In this context, "msg" has nothing to do with the list of ingredients at a Chinese restaurant.

In short, different keys are used in each direction. This could help to prevent certain types of attacks where Trudy tricks Bob into doing something that Alice should have done, or vice versa.

The attentive reader may wonder why $h(\text{msgs}, \text{CLNT}, K)$ is encrypted in messages three and four. In fact, this adds no security, although it does add extra work, so it could be considered a minor flaw in the protocol.

In the SSL protocol of Figure 10.4, Alice, the client, authenticates Bob, the server, but not vice versa. With SSL, it is possible for the server to authenticate the client. If this is desired, Bob sends a "certificate request" in message two. However, this feature is generally not used, particularly in e-commerce situations, since it requires users to have valid certificates. If the server wants to authenticate the client, the server could instead require that the client enter a valid password, in which case the resulting authentication is outside the scope of the SSL protocol.

10.3.1 SSL and the Man-in-the-Middle

Hopefully, SSL prevents the man-in-the-middle, or MiM, attack illustrated in Figure 10.5. But what mechanism in SSL prevents this attack? Recall that Bob's certificate must be signed by a certificate authority. If Trudy sends her own certificate instead of Bob's, the attack will fail when Alice attempts to verify the signature on the certificate. Or, Trudy could make a bogus certificate that says "Bob," keep the private key for herself, and sign the certificate herself. Again, this will not pass muster when Alice tries to verify the signature on "Bob's" certificate (which is really Trudy's certificate). Finally, Trudy could simply send Bob's certificate to Alice, and Alice would verify the signature on this certificate. However, this is not an attack since it would not break the protocol—Alice would authenticate Bob, and Trudy would be left out in the cold.

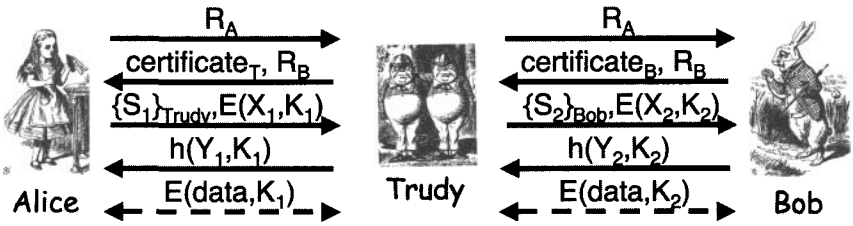

Figure 10.5: Man-in-the-Middle Attack on SSL

However, the real world is not so kind to poor Alice. Typically, SSL is used in a Web browsing session. Then, what happens when Trudy attempts a MiM attack by sending a bogus certificate to Alice? The signature on the

certificate is not valid so the attack should fail. However, Alice does not personally check the signature on the certificate—her browser does. And what does Alice's browser do when it detects a problem with a certificate? As you probably know from experience, the browser provides Alice with a warning. Does Alice heed the warning? If she's like most users, Alice ignores the warning and allows the connection to proceed.[3] Note that when Alice ignores this warning, she's opened the door to the MiM attack in Figure 10.5. Finally, it's important to realize that while this attack is a very real threat, it's not due to a flaw in the SSL protocol. Instead, it's caused by a flaw in human nature, making a patch much more problematic.

10.3.2 SSL Connections

An SSL *session* is established as shown in Figure 10.4. This session establishment protocol is relatively expensive, since public key operations are involved.

SSL was originally developed by Netscape, specifically for use in Web browsing. The application layer protocol for the Web is HTTP, and two versions of it are in common usage, HTTP 1.0 and HTTP 1.1. With version 1.0, it is not uncommon for a Web browser to open multiple parallel connections so as to improve performance. Due to the public key operations, there would be significant overhead if a new SSL session was established for each of these HTTP connections. The designers of SSL were aware of this issue, so they included an efficient protocol for opening new SSL *connections* provided that an SSL session already exists. The idea is simple—after establishing one SSL session, Alice and Bob share a session key K, which can then be used to establish new connections, thereby avoiding expensive public key operations.

The SSL connection protocol appears in Figure 10.6. The protocol is similar to the SSL session establishment protocol, except that the previously established session key K is used instead of the public key operation that are used in the session protocol.

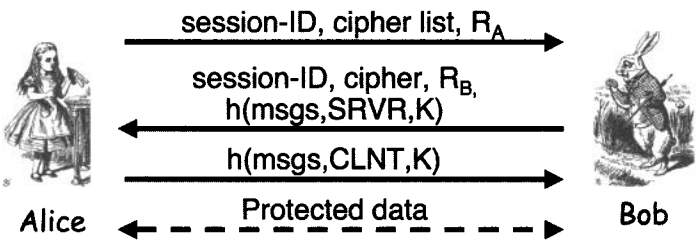

Figure 10.6: SSL Connection Protocol

[3]If possible, Alice would probably disable the warning so that she'd never get this annoying "error" message again.

The bottom line here is that in SSL, one (expensive) session is required, but then we can create any number of (cheap) connections. This is a useful feature that was designed to improve the performance of the protocol when used with HTTP 1.1.

10.3.3 SSL Versus IPSec

In the next section, we'll discuss IPSec, which is short for Internet Protocol Security. The purpose of IPSec is similar to that of SSL, namely, security over the network. However, the implementation of the two protocols is very different. For one thing, SSL is relatively simple, while IPSec is relatively complex.

It might seem logical to discuss IPSec in detail before contrasting it with SSL. However, we might get so lost in the weeds with IPSec that we'd completely lose sight of SSL. So instead of waiting until after we discuss IPSec to contrast the two protocols, we'll do so beforehand. You might consider this a partial preview of IPSec.

The most obvious difference between SSL and IPSec is that the two protocols operate at different layers of the protocol stack. SSL (and its twin,[4] the IEEE standard known as TLS), both live at the socket layer. As a result, SSL resides in user space. IPSec, on the other hand, lives at the network layer and is therefore not directly accessible from user space—it's in the domain of the operating system. When viewed from a high level, this is the fundamental distinction between SSL and IPSec.

Both SSL and IPSec provide encryption, integrity protection, and authentication. SSL is relatively simple and well designed, whereas IPSec is complex and, as a result, includes some significant flaws.

Since IPSec is part of the OS, it must be built-in at that level. In contrast, SSL is part of user space, so it requires nothing special of the OS. IPSec also requires no changes to applications, since all of the security magically happens at the network layer. On the other hand, developers have to make a conscious decision to use SSL.

SSL was built for Web application early on, and its primary use remains secure Web transactions. IPSec is often used to secure a virtual private network, or VPN, an application that creates a secure tunnel between the endpoints. Also, IPSec is required in IP version 6 (IPv6), so if IPv6 ever takes over the world, IPSec will be ubiquitous.

There is, understandably, a reluctance to retrofit applications for SSL. There is, also understandably, a reluctance to use IPSec due to its complexity (which creates some challenging implementation issues). The net result is that the Net is less secure than it should be.

[4] They are fraternal twins, not identical twins.

10.4 IPSec

Figure 10.7 illustrates the primary logical difference between SSL and IPSec, that is, one lives at the socket layer (SSL), while the other resides at the network layer (IPSec). As mentioned above, the major advantage of IPSec is that it's essentially transparent to applications. However, IPSec is a complex protocol, which can perhaps best be described as over-engineered.

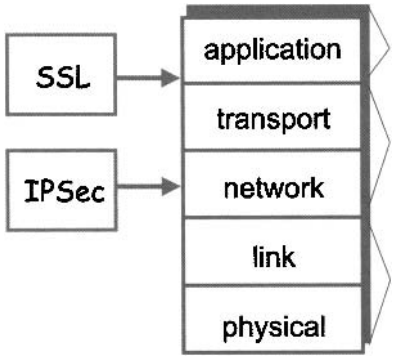

Figure 10.7: IPSec

IPSec has many dubious features, which makes implementation difficult. Also, IPSec has some flaws, probably as a direct result of its complexity. In addition, there are interoperability issues, due to the complexity of the IPSec specification, which seems to run contrary to the point of having a standard. Another complicating factor is that the IPSec specification is split into three pieces, to be found in RFC 2407 [237], RFC 2408 [197], and RFC 2409 [140], and these RFCs were written by disjoint sets of authors using different terminology.

The two main parts to IPSec are

- The Internet Key Exchange, or IKE, which provides for mutual authentication and a session key. There are two phases of IKE, which are analogous to SSL sessions and connections.
- The Encapsulating Security Payload and Authentication Header, or ESP/AH, which together make up the second part of IPSec. ESP[5] provides encryption and integrity protection to IP packets, whereas AH provides integrity only.

Technically, IKE is a standalone protocol that could live a life separate from ESP/AH. However, since IKE's only application in the real world seems to

[5]Contrary to what you are thinking, this protocol cannot read your mind.

be in conjunction with IPSec, we lump them together under the name IPSec. The comment about IPSec being over-engineered applies primarily to IKE. The developers of IKE apparently thought they were creating the Swiss army knife of security protocols—a protocol that would be used to solve every conceivable authentication problem. This explains the multitude of options and features built into IKE. However, since IKE is only used with IPSec, any features or options that are not directly relevant to IPSec are simply extraneous.

First, we'll consider IKE, then ESP/AH. IKE, the more complex of the two, consists of two phases—cleverly called Phase 1 and Phase 2. Phase 1 is the more complex of the two. In Phase 1, a so-called IKE security association, or IKE-SA, is established, while in Phase 2, an IPSec security association, IPSec-SA, is established. Phase 1 corresponds to an SSL session, whereas Phase 2 is comparable to an SSL connection. In IKE, both Phase 1 and Phase 2 must occur before we can do ESP/AH.

Recall that SSL connections serve a specific and useful purpose—they make SSL more efficient when HTTP 1.0 is used. But, unlike SSL, in IPSec there is no obvious need for two phases. And if multiple Phase 2s do not occur (and they typically do not), then it would be more efficient to just require Phase 1 with no Phase 2. However, this is not an option. Apparently, the developers of IKE believed that their protocol was so self-evidently wonderful that users would want to do multiple Phase 2s (one for IPSec, another for something else, another for some other something else, and so on). This is our first example of over-engineering in IPSec, and it won't be the last.

In IKE Phase 1, there are four different key options:

- Public key encryption (original version)
- Public key encryption (improved version)
- Digital signature
- Symmetric key

For each of these key options there is a main mode and an aggressive mode. As a result, there are a staggering eight different versions of IKE Phase 1. Do you need any more evidence that IPSec is over-engineered?

You may be wondering why there are public key encryption and digital signature options in Phase 1. Surprisingly, the answer is not over-engineering. Alice always knows her own private key, but she may not know Bob's public key. With the signature version of IKE Phase 1, Alice does not need to have Bob's public key in hand to start the protocol. In any protocol that uses public key crypto, Alice will need Bob's public key to complete the protocol, but in the signature mode, she can simultaneously begin the protocol and search for Bob's public key. In contrast, in the public key encryption modes,

Alice needs Bob's public key immediately, so she must first find Bob's key before she can begin the protocol. So, there could be an efficiency gain with the signature option.

We'll discuss six of the eight Phase 1 variants, namely, digital signatures (main and aggressive modes), symmetric key (main and aggressive modes), and public key encryption (main and aggressive). We'll consider the original version of public key encryption, since it's slightly simpler, although less efficient, than the improved version.

Each of the Phase 1 variants use an ephemeral Diffie-Hellman key exchange to establish a session key. The benefit of this approach is that it provides perfect forward secrecy (PFS). For each of the variants we discuss, we'll use the following Diffie-Hellman notation. Let a be Alice's (ephemeral) Diffie-Hellman exponent and let b be Bob's (ephemeral) Diffie-Hellman exponent. Let g be the generator and p the prime. Recall that p and g are public.

10.4.1 IKE Phase 1: Digital Signature

The first Phase 1 variant that we'll consider is digital signature, main mode. This six message protocol is illustrated in Figure 10.8, where

$$\begin{aligned}
\text{CP} &= \text{crypto proposed}\\
\text{CS} &= \text{crypto selected}\\
\text{IC} &= \text{initiator cookie}\\
\text{RC} &= \text{responder cookie}\\
K &= h(\text{IC}, \text{RC}, g^{ab} \bmod p, R_A, R_B)\\
\text{SKEYID} &= h(R_A, R_B, g^{ab} \bmod p)\\
\text{proof}_A &= [h(\text{SKEYID}, g^a \bmod p, g^b \bmod p, \text{IC}, \text{RC}, \text{CP}, \text{“Alice”})]_{\text{Alice}}
\end{aligned}$$

Here, h is a hash function and proof_B is analogous to proof_A.

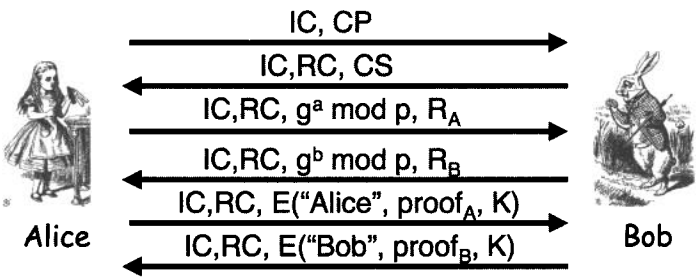

Figure 10.8: Digital Signature, Main Mode

Let's briefly consider each of the six messages that appear in Figure 10.8. In the first message, Alice provides information on the ciphers that she supports and other crypto related information, along with a so-called cookie.[6] In message two, Bob selects from Alice's crypto proposal and sends the cookies, which serve as an identifier for the remainder of the messages in the protocol. The third message includes a nonce and Alice's Diffie-Hellman value. Bob responds similarly in message four, providing a nonce and his Diffie-Hellman value. In the final two messages, Alice and Bob authenticate each other using digital signatures.

Recall that an attacker, Trudy, is said to be passive if she can only observe messages sent between Alice and Bob. In contrast, if Trudy is an active attacker, she can also insert, delete, alter, and replay messages. For the protocol in Figure 10.8, a passive attacker cannot discern Alice or Bob's identity. So this protocol provides anonymity, at least with respect to passive attacks. Does this protocol also provide anonymity in the case of an active attack? This question is considered in Problem 27, which means that the answer is not to be found here.

Each key option has a main mode and an aggressive mode. The main modes are supposed to provide anonymity, while the aggressive modes are not. Anonymity comes at a price—aggressive mode only requires three messages, as opposed to six messages for main mode.

The aggressive mode version of the digital signature key option appears in Figure 10.9. Note that there is no attempt to hide the identities of Alice or Bob, which simplifies the protocol considerably. The notation in Figure 10.9 is the same as that used in Figure 10.8.

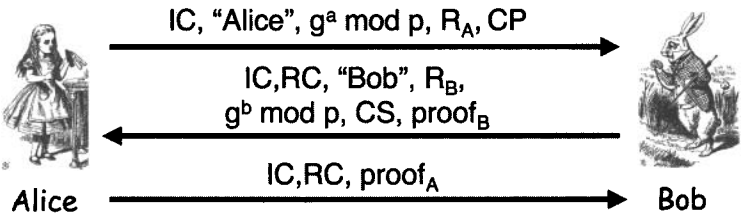

Figure 10.9: Digital Signature, Aggressive Mode

One subtle difference between digital signature main and aggressive modes is that in main mode it is possible to negotiate the values of g and p as part of the "crypto proposed" and "crypto accepted" messages. However, this is not the case in aggressive mode, since the Diffie-Hellman value $g^a \bmod p$ is sent in the first message.

[6]Not to be confused with Web cookies or chocolate chip cookies. We have more to say about these IPSec cookies in Section 10.4.4, below.

As per the appropriate RFCs, for each key option main mode MUST be implemented, while aggressive mode SHOULD be implemented. In [162], the authors interpret this to mean that if aggressive mode is not implemented, "you should feel guilty about it."

10.4.2 IKE Phase 1: Symmetric Key

The next version of Phase 1 that we'll consider is the symmetric key option—both main mode and aggressive mode. As above, the main mode is a six-message protocol, where the format is formally the same as in Figure 10.8, above, except that the notation is interpreted as follows.

$$\begin{aligned} K_{AB} &= \text{symmetric key shared in advance} \\ K &= h(\text{IC}, \text{RC}, g^{ab} \bmod p, R_A, R_B, K_{AB}) \\ \text{SKEYID} &= h(K, g^{ab} \bmod p) \\ \text{proof}_A &= h(\text{SKEYID}, g^a \bmod p, g^b \bmod p, \text{IC}, \text{RC}, \text{CP}, \text{Alice}) \end{aligned}$$

Again, the purported advantage of the complex six-message main mode over the corresponding aggressive mode is that main mode is supposed to provide anonymity. But there is a Catch-22 in this main mode. Note that in message five Alice sends her identity, encrypted with key K. But Bob has to use the key K_{AB} to determine K. So Bob has to know to use the key K_{AB} *before* he knows that he's talking to Alice. However, Bob is a busy server who deals with lots of users (Alice, Charlie, Dave, ...). How can Bob possibly know that he is supposed to use the key he shares with Alice before he knows he's talking to Alice? The answer is that he cannot, at least not based on any information available within the protocol itself.

The developers of IPSec recognized this snafu. And their solution? Bob is to rely on the IP address to determine which key to use. So, Bob must use the IP address of incoming packets to determine who he's talking to before he knows who he's talking to (or something like that...). The bottom line is that Alice's IP address acts as her identity.

There are a couple of problems with this approach. First, Alice must have a static IP address—this mode fails if Alice's IP address changes. A more fundamental issue is that the protocol is complex and uses six messages, presumably to hide identities. But the protocol fails to hide identities, unless you consider a static IP address to be secret. So it would seem pointless to use symmetric key main mode instead of the simpler and more efficient aggressive mode, which we describe next.[7]

IPSec symmetric key aggressive mode follows the same format as the digital signature aggressive mode in Figure 10.9, with the key and signature

[7]Of course, main mode MUST be implemented, while aggressive mode SHOULD be implemented. Go figure.

computed as in symmetric key main mode. As with the digital signature variant, the main difference from main mode is that aggressive mode does not attempt to hide identities. Since symmetric key main mode also fails to effectively hide Alice's identity, this is not a serious limitation of aggressive mode in this case.

10.4.3 IKE Phase 1: Public Key Encryption

Next, we'll consider the public key encryption version of IKE Phase 1, both main and aggressive modes. We've already seen the digital signature versions. In the main mode of the encryption version, Alice must know Bob's public key in advance and vice versa. Although it would be possible to exchange certificates, that would reveal the identities of Alice and Bob, defeating the primary advantage of main mode. So an assumption here is that Alice and Bob have access to each other's certificates, without sending them over the network.

The public key encryption main mode protocol is given in Figure 10.10, where the notation is as in the previous modes, except

$$\begin{aligned} K &= h(\text{IC}, \text{RC}, g^{ab} \bmod p, R_A, R_B) \\ \text{SKEYID} &= h(R_A, R_B, g^{ab} \bmod p) \\ \text{proof}_A &= h(\text{SKEYID}, g^a \bmod p, g^b \bmod p, \text{IC}, \text{RC}, \text{CP}, \text{"Alice"}) \end{aligned}$$

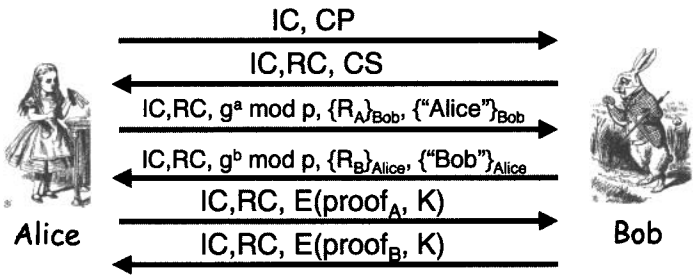

Figure 10.10: Public Key Encryption, Main Mode

Public key encryption, aggressive mode, appears in Figure 10.11, where the notation is similar to main mode. Interestingly, unlike the other aggressive modes, public key encryption aggressive mode allows Alice and Bob to remain anonymous. Since this is the case, is there any possible advantage of main mode over aggressive mode? The answer is yes, but it's a minor issue (see Problem 25 at the end of the chapter).

There is an interesting security quirk that arises in the public key encryption versions—both main and aggressive modes. For simplicity, let's consider

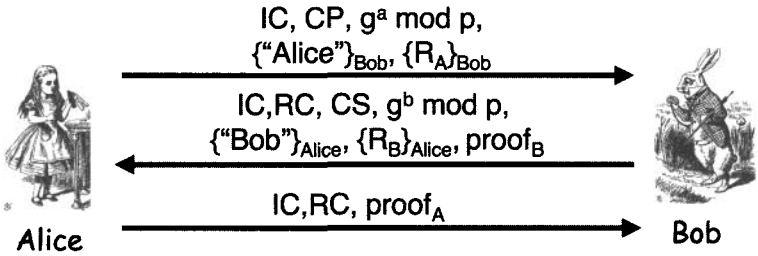

Figure 10.11: Public Key Encryption, Aggressive Mode

aggressive mode. Suppose Trudy generates Diffie-Hellman exponents a and b and random nonces R_A and R_B. Then Trudy can compute all of the remaining quantities that appear in the protocol in Figure 10.11, namely, $g^{ab} \bmod p$, K, SKEYID, proof_A, and proof_B. The reason that Trudy can do this is because the public keys of Alice and Bob are public.

Why would Trudy go to the trouble of generating all of these values? Once Trudy has done so, she can create an entire conversation that appears to be a valid IPSec transaction between Alice and Bob, as indicated in Figure 10.12. Amazingly, this conversation appears to be valid to any observer, including Alice and/or Bob!

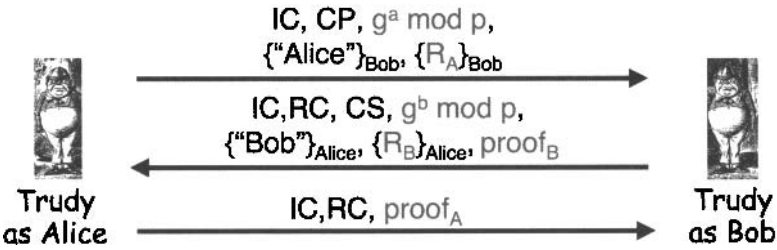

Figure 10.12: Trudy Making Mischief

Note that in Figure 10.12, Trudy is playing the roles of both Alice and Bob. Here, Trudy does not convince Bob that she's Alice, she does not convince Alice that she's Bob, nor does she determine a session key used by Alice and Bob. So, this is a very different kind of attack than we have previously seen. Or maybe it's not an attack at all.

But surely, the fact that Trudy can create a fake conversation that appears to be a legitimate connection between Alice and Bob is a security flaw. Surprisingly, in this mode of IPSec it is considered a security feature, which goes by the name of *plausible deniability*. A protocol that includes plausible deniability allows Alice and Bob to deny that a conversation ever took place,

since anyone could have faked the whole thing. In some situations, this could be a desirable feature. On the other hand, in some situations it might be a problem. For example, if Alice makes a purchase from Bob, she could later repudiate it, unless Bob also required a digital signature from Alice.

10.4.4 IPSec Cookies

The cookies IC and RC that appear in the IPSec protocols above are officially known as "anti-clogging tokens" in the relevant RFCs. These IPSec cookies have no relation to Web cookies, which are used to maintain state across HTTP sessions. Instead, the stated purpose of IPSec cookies is to make denial of service, or DoS, attacks more difficult.

Consider TCP SYN flooding, which is a prototypical DoS attack. Each TCP SYN request causes the server to do a little work (create a SEQ number, for example) and to keep some amount of state. That is, the server must remember the so-called half-open connection so that it can complete the connection when the corresponding ACK arrives in the third step of the three-way handshake. It is this keeping of state that an attacker can exploit to create a DoS. If the attacker bombards a server with a large number of SYN packets and never completes the resulting half-open connections, the server will eventually deplete its resources. When this occurs, the server cannot handle legitimate SYN requests and a DoS results.

To reduce the threat of DoS in IPSec, the server Bob would like to remain stateless as much as possible. The IPSec cookies are supposed to help Bob remain stateless. However, they clearly fail to achieve their design goal. In each of the main mode protocols, Bob must remember the crypto proposal, CP, from message one, since it is required in message six when Bob computes proof_B. Consequently, Bob must keep state beginning with the first message. The IPSec cookies therefore offer no significant DoS protection.

10.4.5 IKE Phase 1 Summary

Regardless of which of the eight versions is used, successful completion of IKE Phase 1 results in mutual authentication and a shared session key. This is known an an IKE Security Association (IKE-SA).

IKE Phase 1 is computationally expensive in any of the public key modes, and the main modes also require six messages. Developers of IKE assumed that it would be used for lots of things, not just IPSec (which explains the over-engineering). So they included an inexpensive Phase 2, which must be used after the IKE-SA has been established in Phase 1. That is, a separate Phase 2 is required for each different application that will make use of the IKE-SA. However, if IKE is only used for IPSec (as is the case in practice), the potential efficiency provided by multiple Phase 2s is not realized.

IKE Phase 2 is used to establish a so-called IPSec Security Association, or IPSec-SA. The IKE Phase 1 is more or less equivalent to establishing an SSL session, whereas IKE Phase 2 is more or less equivalent to establishing an SSL connection. Again, the designers of IPSec wanted to make it as flexible as possible, since they assumed it would be used for lots of things other than IPSec. In fact, IKE could conceivably be used for lots of things other than IPSec, however, in practice, it's not.

10.4.6 IKE Phase 2

IKE Phase 2 is mercifully simple—at least in comparison to Phase 1. Before IKE Phase 2 can occur, IKE Phase 1 must be completed, in which case a shared session key K, the IPSec cookies, IC, RC, and the IKE-SA have all been established and are known to Alice and Bob. Given that this is the case, the IKE Phase 2 protocol appears in Figure 10.13, where the following holds true.

- The crypto proposal includes ESP or AH (discussed below). This is where Alice and Bob decide whether to use ESP or AH.
- SA is an identifier for the IKE-SA established in Phase 1.
- The hashes numbered 1, 2, and 3 depend on SKEYID, R_A, R_B, and the IKE SA from Phase 1.
- The keys are derived from KEYMAT = $h(\text{SKEYID}, R_A, R_B, \text{junk})$, where the "junk" is known to all (including an attacker).
- The value of SKEYID depends on the Phase 1 key method.
- Optionally, PFS can be employed, using an ephemeral Diffie-Hellman exchange.

Note that R_A and R_B in Figure 10.13 are not the same as those from IKE Phase 1. As a result, the keys generated in each Phase 2 differ from the Phase 1 key and from each other.

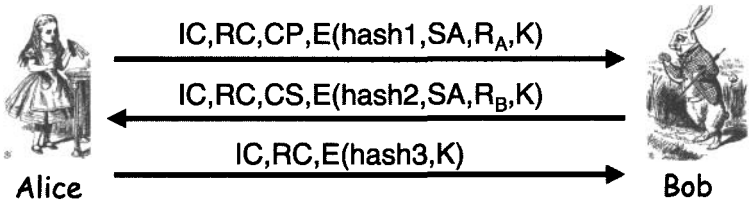

Figure 10.13: IKE Phase 2

After completing IKE Phase 1, we have established an IKE-SA, and after completing IKE Phase 2, we have established an IPSec-SA. After Phase 2, both Alice and Bob have been authenticated and they have a shared session key for use in the current connection.

Recall that in SSL, once we completed mutual authentication and had established a session key, we were done. Since SSL deals with application layer data, we simply encrypt and integrity protect in a standard way. In SSL, the network is transparent to Alice and Bob because SSL lives at the socket layer—which is really part of the application layer. This is one advantage to dealing with application layer data.

In IPSec, protecting the data is not so straightforward. Assuming IPSec authentication succeeds and we establish a session key, then we need to protect IP datagrams. The complication here is that protection must occur at the network layer. But before we discuss this issue in detail, we need to consider IP datagrams from the perspective of IPSec.

10.4.7 IPSec and IP Datagrams

An IP datagram consists of a header and data. The IP header is illustrated in the Appendix in Figure A-5. If the `options` field is empty (as it usually is), then the IP header consists of 20 bytes. For the purposes of IPSec, one important point is that routers must see the destination address in the IP header so that they can route the packet. Most other header fields are also used in conjunction with routing the packet. Since the routers do not have access to the session key, we cannot encrypt the IP header.

A second crucial point is that some of the fields in the IP header change as the packet is forwarded. For example, the TTL field—which contains the number of hops remaining before the packet dies—is decremented by each router that handles the packet. Since the session key is not known to the routers, any header fields that change cannot be integrity protected. In IPSec-speak, the header fields that can change are known as mutable fields.

Next, we look inside an IP datagram. Consider, for example, a Web browsing session. The application layer protocol for such traffic is HTTP, and the transport layer protocol is TCP. In this case, IP encapsulates a TCP packet, which encapsulates an HTTP packet as is illustrated in Figure 10.14. The point here is that, from the perspective of IP (and hence, IPSec), the data includes more than application layer data. In this example, the "data" includes the TCP and HTTP headers, as well as the application layer data. We'll see why this is relevant below.

As previously mentioned, IPSec uses either ESP or AH to protect an IP datagram. Depending on which is selected, an ESP header or an AH header is included in an IPSec-protected datagram. This header tells the recipient to treat this as an ESP or AH packet, not as a standard IP datagram.

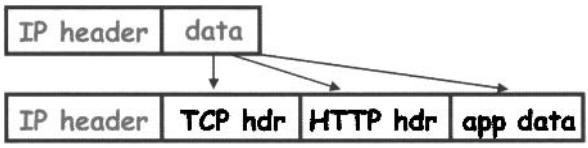

Figure 10.14: IP Datagram

10.4.8 Transport and Tunnel Modes

Independent of whether ESP or AH is used, IPSec employs either *transport mode* or *tunnel mode*. In transport mode, as illustrated in Figure 10.15, the new ESP/AH header is sandwiched between the IP header and the data. Transport mode is more efficient since it adds a minimal amount of additional header information. Note that in transport mode the original IP header remains intact. The downside of transport mode is that a passive attacker can see the headers. So, if Trudy observes an IPSec protected conversation between Alice and Bob where transport mode is used, the headers will reveal that Alice and Bob are communicating.[8]

Transport mode is designed for host-to-host communication, that is, when Alice and Bob are communicating directly with each other using IPSec. This is illustrated in Figure 10.16.

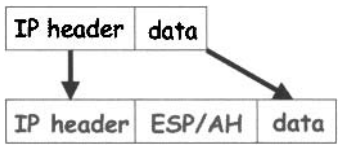

Figure 10.15: IPSec Transport Mode

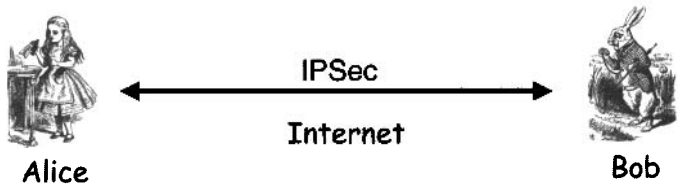

Figure 10.16: IPSec from Host-to-Host

In tunnel mode, as illustrated in Figure 10.17, the entire IP packet is encapsulated in a new IP packet. One advantage of this approach is that the

[8]Recall that we cannot encrypt the header.

original IP header is no longer visible to an attacker—assuming the packet is encrypted. However, if Alice and Bob are communicating directly with each other, the new IP header will be the same as the encapsulated IP header, so hiding the original header would be pointless. However, IPSec is often used from firewall to firewall, not from host to host. That is, Alice's firewall and Bob's firewall communicate using IPSec, not Alice and Bob directly. Suppose IPSec is being used from firewall to firewall. Using tunnel mode, the new IP header will only reveal that the packet is being sent between Alice's firewall and Bob's firewall. So, if the packet is encrypted, Trudy would know that Alice's and Bob's firewalls are communicating, but she would not know which specific hosts behind the firewalls are communicating.

Tunnel mode was designed for firewall-to-firewall communication. Again, when tunnel mode is used from firewall to firewall—as illustrated in Figure 10.18—Trudy does not know which hosts are communicating. The disadvantage of tunnel mode is the overhead of an additional IP header.

Figure 10.17: IPSec Tunnel Mode

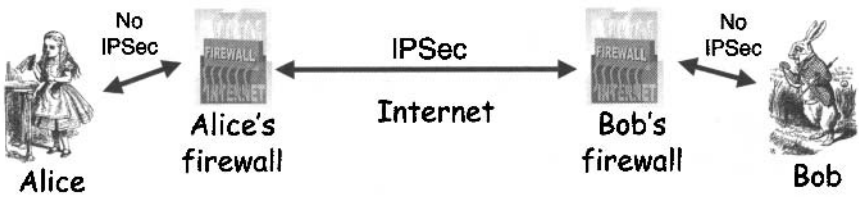

Figure 10.18: IPSec from Firewall to Firewall

Technically, transport mode is not necessary, since we could encapsulate the original IP packet in a new IPSec packet, even in the host-to-host case. For firewall-to-firewall protected traffic, tunnel mode is necessary, as we must preserve the original IP header so that the destination firewall can route the packet to the destination host. But transport mode is more efficient, which makes it preferable when traffic is protected from host to host.

10.4.9 ESP and AH

Once we've decided whether to use transport mode or tunnel mode, then we must (finally) consider the type of protection we actually want to apply to

the IP datagrams. The choices are confidentiality, integrity, or both. But we also must consider the protection, if any, to apply to the header. In IPSec, the only choices are AH and ESP. So, what protection options do each of these provide?

AH, the Authentication Header, provides integrity only, that is, AH provides no encryption. The AH integrity protection applies to everything beyond the IP header and some fields of the header. As previously mentioned, not all fields of the IP header can be integrity protected (TTL, for example). AH classifies IP header fields as mutable or immutable, and it applies its integrity protection to all of the immutable fields.

In ESP, the Encapsulating Security Payload, both integrity and confidentiality are required. Both the confidentiality and integrity protection are applied to everything beyond the IP header, that is, the "data" from the perspective of IP. No protection is applied to the IP header

Encryption is required in ESP. However, there is a trick whereby ESP can be used for integrity only. In ESP, Alice and Bob negotiate the cipher that they will use. One of the ciphers that MUST be supported is the NULL cipher, described in RFC 2410 [123]. Here are some excerpts from this unusual RFC.

- NULL encryption is a block cipher, the origins of which appear to be lost in antiquity.
- Despite rumors, there is no evidence that NSA suppressed publication of this algorithm.
- Evidence suggests it was developed in Roman times as an exportable version of Caesar's cipher.
- NULL encryption can make use of keys of varying length.
- No IV is required.
- NULL encryption is defined by $\text{Null}(P, K) = P$ for any plaintext P and any key K.

This RFC proves that security people are strange.[9]

In ESP, if the NULL cipher is selected then no encryption is applied, but the data is integrity protected. This case looks suspiciously similar to AH. So, why does AH exist?

There are three reasons given to justify the existence of AH. As previously noted, the IP header can't be encrypted since routers must see the header to route packets. But AH does provide integrity protection to the immutable

[9] As if you didn't already know that.

fields in the IP header, whereas ESP provides no protection to the header. That is, AH provides slightly more integrity protection than ESP/NULL.

A second reason for the existence of AH is that ESP encrypts everything beyond the IP header, provided a non-NULL cipher is selected. If ESP is used and the packet is encrypted, a firewall can't look inside the packet to, for example, examine the TCP header. Perhaps surprisingly, ESP with NULL encryption doesn't solve this problem. When the firewall sees the ESP header, it will know that ESP is being used. However, the header does not tell the firewall that the NULL cipher is used—that was negotiated between Alice and Bob and is not included in the header. So, when a firewall sees that ESP is used, it has no way to know whether the TCP header is encrypted or not. In contrast, when a firewall sees that AH is used, it knows that nothing is encrypted.

Neither of these reasons for the existence of AH is particularly persuasive. The designers of AH/ESP could have made minor modifications to the protocol so that ESP alone could overcome these drawbacks. But there is a more convincing reason given for the existence of AH. At one meeting where the IPSec standard was being developed, "someone from Microsoft gave an impassioned speech about how AH was useless ..." and "... everyone in the room looked around and said, Hmm. He's right, and we hate AH also, but if it annoys Microsoft let's leave it in since we hate Microsoft more than we hate AH" [162]. So now you know the rest of the story.

10.5 Kerberos

In Greek mythology, Kerberos is a three-headed dog that guards the entrance to Hades.[10] In security, Kerberos is a popular authentication protocol that uses symmetric key cryptography and timestamps. Kerberos originated at MIT and is based on work by Needham and Schroeder [217]. Whereas SSL and IPSec are designed for the Internet, Kerberos is designed for a smaller scale, such as on a local area network (LAN) or within a corporation.

Suppose we have N users, where each pair needs to be able to authenticate each other. If our authentication protocol is based on public key cryptography, then each user requires a public-private key pair and, consequently, N key pairs are needed. On the other hand, if our authentication protocol is based on symmetric keys, it would appear that each pair of users must share a symmetric key, in which case $N(N-1)/2 \approx N^2$ keys are required. Consequently, authentication based on symmetric keys doesn't scale. However, by relying on a Trusted Third Party (TTP), Kerberos only requires N symmetric keys for N users. Users do not share keys with each other. Instead each user shares one key with the KDC, that is, Alice and the KDC share K_A, Bob

[10]The authors of [162] ask, "Wouldn't it make more sense to guard the exit?"

and the KDC share K_B, Carol and the KDC share K_C, and so on. Then, the KDC acts as a go-between that enables any pair of users to communicate securely with each other. The bottom line is that Kerberos uses symmetric keys in a way that does scale.

The Kerberos TTP is a security critical component that must be protected from attack. This is certainly a security issue, but in contrast to a system that uses public keys, no public key infrastructure (PKI) is required.[11] In essence, the Kerberos TTP plays a similar role as a certificate authority in a public key system.

The Kerberos TTP is known as the *key distribution center*, or KDC.[12] Since the KDC acts as a TTP, if it's compromised, the security of the entire system is compromised.

As noted above, the KDC shares a symmetric key K_A with user Alice, and it shares a symmetric key K_B with Bob, and so on. The KDC also has a master key K_{KDC}, which is known only to the KDC. Although it might seem senseless to have a key that only the KDC knows, we'll see that this key plays a critical role. In particular, the key K_{KDC} allows the KDC to remain stateless, which eliminates most denial of service attacks. A stateless KDC is a major security feature of Kerberos.

Kerberos is used for authentication and to establish a session key that can subsequently be used for confidentiality and integrity. In principle, any symmetric cipher can be used with Kerberos. However, in practice, it seems the crypto algorithm of choice is the Data Encryption Standard (DES).

In Kerboros-speak, the KDC issues various types of *tickets*. Understanding these tickets is critical to understanding Kerberos. A ticket contains the keys and other information required to access network resource. One special ticket that the KDC issues is the all-important *ticket-granting ticket*, or TGT. A TGT, which is issued when a user initially logs into the system, acts as the user's credentials. The TGT is then used to obtain (ordinary) tickets that enable access to network resources. The use of TGTs is crucial to the statelessness of Kerberos.

Each TGT contains a session key, the user ID of the user to whom the TGT is issued, and an expiration time. For simplicity, we'll ignore the expiration time, but it's worth noting that TGTs don't last forever. Every TGT is encrypted with the key K_{KDC}. Recall that only the KDC knows the key K_{KDC}. As a result, a TGT can only be read by the KDC.

Why does the KDC encrypt a user's TGT with a key that only the KDC knows and then send the result to the user? The alternative would be for the KDC to maintain a database of which users are logged in, their session keys, etc. That is, the TGT would have to maintain state. In effect, TGTs

[11] As we discussed in Chapter 4, PKI presents a substantial challenge in practice.

[12] The most difficult part about Kerberos is keeping track of all of the acronyms. There are a lot more acronyms to come—we're just getting warmed up.

provides a simple, effective, and secure way to distribute this database to the users. Then when, say, Alice presents her TGT to the KDC, the KDC can decrypt it and, voila, it remembers everything it needs to know about Alice.[13] The role of the TGT will become clear below. For now, just note that TGTs are a clever design feature of Kerberos.

10.5.1 Kerberized Login

To understand Kerberos, let's first consider how a "Kerberized" login works, that is, we'll examine the steps that occur when Alice logs in to a system where Kerberos is used for authentication. As on most systems, Alice first enters her username and password. In Kerberos, Alice's computer then derives the key K_A from Alice's password, where K_A is the key that Alice and the KDC share. Alice's computer uses K_A to obtain Alice's TGT from the KDC. Alice can then use her TGT (i.e., her credentials) to securely access network resources. Once Alice has logged in, all of the security is automatic and takes place behind the scenes, without any additional involvement by Alice.

A Kerberized login is illustrated in Figure 10.19, where the following notation is used.

- The key K_A is derived as $K_A = h(\text{Alice's password})$
- The KDC creates the session key S_A
- Alice's computer uses K_A to obtain S_A and the TGT; then Alice's computer forgets K_A
- TGT $= E(\text{"Alice"}, S_A; K_{\text{KDC}})$

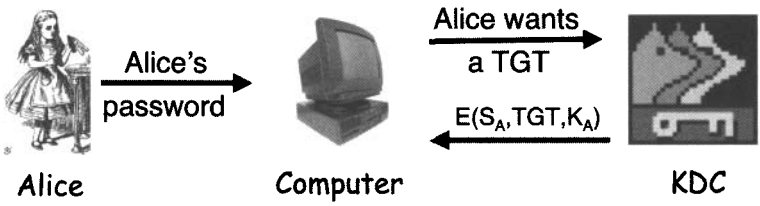

Figure 10.19: Kerberized Login

[13] Your hapless author's ill-fated startup company had a similar situation, i.e., a database of customer security-related information that had to be maintained (assuming the company had ever actually had any customers, that is). Instead of creating a security-critical database, the company chose to encrypt each user's information with a key known only to the company, then distribute this encrypted data to the appropriate user. Users then had to present this encrypted data before they could access any security-related features of the system. This is essentially the same trick used in Kerberos TGTs.

One major advantage to the Kerberized login is that the entire security process (beyond the password entry) is transparent to Alice. The major disadvantage is that the reliance on the security of the KDC is total.

10.5.2 Kerberos Ticket

Once Alice's computer receives its TGT, it can then use the TGT to request access to network resources. For example, suppose that Alice wants to talk to Bob. Then Alice's computer presents its TGT to the KDC, along with an authenticator. The authenticator is an encrypted timestamp that serves to avoid a replay. After the KDC verifies Alice's authenticator, it responds with a "ticket to Bob." Alice's computer then uses this ticket to Bob to securely communicate directly with Bob's computer. Alice's acquisition of the ticket to Bob is illustrated in Figure 10.20, where the following notation is used.

$$\begin{aligned} \text{REQUEST} &= (\text{TGT}, \text{authenticator}) \\ \text{authenticator} &= E(\text{timestamp}, S_A) \\ \text{REPLY} &= E(\text{"Bob"}, K_{AB}, \text{ticket to Bob}; S_A) \\ \text{ticket to Bob} &= E(\text{"Alice"}, K_{AB}; K_B) \end{aligned}$$

In Figure 10.20, the KDC obtains the key S_A from the TGT and uses this key to verify the timestamp. Also, the key K_{AB} is the session key that Alice and Bob will use for their session.

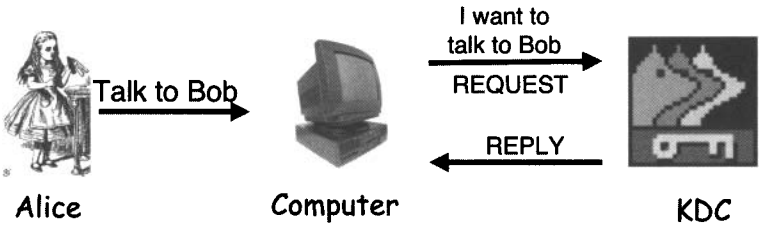

Figure 10.20: Alice Gets Ticket to Bob

Once Alice has obtained the "ticket to Bob," she can then securely communicate with Bob. This process is illustrated in Figure 10.21, where the ticket to Bob is as above and

$$\text{authenticator} = E(\text{timestamp}, K_{AB}).$$

Note that Bob decrypts "ticket to Bob" with his key K_B to obtain K_{AB}, which he then uses to verify the timestamp. The key K_{AB} is also used to protect the confidentiality and integrity of the subsequent conversation between Alice and Bob.

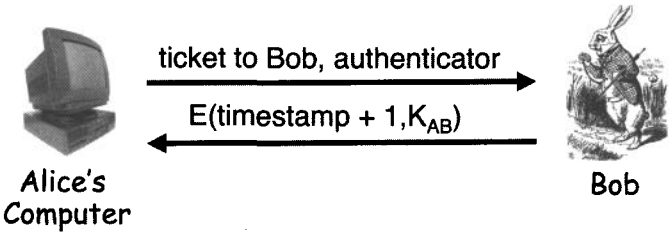

Figure 10.21: Alice Contacts Bob

Since timestamps are used for replay prevention, Kerberos minimizes the number of messages that must be sent. As we mentioned in the previous chapter, the primary drawback to using timestamps is that time becomes a security-critical parameter. Another issue with timestamps is that we can't expect all clocks to be perfectly synchronized and therefore some clock skew must be tolerated. In Kerberos, this clock skew is by default set to five minutes, which seems like an eternity in a networked world.

10.5.3 Kerberos Security

Recall that, when Alice logs in, the KDC sends $E(S_A, \text{TGT}; K_A)$ to Alice, where $\text{TGT} = E(\text{"Alice"}, S_A; K_{\text{KDC}})$. Since the TGT is encrypted with the key K_{KDC}, why is the TGT encrypted again with the key K_A? The answer is that this is a minor flaw in Kerberos, since it's extra work that provides no additional security. If the key K_{KDC} is compromised, the entire security of the system is broken, so there is no added benefit to encrypting the TGT again after it's already encrypted with K_{KDC}.

Notice that, in Figure 10.20, Alice remains anonymous in the REQUEST. This is a nice security feature that is a side benefit of the fact that the TGT is encrypted with the key K_{KDC}. That is, the KDC does not need to know who is making the REQUEST before it can decrypt the TGT, since all TGTs are encrypted with K_{KDC}. Anonymity with symmetric keys can be difficult, as we saw with the IPSec symmetric key main mode. But, in this part of Kerberos, anonymity is easy.

In the Kerberos example above, why is "ticket to Bob" sent to Alice, when Alice simply forwards it on to Bob? Apparently, it would be more efficient to have the KDC send the ticket directly to Bob, and the designers of Kerberos were certainly concerned with efficiency (e.g., they use timestamps instead of nonces). However, if the ticket to Bob arrives at Bob before Alice initiates contact, then Bob would have to remember the key K_{AB} until it's needed. That is, Bob would need to maintain state. Statelessness is an important feature of Kerberos.

Finally, how does Kerberos prevent replay attacks? Replay prevention relies on the timestamps that appear in the authenticators. But there is still an issue of replay within the clock skew. To prevent such replay attacks, the KDC would need to remember all timestamps received within the clock skew interval. However, most Kerberos implementations apparently don't bother to do this [162].

Before departing the realm of Kerberos, we consider a design alternative. Suppose we have the KDC remember session keys instead of putting these in the TGT. This design would eliminate the need for TGTs. But it would also require the KDC to maintain state, and a stateless KDC is one of the most impressive design features in Kerberos.

10.6 WEP

Wired Equivalent Privacy, or WEP, is a security protocol that was designed to make a wireless local area network (LAN) as secure as a wired LAN. By any measure, WEP is a seriously flawed protocol. As Tanenbaum so aptly puts it [298]:

> The 802.11 standard prescribes a data link-level security protocol called WEP (Wired Equivalent Privacy), which is designed to make the security of a wireless LAN as good as that of a wired LAN. Since the default for a wired LAN is no security at all, this goal is easy to achieve, and WEP achieves it as we shall see.

10.6.1 WEP Authentication

In WEP, a wireless access point shares a single symmetric key with all users. While it is not ideal to share one key among many users, it certainly does simplify things for the access point. In any case, the actual WEP authentication protocol is a simple challenge-response, as illustrated in Figure 10.22, where Bob is the access point, Alice is a user, and K is the shared symmetric key.

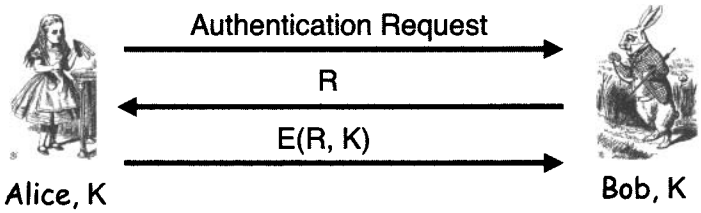

Figure 10.22: WEP Authentication

10.6.2 WEP Encryption

Once Alice has been authenticated, packets are encrypted using the RC4 stream cipher (see Section 3.2.2 for details on the RC4 algorithm), as illustrated in Figure 10.23. Each packet is encrypted with a key $K_{\text{IV}} = (\text{IV}, K)$, where IV is a 3-byte initialization vector that is sent in the clear with the packet, and K is the same key used for authentication. The goal here is to encrypt packets with distinct keys, since reuse of the key would be a bad idea (see Problem 36). Note that, for each packet, Trudy knows the 3-byte IV, but she does not know K. So the encryption key varies and it's not known by Trudy.

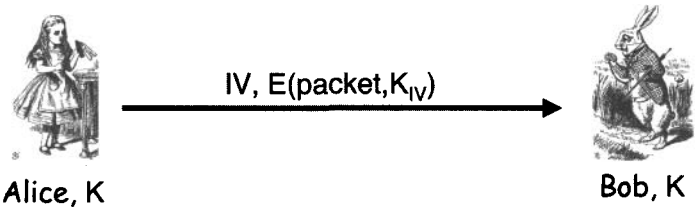

Figure 10.23: WEP Encryption

Since the IV is only three bytes long, and the key K seldom changes, the encryption key $K_{\text{IV}} = (\text{IV}, K)$ will repeat often (see Problems 37). Furthermore, whenever the key K_{IV} repeats, Trudy will know it, since the IV is visible (assuming K has not changed). RC4 is a stream cipher, so a repeated key implies reuse of the keystream, which is a serious problem. Further repeats of the same IV make Trudy's job even easier.

The number of repeated encryption keys could be reduced if K was changed regularly. Unfortunately, the long-term key K seldom changes since, in WEP, such a change is a manual process and the access point and all hosts must update their keys. That is, there is no key update procedure built into WEP.

The bottom line is that, whenever Trudy sees a repeated IV, she can safely assume the same keystream was used. Since the IV is only 24 bits, repeats will occur relatively often. And, since a stream cipher is used, a repeated keystream is at least as bad as reuse of a one-time pad.

In addition to this small-IV problem, there is another distinct cryptanalytic attack on WEP encryption. While RC4 is considered a strong cipher when used correctly, there is a practical attack that can be used to recover the RC4 key from WEP ciphertext. This clever attack, which can be considered a type of related key attack, is due to Fluhrer, Mantin, and Shamir [112]. This attack is discussed in detail in Section 6.3 of Chapter 6, or see [284] for more information.

10.6.3 WEP Non-Integrity

WEP has numerous security problems, but one of the most egregious is that it uses a cyclic redundancy check (CRC) for "integrity" protection. Recall that a cryptographic integrity check is supposed to detect malicious tampering with the data—not just transmission errors. While a CRC is a good error detection method, it is useless for cryptographic integrity, since an intelligent adversary can alter the data and, simultaneously, the CRC value so that the integrity check is passed. This is precisely the attack that a true cryptographic integrity check, such as a MAC, HMAC, or digital signature, will prevent.

This integrity problem is made worse by the fact that the data is encrypted with a stream cipher. Because a stream cipher is used, WEP encryption is linear, which allows Trudy to make changes directly to the ciphertext and change the corresponding CRC value so that the receiver will not detect the tampering. That is, Trudy does not need know the key or plaintext to make undetectable changes to the data. Under this scenario, Trudy won't know what changes she has made, but the point is that the data can be corrupted in a way that neither Alice nor Bob can detect.

The problems only get worse if Trudy should happen to know some of the plaintext. For example, suppose that Trudy knows the destination IP address of a given WEP-encrypted packet. Then without any knowledge of the key, Trudy can change the destination IP address to an IP address of her choosing (for example, her own IP address), and change the CRC integrity check so that her tampering will go undetected. Since WEP traffic is only encrypted from the host to the wireless access point (and vice versa), when the altered packet arrives at the access point, it will be decrypted and forwarded to Trudy's preferred IP address. From the perspective of a lazy cryptanalyst, it doesn't get any better than this. Again, this attack is made possible by the lack of any real integrity check. The bottom line is that the WEP "integrity check" provides no cryptographic integrity whatsoever.

10.6.4 Other WEP Issues

There are many more WEP security vulnerabilities. For example, if Trudy can send a message over the wireless link and intercept the ciphertext, then she will know the plaintext and the corresponding ciphertext, which enables her to immediately recover the keystream. This same keystream will be used to encrypt any message that uses the same IV, provided the long-term key has not changed (which, as pointed out above, it seldom does).

Would Trudy ever know the plaintext of an encrypted message sent over the wireless link? Perhaps Trudy could send an email message to Alice and ask her to forward it to another person. If Alice does so, then Trudy could intercepted the ciphertext message corresponding to the known plaintext.

Another issue is that, by default, a WEP access point broadcasts its SSID (the Service Set Identifier), which acts as its ID. The client must use the SSID when authenticating to the access point. One security feature of WEP makes it possible to configure the access point so that it does not broadcast the SSID, in which case the SSID acts something like a password that users must know to authenticate to the access point. However, users send the SSID in the clear when contacting the access point, and Trudy only needs to intercept one such packet to discover the SSID "password." Even worse, there are tools that will force WEP clients to de-authenticate, in which case the clients will then automatically attempt to re-authenticate, in the process, sending the SSID in the clear. Consequently, as long as there is at least one active user, it's a fairly simple process for Trudy to obtain the SSID.

10.6.5 WEP: The Bottom Line

It's difficult—if not impossible—to view WEP as anything but a security disaster. However, in spite of all of its multiple security problems, in some circumstances it may be possible to make WEP moderately secure in practice. Ironically, this has more to do with the inherent insecurity of WEP than with any inherent security of WEP. Suppose that you configure your WEP access points so that it encrypts the data, it does not broadcast its SSID, and you use access control (i.e., only machines with specified MAC addresses are allowed to use the access point). Then an attacker must expend some effort to gain access—at a minimum, Trudy must break the encryption, spoof her MAC address, and probably force users to de-authenticate so that she can obtain the SSID. While there are tools to help with all of these tasks, it would likely be much simpler for Trudy to find an unprotected WEP network. Like most people, Trudy generally chooses the path of least resistance. Of course, if Trudy has reason to specifically target your WEP installation (as opposed to simply wanting free network access), you will be vulnerable as long as you rely on WEP.

Finally, we note that there are more secure alternatives to WEP. For example, Wi-Fi Protected Access (WPA) is significantly stronger, but it was designed to use the same hardware as WEP, so some security compromises were necessary. A few attacks on WPA are known but, as a practical matter, it seems to be secure. There is also a WPA2 which, in principle, is somewhat stronger than WPA, but it requires more powerful hardware. As with WPA, there are some claimed attacks on WPA2, but these also appear to be of little practical significance. Today, WEP can be broken in minutes whereas the only serious threats against WPA and WPA2 are password cracking attacks. If reasonably strong passwords are chosen, WPA and WPA2 both would be considered practically secure, by any conceivable definition. In any case, both WPA and WPA2 are vast improvements over WEP [325].

10.7 GSM

To date, many wireless protocols, such as WEP, have a poor track record with respect to security [17, 38, 93]. In this section we'll discuss the security of GSM cell phones. GSM illustrates some of the unique security problems that arise in a wireless environment. It's also an excellent example of how mistakes at the design phase are extremely difficult to correct later. But before we delve into to GSM security, we need some background information on the development of cell phone technology.

Back in the computing stone age (prior to the 1980s, that is) cell phones were expensive, completely insecure, and as large as a brick. These *first-generation* cell phones were analog, not digital, and there were few standards and little or no thought was given to security.

The biggest security issue with early cell phones was their susceptibility to *cloning*. These cell phones would send their identity in the clear when a phone call was placed, and this identity was used to determine who to bill for the phone call. Since the ID was sent over a wireless media, it could easily be captured and then used to make a copy or clone, of the phone. This allowed the bad guys to make free phone calls, which did not please cellular phone companies, who ultimately had to bear the cost. Cell phone cloning became a big business, with fake base stations created simply to harvest IDs [14].

Into this chaotic environment came GSM, which began in 1982 as Groupe Spéciale Mobile, but in 1986 it was formally rechristened as Global System for Mobile Communications.[14] The founding of GSM marks the official beginning of *second-generation* cell phone technology [142]. We'll have much more to say about GSM security below.

Recently, *third-generation* cell phones have become popular. The 3rd Generation Partnership Project, or 3GPP [1], is the trade group behind 3G phones. We'll briefly mention the security architecture promoted by the 3GPP after we complete our survey of GSM security.

10.7.1 GSM Architecture

The general architecture of GSM is illustrated in Figure 10.24, where the following terminology is used.

- The *mobile* is the cell phone.
- The *air interface* is where the wireless transmission from the cell phone to a base station occurs.
- The *visited network* typically includes multiple base stations and a *base station controller*, which acts as a hub for connecting the base stations

[14]This is a tribute to the universality of three-letter acronyms.

under its control to the rest of the GSM network. The base station controller includes a *visitor location registry*, or VLR, which is used to keep tabs on all mobiles currently active in the VLR's network.

- The *public switched telephone network*, or PSTN, is the ordinary (non-cellular) telephone system. The PSTN is sometimes referred to as "land lines" to distinguish it from the wireless network.
- The *home network* is the network where the mobile is registered. Each mobile is associated with a unique home network. The home network includes a *home location registry*, or HLR, which keeps track of the most recent location of all mobiles listed in the HLR. The *authentication center*, or AuC, maintains the crucial billing information for all mobiles that belong to the corresponding HLR.

We'll discuss these pieces of the GSM puzzle in more detail below.

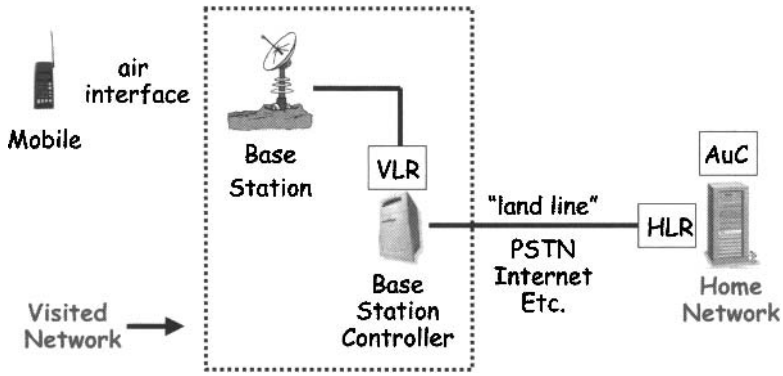

Figure 10.24: GSM Overview

Each GSM mobile phone contains a Subscriber Identity Module, or SIM, which is a tamper-resistant smartcard. The SIM contains an International Mobile Subscriber ID, or IMSI, which, not surprisingly, is used to identify the mobile. The SIM also contains a 128-bit key that is known only by the mobile and its home network. This key is universally know as Ki, so we'll follow the standard notation.

The purpose of using a smartcard for the SIM is to provide an inexpensive form of tamper-resistant hardware. The SIM card also provides two-factor authentication, relying on "something you have" (the mobile containing the SIM) and "something you know" in the form of a four-digit PIN. However, the PIN is usually treated as an annoyance, and it's often not used.

Again, the visited network is the network where the mobile is currently located. A base station is one cell in the cellular system, whereas the base

station controller manages a collection of cells. The VLR has information on all mobiles currently visiting the base station controller's territory.

The home network stores a given mobile's crucial information, namely, its IMSI and key Ki. Note that the IMSI and Ki are, in effect, the username and "password" for the mobile when it wants to access the network to make a call. The HLR keeps track of the most recent location of each of it's registered mobiles, while the AuC contains each registered mobile's IMSI and key Ki.

10.7.2 GSM Security Architecture

Now we're ready to take a close look at the GSM security architecture. The primary security goals set forth by the designers of GSM were the following.

- Make GSM as secure as ordinary telephones (the PSTN)
- Prevent cell phone cloning

Note that GSM was not designed to resist an active attack. At the time, active attacks were considered infeasible, since the necessary equipment was costly. However, today the cost of such equipment is little more than that of a good laptop computer, so neglecting active attacks was probably shortsighted. The designers of GSM considered the biggest threats to be insecure billing, corruption, and similar low-tech attacks.

GSM attempts to deal with three security issues: anonymity, authentication, and confidentiality. In GSM, the anonymity is supposed to prevent intercepted traffic from being used to identify the caller. Anonymity is not particularly important to the phone companies, except to the extent that it is important for customer confidence. Anonymity is something users might reasonably expect from non-cellular phone calls.

Authentication, on the other hand, is of paramount importance to phone companies, since correct authentication is necessary for proper billing. The first-generation cloning problems can be viewed as an authentication failure. As with anonymity, confidentiality of calls over the air interface is important to customers, and so, to that extent, it's important to phone companies.

Next, we'll look at GSM's approach to anonymity, authentication, and confidentiality in more detail. Then we'll discuss some of the many security flaws in GSM.

10.7.2.1 Anonymity

GSM provides a very limited form of anonymity. The IMSI is sent in the clear over the air interface at the start of a call. Then a random Temporary Mobile Subscriber ID, or TMSI, is assigned to the caller, and the TMSI is subsequently used to identify the caller. In addition, the TMSI changes frequently. The net effect is that, if an attacker captures the initial part

of a call, the caller's anonymity will be compromised. But if the attacker misses the initial part of the call, then the anonymity is, in a practical sense, reasonably well protected. Although this is not a strong form of anonymity, it may be sufficient for real-world situations where an attacker could have difficulty filtering the IMSIs out of a large volume of traffic. It seems that the GSM designers did not take anonymity too seriously.

10.7.2.2 Authentication

From the phone company's perspective, authentication is the most critical aspect of the GSM security architecture. Authenticating the user to the base station is necessary to ensure that the phone company will get paid for the service they provide. In GSM, the caller is authenticated to the base station, but the authentication is not mutual. That is, the GSM designers decided that it was not necessary to verify the identity of the base station. We'll see that this was a significant security oversight.

GSM authentication uses a simple challenge-response mechanism. The caller's IMSI is received by the base station, which then passes it to the caller's home network. Recall that the home network knows the caller's IMSI and key Ki. The home network generates a random challenge, RAND, and computes the "expected response," XRES = A3(RAND, Ki), where A3 is a hash function. Then the pair (RAND, XRES) is sent from the home network to the base station. The base station sends the challenge, RAND, to the mobile. The mobile's response is denoted SRES, where SRES is computed by the mobile as SRES = A3(RAND, Ki). To complete the authentication, the mobile sends SRES to the base station which verifies that SRES = XRES. Note that in this authentication protocol, the caller's key Ki never leaves its home network or the mobile. It's important that Trudy cannot obtain Ki, since that would enable her to clone the caller's phone.

10.7.2.3 Confidentiality

GSM uses a stream cipher to encrypt the data. The reason for this choice is due to the relatively high error rate in the cell phone environment, which is typically about 1 in 1000 bits. With a block cipher, each transmission error causes one or two plaintext blocks to be garbled (depending on the mode), while a stream cipher garbles only those plaintext bits corresponding to the specific ciphertext bits that are in error.[15]

The GSM encryption key is universally denoted as Kc, so we'll follow that convention. When the home network receives the IMSI from the base station controller, the home network computes Kc = A8(RAND, Ki), where A8 is

[15]It is possible to use error correcting codes to minimize the effects of transmission errors, making block ciphers feasible. However, this adds another layer of complexity to the process.

another hash function. Then Kc is sent along with the pair RAND and XRES, that is, the triple (RAND, XRES, Kc) is sent from the home network to the base station.[16]

Once the base station receives the triple (RAND, XRES, Kc), it uses the authentication protocol described above. If this succeeds, the mobile computes Kc = A8(RAND, Ki). The base station already knows Kc, so the mobile and base station have a shared symmetric key with which to encrypt the conversation. As mentioned above, the data is encrypted with the A5/1 stream cipher. As with authentication, the caller's master key Ki never leaves its home network.

10.7.3 GSM Authentication Protocol

The part of the GSM protocol that occurs between the mobile and the base station is illustrated in Figure 10.25. A few security concerns with this protocol are as follows [228].

- The RAND is hashed together with Ki to produce the encryption key Kc. Also, the value of RAND is hashed with Ki to generate SRES, which a passive attacker can see. As a result, it's necessary that SRES and Kc be uncorrelated—otherwise there would have been a shortcut attack on Kc. These hash values will be uncorrelated if a secure cryptographic hash function is used.

- It must not be possible to deduce Ki from known RAND and SRES pairs, since such pairs are available to a passive attacker. This is analogous to a known plaintext attack with a hash function in place of a cipher.

- It must not be possible to deduce Ki from chosen RAND and SRES pairs, which is analogous to a chosen plaintext attack on the hash function. Although this attack might seem implausible, with possession of the SIM card, an attacker can choose the RAND values and observe the corresponding SRES values.[17]

[16]Note that the encryption key Kc is sent from the home network to the base station. Trudy may be able to obtain the encryption key by simply observing traffic sent over the network. In contrast, the authentication key Ki never leaves the home network or the mobile, so it is not subject to such an attack. This shows the relative importance the GSM architects placed on authentication as compared to confidentiality.

[17]If this attack is feasible, it is a threat even if it's slow, since the person who sells the phone would likely possess it for an extended period of time. On the other hand, if the attack is fast, then a phone that is "lost" for a few minutes would be subject to cloning.

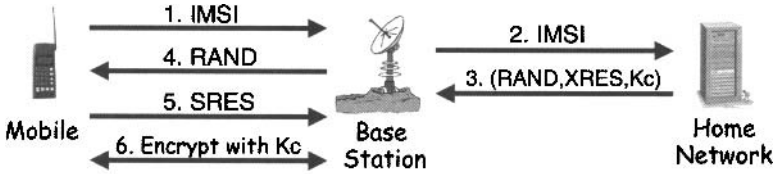

Figure 10.25: GSM Authentication and Encryption

10.7.4 GSM Security Flaws

Next, we'll discuss security flaws in GSM—there are cryptographic flaws and there are protocol flaws as well. But, arguably, the most serious problems arise from invalid security assumptions made by the designers of GSM.

10.7.4.1 Crypto Flaws

There are several cryptographic flaws in GSM. The hashes A3 and A8 both are based on a hash function known as COMP128. The hash COMP128 was developed as a secret design, in violation of Kerckhoffs' Principle. Not surprisingly, COMP128 was later found to be weak—it can be broken by 150,000 chosen "plaintexts" [130]. What this means in practice is that an attacker who has access to a SIM card can determine the key Ki in 2 to 10 hours, depending in the speed of the card. In particular, an unscrupulous seller could determine Ki before selling a phone, then create clones that would have their calls billed to the purchaser of the phone. Below, we'll mention another attack on COMP128.

There are two different forms of the encryption algorithm A5, which are known as A5/1 and A5/2. Recall that we discussed A5/1 in Chapter 3. As with COMP128, both of these ciphers were developed in violation of Kerckhoffs' Principle and both are weak. The A5/2 algorithm is the weaker of the two [26, 234] but feasible attacks on A5/1 are known [33].

10.7.4.2 Invalid Assumptions

There is a serious design flaw in the GSM protocol. A GSM phone call is encrypted between the mobile and the base station but not from the base station to the base station controller. Recall that a design goal of GSM was to develop a system as secure as the public switched telephone network (PSTN). As a result, if a GSM phone call is at some point routed over the PSTN, then from that point on, no further special protection is required. Consequently, the emphasis of GSM security is on protecting the phone call over the air interface, between the mobile and the base station.

The designers of GSM assumed that once the call reached the base station, it would be routed over the PSTN to the base station controller. This is implied by the solid line between the base station and base station controller in Figure 10.24. Due to this assumption, the GSM security protocol does not protect the conversation when it is sent from the base station to the base station controller. However, many GSM systems actually transmit calls between a base station and its base station controller over a microwave link [228]. Since microwave is a wireless media, it is possible (but not easy) for an attacker to eavesdrop on unprotected calls over this link, rendering the encryption over the air interface useless.

10.7.4.3 SIM Attacks

Several attacks have been developed on various generations of SIM cards. In one *optical fault induction* attack, an attacker could force a SIM card to divulge its Ki by using an ordinary flashbulb [269]. In another class of attacks, known as *partitioning attacks*, timing and power consumption analysis could be used to recover Ki using as few as eight adaptively chosen plaintexts [243]. As a result, an attacker who has possession of the SIM could recover Ki in seconds and, consequently, a misplaced cell phone could be cloned in seconds.

10.7.4.4 Fake Base Station

Another serious flaw with the GSM protocol is the threat posed by a fake base station. This attack, which is illustrated in Figure 10.26, exploits two flaws in the protocol. First, the authentication is not mutual. While the caller is authenticated to the base station (which is necessary for proper billing), the designers of GSM felt it was not worth the extra effort to authenticate the base station to the caller. Although they were aware of the possibility of a fake base station, apparently the protocol designers believed that the probability of such an attack was too remote to justify the (small) additional cost of mutual authentication. The second flaw that this attack exploits is that encryption over the air interface is not automatic. In fact, the base station determines whether the call is encrypted or not, and the caller does not know which is the case.

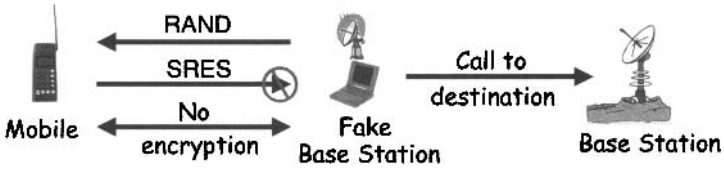

Figure 10.26: GSM Fake Base Station

In the attack illustrated in Figure 10.26, the fake base station sends a random value to the mobile, which the mobile assumes is RAND. The mobile replies with the corresponding SRES, which the fake base station discards, since it does not intend to authenticate the caller (in fact, it cannot authenticate the caller). The fake base station then tells the mobile not to encrypt the call. Unbeknownst to either the caller or the recipient, the fake base station then places a call to the the intended recipient and forwards the conversation from the caller to the recipient and vice versa. The fake base station can then eavesdrop on the entire conversation.

Note that in this fake base station attack, the fake base station would be billed for the call, not the caller. The attack might be detected if the caller complained about not being billed for the phone call. But, would anyone complain about not receiving a bill?

Also, the fake base station is in position to send any RAND it chooses and receives the corresponding SRES. Therefore, it can conduct a chosen plaintext attack on the SIM without possessing the SIM card. The SIM attack mentioned above that requires eight adaptively chosen plaintexts would be feasible with a fake base station.

Another major flaw with the GSM protocol is that it provides no replay protection. A compromised triple $(\text{RAND}, \text{XRES}, Kc)$ can be replayed forever. As a result, one compromised triple gives an attacker a key Kc that is valid indefinitely. A clever fake base station operator could even use a compromised triple to "protect" the conversation between the mobile and the fake base station so that nobody else could eavesdrop on the conversation.

Finally, it's worth noting that denial of service is always an issue in a wireless environment, since the signal can be jammed. But jamming is an issue beyond the scope of a security protocol.

10.7.5 GSM Conclusions

From our discussion of GSM security flaws, it might seem that GSM is a colossal security failure. However, GSM was certainly a commercial success, which raises some questions about the financial significance of good security. In any case, it is interesting to consider whether GSM achieved its security design goals. Recall that the two goals set forth by the designers of GSM were to eliminate the cloning that had plagued first-generation systems and to make the air interface as secure as the PSTN. Although it is possible to clone GSM phones, it never became a significant problem in practice. So it would seem that GSM did achieve its first security goal.

Did GSM make the air interface as secure as the PSTN? There are attacks on the GSM air interface (e.g., fake base station), but there are also attacks on the PSTN (tapping a line) that are at least as severe. So it could be argued that GSM achieved its second design goal, although this is debatable.

The real problem with GSM security is that the initial design goals were too limited. The major insecurities in GSM include weak crypto, SIM issues, the fake base station attack, and a total lack of replay protection. In the PSTN, the primary insecurity is tapping, though there are others threats, such as attacks on cordless phones. Overall, GSM could reasonably be considered a modest security success.

10.7.6 3GPP

The security design for third-generation cell phones was spearheaded by the 3GPP. This group clearly set their sights higher than the designers of GSM. Perhaps surprisingly, the 3GPP security model is built on the foundation of GSM. However, the 3GPP developers carefully patched all of the known GSM vulnerabilities. For example, 3GPP includes mutual authentication and integrity protection of all signaling, including the "start encryption" command for the base station to mobile communication. These improvements eliminate the GSM-style fake base station attack. Also, in 3GPP, the keys can't be reused and triples can't be replayed. The weak proprietary crypto algorithms of GSM (COMP128, A5/1, and A5/2) have been replaced by the strong encryption algorithm, KASUMI, which has undergone rigorous peer review. In addition, the encryption has been extended from the mobile all the way to the base station controller.

The history of mobile phones, from the first-generation through GSM and now 3GPP, nicely illustrates the evolution that often occurs in security. As the attackers develop new attacks, the defenders respond with new protections, which the attackers again probe for weaknesses. Ideally, this arms race approach to security could be avoided by a careful design and analysis prior to the initial development. However, it's unrealistic to believe that the designers of first-generation cell phones could have imagined the mobile world of today. Attacks such as the fake base station, which would have seemed improbable at one time, are now easily implemented. With this in mind, we should realize that, although 3GPP clearly promises more security than GSM could deliver, it's possible that attacks will eventually surface. In short, the security arms race continues.

10.8 Summary

In this chapter, we discussed several real-world security protocols in detail. We first considered SSH, which is a fairly straightforward protocol. Then we looked at SSL which is a well-designed protocol that is widely used on the Internet.

We saw that IPSec is a complex protocol with some serious security issues. The designers of IPSec over-engineered the protocol, which is the source of

its complex. IPSec provides a good illustration of the maxim that complexity is the enemy of security.

Kerberos is a widely deployed authentication protocol that relies on symmetric key cryptography and timestamps. The ability of the Kerberos KDC to remain stateless is one of the many clever features of the protocol.

We finished the chapter with a discussion of two wireless protocols, WEP and GSM. WEP is a seriously flawed protocol—one of its many problems is the lack of any meaningful integrity check. You'd be hard pressed to find a better example to illustrate the pitfalls that arise when integrity is not protected.

GSM is another protocol with some major problems. The actual GSM security protocol is simple, but it has a large number of flaws. While complexity might be the enemy of security, GSM illustrates that simplicity isn't necessarily security's best friend. Arguably, the most serious problem with GSM is that its designers were not ambitious enough, since they didn't design GSM to withstand attacks that are easy today. This is perhaps excusable given that some of these attacks seemed far fetched in 1982 when GSM was developed. GSM also shows that it's difficult to overcome security flaws after the fact.

The security of third-generation cell phones is built on the GSM model, with all of the known security flaws in GSM having been patched. It will be interesting to see how 3GPP security holds up in practice.

10.9 Problems

1. Consider the SSH protocol in Figure 10.1.

   a. Explain precisely how and where Alice is authenticated. What prevents a replay attack?

   b. If Trudy is a passive attacker (i.e., she can only observe messages), she cannot determine the key K. Why?

   c. Show that if Trudy is an active attacker (i.e., she can actively send messages) and she can impersonate Bob, then she can determine the key K that Alice uses in the last message. Explain why this does not break the protocol.

   d. What is the purpose of the encrypting the final message with the key K?

2. Consider the SSH protocol in Figure 10.1. One variant of the protocol allows us to replace Alice's certificate, certificate_A, with Alice's password, password_A. Then we must also remove S_A from the final message. This modification yields a version of SSH where Alice is authenticated based on a password.

a. What does Bob need to know so that he can authenticate Alice?

b. Based on Problem 1, part b, we see that Trudy, as an active attacker, can establish a shared symmetric key K with Alice. Assuming this is the case, can Trudy then use K to determine Alice's password?

c. What are the significant advantages and disadvantages of this version of SSH, as compared to the version in Figure 10.1, which is based on certificates?

3. Consider the SSH protocol in Figure 10.1. One variant of the protocol allows us to replace certificate$_A$ with Alice's public key. In this version of the protocol, Alice must have a public/private key pair, but she is not required to have a certificate. It is also possible to replace certificate$_B$ with Bob's public key.

a. Suppose that Bob has a certificate, but Alice does not. What must Bob do so that he can authenticate Alice?

b. Suppose that Alice has a certificate, but Bob does not. What must Alice do so that she can authenticate Bob?

c. What are the significant advantages and disadvantages of this public key version of SSH, as compared to the certificate version in Figure 10.1?

4. Use Wireshark [328] to capture SSH authentication packets.

a. Identify the packets that correspond to the messages shown in Figure 10.1.

b. What other SSH packets do you observe, and what do these packets contain?

5. Consider the SSH specification, which can be found in RFC 4252 [331] and RFC 4253 [333].

a. Which message or messages in Figure 10.1 correspond to the message or messages labeled as SSH_MSG_KEXINIT in the protocol specification?

b. Which message or messages in Figure 10.1 correspond to the message or messages labeled as SSH_MSG_NEWKEYS in the protocol specification?

c. Which message or messages in Figure 10.1 correspond to the message or messages labeled as SSH_MSG_USERAUTH in the protocol specification?

d. In the actual SSH protocol, there are two additional messages that would come between the fourth and fifth messages in Figure 10.1. What are these messages and what purpose do they serve?

6. Consider the SSL protocol in Figure 10.4.

   a. Suppose that the nonces R_A and R_B are removed from the protocol and we define $K = h(S)$. What effect, if any, does this have on the security of the authentication protocol?

   b. Suppose that we change message four to

   $$\text{HMAC}(\text{msgs}, \text{SRVR}, K).$$

   What effect, if any, does this have on the security of the authentication protocol?

   c. Suppose that we change message three to

   $$\{S\}_{\text{Bob}},\ h(\text{msgs}, \text{CLNT}, K).$$

   What effect, if any, does this have on the security of the authentication protocol?

7. Consider the SSL protocol in Figure 10.4. Modify the protocol so that the authentication is based on a digital signature. Your protocol must provide secure authentication of the server Bob, and a secure session key.

8. Consider the SSL protocol in Figure 10.4. This protocol does not allow Bob to remain anonymous, since his certificate identifies him.

   a. Modify the SSL session protocol so that Bob can remain anonymous with respect to a passive attacker.

   b. Can you solve part a without increasing the number of messages?

9. The SSL protocol discussed in Section 10.3 uses public key cryptography.

   a. Design a variant of SSL that is based on symmetric key cryptography.

   b. What is the primary disadvantage of using symmetric keys for an SSL-like protocol?

10. Use Wireshark [328] to capture SSL authentication packets.

    a. Identify the packets that correspond to the messages shown in Figure 10.4.

b. What do the other SSL packets contain?

11. SSL and IPSec are both designed to provide security over the network.

    a. What are the primary advantages of SSL over IPSec?

    b. What are the primary advantages of IPSec over SSL?

12. SSL and IPSec are both designed to provide security over the network.

    a. What are the significant similarities between the two protocols?

    b. What are the significant differences between the two protocols?

13. Consider a man-in-the-middle attack on an SSL session between Alice and Bob.

    a. At what point should this attack fail?

    b. What mistake might Alice reasonably make that would allow this attack to succeed?

14. In Kerberos, Alice's key K_A, which is shared by Alice and the KDC, is computed (on Alice's computer) as $K_A = h(\text{Alice's password})$. Alternatively, this could have been implemented as follows. Initially, the key K_A is randomly generated on Alice's computer. The key is stored on Alice's computer as $E(K_A, K)$ where the key K is computed as $K = h(\text{Alice's password})$. The key K_A is also stored on the KDC.

    a. What are the advantages to this alternate approach of generating and storing K_A?

    b. Are there any disadvantages to computing and storing $E(K_A, K)$?

15. Consider the Kerberos interaction discussed in Section 10.5.2.

    a. Why is the ticket to Bob encrypted with K_B?

    b. Why is "Alice" included in the (encrypted) ticket to Bob?

    c. In the REPLY message, why is the ticket to Bob encrypted with the key S_A?

    d. Why is the ticket to Bob sent to Alice (who must then forward it to Bob) instead of being sent directly to Bob?

16. Consider the Kerberized login discussed in this chapter.

    a. What is a TGT and what is its purpose?

    b. Why is the TGT sent to Alice instead of being stored on the KDC?

    c. Why is the TGT encrypted with K_{KDC}?

d. Why is the TGT encrypted with K_A when it is sent from the KDC to Alice's computer?

17. This problem deals with Kerberos.

    a. Why can Alice remain anonymous when requesting a ticket to Bob?

    b. Why can Alice not remain anonymous when requesting a TGT from the KDC?

    c. Why can Alice remain anonymous when she sends the "ticket to Bob" to Bob?

18. Suppose we use symmetric keys for authentication and each of N users must be able to authenticate any of the other $N-1$ users. Evidently, such a system requires one symmetric key for each pair of users, or on the order of N^2 keys. On the other hand, if we use public keys, only N key pairs are required, but we must then deal with PKI issues.

    a. Kerberos authentication uses symmetric keys, yet only N keys are required for N users. How is this accomplished?

    b. In Kerberos, no PKI is required. But, in security, there is no free lunch, so what's the tradeoff?

19. Dog race tracks often employ Automatic Betting Machines (ABMs),[18] which are somewhat analogous to ATM machines. An ABM is a terminal where Alice can place her own bets and scan her winning tickets. An ABM does not accept or dispense cash. Instead, an ABM only accepts and dispenses *vouchers*. A voucher can also be purchased from a special voucher machine for cash, but a voucher can only be redeemed for cash by a human teller.

    A voucher includes 15 hexadecimal digits, which can be read by a human or scanned by a machine—the machine reads a bar code on the voucher. When a voucher is redeemed, the information is recorded in a voucher database and a paper receipt is printed. For security reasons, the (human) teller must submit the paper receipt which serves as the physical record that the voucher was cashed.

    A voucher is valid for one year from its date of issue. However, the older that a voucher is, the more likely that it has been lost and will never be redeemed. Since vouchers are printed on cheap paper, they are often damaged to the point where they fail to scan, and they can even be difficult for human tellers to process manually.

[18] Not to be confused with anti-ballistic missiles.

A list of all outstanding vouchers is kept in a database. Any human teller can read the first 10 hex digits from this database for any outstanding voucher. But, for security reasons, the last five hex digits are not available to tellers.

If Ted, a teller, is asked to cash a valid voucher that doesn't scan, he must manually enter its hex digits. Using the database, it's generally easy for Ted to match the first 10 hex digits. However, the last five hex digits must be determined from the voucher itself. Determining these last five hex digits can be difficult, particularly if the voucher is in poor condition.

To help overworked tellers, Carl, a clever programmer, added a wildcard feature to the manual voucher entry program. Using this feature, Ted (or any other teller) can enter any of the last five hex digits that are readable and "*" for any unreadable digits. Carl's program will then inform Ted whether an outstanding voucher exists that matches in the digits that were entered, ignoring any position with a "*." Note that this program does not give Ted the missing digits, but instead, it simply returns a yes or no answer.

Suppose that Ted is given a voucher for which none of the last five hex digits can be read.

a. Without the wildcard feature, how many guesses must Ted make, on average, to recover the last five hex digits of this particular voucher?

b. Using the wildcard feature, how many guesses, on average, must Ted make to recover the last 5 hex digits of this voucher?

c. How could Dave, a dishonest teller, exploit the wildcard feature to cheat the system?

d. What is the risk for Dave? That is, how might Dave get caught under the current system?

e. Modify the current system so that it allows tellers to securely and efficiently deal with vouchers that fail to scan automatically, but also makes it impossible (or at least more difficult) for Dave to cheat the system.

20. IPSec is a much more complex protocol than SSL, which is often attributed to the fact that IPSec is over-engineered. Suppose that IPSec was not over-engineered. Then would IPSec still be more complex than SSL? In other words, is IPSec inherently more complex than SSL, or not?

21. IKE has two phases, Phase 1 and Phase 2. In IKE Phase 1, there are four key options and, for each of these, there is a main mode and an aggressive mode.

    a. What are the primary differences between main mode and aggressive mode?

    b. What is the primary advantage of the Phase 1 digital signature key option over Phase 1 public key encryption?

22. IKE has two phases, Phase 1 and Phase 2. In IKE Phase 1, there are four key options and, for each of these, there is a main mode and an aggressive mode.

    a. Explain the difference between Phase 1 and Phase 2.

    b. What is the primary advantage of Phase 1 public key encryption main mode over Phase 1 symmetric key encryption main mode?

23. IPSec cookies are also known as anti-clogging tokens.

    a. What is the intended security purpose of IPSec cookies?

    b. Why do IPSec cookies fail to fulfill their intended purpose?

    c. Redesign the IPSec Phase 1 symmetric key signing main mode so that the IPSec cookies do serve their intended purpose.

24. In IKE Phase 1 digital signature main mode, proof_A and proof_B are signed by Alice and Bob, respectively. However, in IKE Phase 1, public key encryption main mode, proof_A and proof_B are neither signed nor encrypted with public keys. Why is it necessary to sign these values in digital signature mode, yet it is not necessary to public key encrypt (or sign) them in public key encryption mode?

25. As noted in the text, IKE Phase 1 public key encryption aggressive mode[19] allows Alice and Bob to remain anonymous. Since anonymity is usually given as the primary advantage of main mode over aggressive mode, is there any reason to ever use public key encryption main mode?

26. IKE Phase 1 uses ephemeral Diffie–Hellman for perfect forward secrecy (PFS). Recall that in our example of PFS in Section 9.3.4 of Chapter 9, we encrypted the Diffie–Hellman values with a symmetric key to prevent the man-in-the-middle attack. However, the Diffie–Hellman values are not encrypted in IKE. Is this a security flaw? Explain.

[19]Don't try saying "IKE Phase 1 public key encryption aggressive mode" all at once or you might give yourself a hernia.

27. We say that Trudy is a *passive attacker* if she can only observe the messages sent between Alice and Bob. If Trudy is also able to insert, delete, or modify messages, we say that Trudy is an *active attacker.* If, in addition to being an active attacker, Trudy is able to establish a legitimate connection with Alice or Bob, then we say that Trudy is an *insider.* Consider IKE Phase 1 digital signature main mode.

    a. As a passive attacker, can Trudy determine Alice's identity?

    b. As a passive attacker, can Trudy determine Bob's identity?

    c. As an active attacker, can Trudy determine Alice's identity?

    d. As an active attacker, can Trudy determine Bob's identity?

    e. As an insider, can Trudy determine Alice's identity?

    f. As an insider, can Trudy determine Bob's identity?

28. Repeat Problem 27 for symmetric key encryption, main mode.

29. Repeat Problem 27 for public key encryption, main mode.

30. Repeat Problem 27 for public key encryption, aggressive mode.

31. Recall that IPSec transport mode was designed for host-to-host communication, while tunnel mode was designed for firewall-to-firewall communication.

    a. Why does IPSec tunnel mode fail to hide the header information when used from host to host?

    b. Does IPSec tunnel mode also fail to hide the header information when used from firewall to firewall? Why or why not?

32. Recall that IPSec transport mode was designed for host-to-host communication, while tunnel mode was designed for firewall-to-firewall communication.

    a. Can transport mode be used for firewall-to-firewall communication? Why or why not?

    b. Can tunnel mode be used for host-to-host communication? Why or why not?

33. ESP requires both encryption and integrity, yet it is possible to use ESP for integrity only. Explain this apparent contradiction.

34. What are the significant differences, if any, between AH and ESP with NULL encryption?

35. Suppose that IPSec is used from host to host as illustrated in Figure 10.16, but Alice and Bob are both behind firewalls. What problems, if any, does IPSec create for the firewalls under the following assumptions.

    a. ESP with non-NULL encryption is used.

    b. ESP with NULL encryption is used.

    c. AH is used.

36. Suppose that we modify WEP so that it encrypts each packet using RC4 with the key K, where K is the same key that is used for authentication.

    a. Is this a good idea? Why or why not?

    b. Would this approach be better or worse than $K_{\text{IV}} = (\text{IV}, K)$, as is actually done in WEP?

37. WEP is supposed to protect data sent over a wireless link. As discussed in the text, WEP has many security flaws, one of which involves its use of initialization vectors, or IVs. WEP IVs are 24 bits long. WEP uses a fixed long-term key K. For each packet, WEP sends an IV in the clear along with the encrypted packet, where the packet is encrypted with a stream cipher using the key $K_{\text{IV}} = (\text{IV}, K)$, that is, the IV is pre-pended to the long-term key K. Suppose that a particular WEP connection sends packets containing 1500 bytes over an 11 Mbps link.

    a. If the IVs are chosen at random, what is the expected amount of time until the first IV repeats? What is the expected amount of time until some IV repeats?

    b. If the IVs are not selected at random but are instead selected in sequence, say, $\text{IV}_i = i$, for $i = 0, 1, 2, \ldots, 2^{24} - 1$, what is the expected amount of time until the first IV repeats? What is the expected amount of time until some IV is repeated?

    c. Why is a repeated IV a security concern?

    d. Why is WEP "unsafe at any key length" [321]? That is, why is WEP no more secure if K is 256 bits than if K is 40 bits? Hint: See [112] for more information.

38. On page 379 it is claimed that if Trudy knows the destination IP address of a WEP-encrypted packet, she can change the IP address to any address of her choosing, and the access point will send the packet to Trudy's selected IP address.

    a. Suppose that C is the encrypted IP address, P is the plaintext IP address (which is known to Trudy), and X is the IP address where

Trudy wants the packet sent. In terms of C, P, and X, what will Trudy insert in place of C?

b. What else must Trudy do for this attack to succeed?

39. WEP also incorporates a couple of security features that were only briefly mentioned in the text. In this problem, we consider two of these features.

    a. By default, a WEP access point broadcasts its SSID, which acts as the name (or ID) of the access point. A client must send the SSID to the access point (in the clear) before it can send data to the access point. It is possible to set WEP so that it does not broadcast the SSID, in which case the SSID is supposed to act like a password. Is this a useful security feature? Why or why not?

    b. It is possible to configure the access point so that it will only accept connections from devices with specified MAC addresses. Is this a useful security feature? Why or why not?

40. After the terrorist attacks of September 11, 2001, it was widely reported that the Russian government ordered all GSM base stations in Russia to transmit all phone calls unencrypted.

    a. Why would the Russian government have given such an order?

    b. Are these news reports consistent with the technical description of the GSM security protocol given in this chapter?

41. Modify the GSM security protocol, which appears in Figure 10.25, so that it provides mutual authentication.

42. In GSM, each home network has an AuC database containing user keys Ki. Instead, a process known as *key diversification* could be used. Key diversification works as follows. Let h be a secure cryptographic hash function and let K_M be a master key known only to the AuCs. In GSM, each user has a unique ID known as an IMSI. In this key diversification scheme, a user's key Ki would be given by $Ki = h(K_M, \text{IMSI})$, and this key would be stored on the mobile. Then given any IMSI, the AuC would compute the key as $Ki = h(K_M, \text{IMSI})$.

    a. What is the primary advantage of key diversification?

    b. What is the primary disadvantage of key diversification?

    c. Why do you think the designers of GSM chose not to employ key diversification?

43. Give a secure one-message protocol that prevents cell phone cloning and establishes a shared encryption key. Mimic the GSM protocol.

44. Give a secure two-message protocol that prevents cell phone cloning, prevents a fake base station attack, and establishes a shared session key. Mimic the GSM protocol.