Part I

Introduction

In This Part

CHAPTER 1

The Context of Cryptography

Cryptography is the art and science of encryption. At least, that is how it started out. Nowadays it is much broader, covering authentication, digital signatures, and many more elementary security functions. It is still both an art and a science: to build good cryptographic systems requires a scientific background and a healthy dose of the black magic that is a combination of experience and the right mentality for thinking about security problems. This book is designed to help you cultivate these critical ingredients.

Cryptography is an extremely varied field. At a cryptography research conference, you can encounter a wide range of topics, including computer security, higher algebra, economics, quantum physics, civil and criminal law, statistics, chip designs, extreme software optimization, politics, user interface design, and everything in between. In some ways, this book concentrates on only a very small part of cryptography: the practical side. We aim to teach you how to implement cryptography in real-world systems. In other ways, this book is much broader, helping you gain experience in security engineering and nurturing your ability to think about cryptography and security issues like a security professional. These broader lessons will help you successfully tackle security challenges, whether directly related to cryptography or not.

The variety in this field is what makes cryptography such a fascinating area to work in. It is really a mixture of widely different fields. There is always something new to learn, and new ideas come from all directions. It is also one of the reasons why cryptography is so difficult. It is impossible to understand it all. There is nobody in the world who knows everything about cryptography. There isn't even anybody who knows most of it. We certainly don't know

everything there is to know about the subject of this book. So here is your first lesson in cryptography: keep a critical mind. Don't blindly trust anything, even if it is in print. You'll soon see that having this critical mind is an essential ingredient of what we call ''professional paranoia.''

1.1 The Role of Cryptography

Cryptography by itself is fairly useless. It has to be part of a much larger system. We like to compare cryptography to locks in the physical world. A lock by itself is a singularly useless thing. It needs to be part of a much larger system. This larger system can be a door on a building, a chain, a safe, or something else. This larger system even extends to the people who are supposed to use the lock: they need to remember to actually lock it and to not leave the key around for anyone to find. The same goes for cryptography: it is just a small part of a much larger security system.

Even though cryptography is only a small part of the security system, it is a very critical part. Cryptography is the part that has to provide access to some people but not to others. This is very tricky. Most parts of the security system are like walls and fences in that they are designed to keep everybody out. Cryptography takes on the role of the lock: it has to distinguish between ''good'' access and ''bad'' access. This is much more difficult than just keeping everybody out. Therefore, the cryptography and its surrounding elements form a natural point of attack for any security system.

This does not imply that cryptography is always the weak point of a system. In some cases, even bad cryptography can be much better than the rest of the security system. You have probably seen the door to a bank vault, at least in the movies. You know, 10-inch-thick, hardened steel, with huge bolts to lock it in place. It certainly looks impressive. We often find the digital equivalent of such a vault door installed in a tent. The people standing around it are arguing over how thick the door should be, rather than spending their time looking at the tent. It is all too easy to spend hours arguing over the exact key length of cryptographic systems, but fail to notice or fix buffer overflow vulnerabilities in a Web application. The result is predictable: the attackers find a buffer overflow and never bother attacking the cryptography. Cryptography is only truly useful if the rest of the system is also sufficiently secure against the attackers.

There are, however, reasons why cryptography is important to get right, even in systems that have other weaknesses. Different weaknesses are useful to different attackers in different ways. For example, an attacker who breaks the cryptography has a low chance of being detected. There will be no traces of the attack, since the attacker's access will look just like a ''good'' access. This

is comparable to a real-life break-in. If the burglar uses a crowbar to break in, you will at least see that a break-in has occurred. If the burglar picks the lock, you might never find out that a burglary occurred. Many modes of attack leave traces, or disturb the system in some way. An attack on the cryptography can be fleeting and invisible, allowing the attacker to come back again and again.

1.2 The Weakest Link Property

Print the following sentence in a very large font and paste it along the top of your monitor.

A security system is only as strong as its weakest link.

Look at it every day, and try to understand the implications. The weakest link property is one of the main reasons why security systems are so fiendishly hard to get right.

Every security system consists of a large number of parts. We must assume that our opponent is smart and that he is going to attack the system at the weakest part. It doesn't matter how strong the other parts are. Just as in a chain, the weakest link will break first. It doesn't matter how strong the other links in the chain are.

Niels used to work in an office building where all the office doors were locked every night. Sounds very safe, right? The only problem was that the building had a false ceiling. You could lift up the ceiling panels and climb over any door or wall. If you took out the ceiling panels, the whole floor looked like a set of tall cubicles with doors on them. And these doors had locks. Sure, locking the doors made it slightly harder for the burglar, but it also made it harder for the security guard to check the offices during his nightly rounds. It isn't clear at all whether the overall security was improved or made worse by locking the doors. In this example, the weakest link property prevented the locking of the doors from being very effective. It might have improved the strength of a particular link (the door), but there was another link (the ceiling) that was still weak. The overall effect of locking the doors was at best very small, and its negative side effects could well have exceeded its positive contribution.

To improve the security of a system, we must improve the weakest link. But to do that, we need to know what the links are and which ones are weak. This is best done using a hierarchical tree structure. Each part of a system has multiple links, and each link in turn has sublinks. We can organize the links into what we call an *attack tree* [113]. We give an example in Figure 1.1. Let's say that we want to break into a bank vault. The first-level links are the walls, the floor, the door, and the ceiling. Breaking through any one of them gets

us into the vault. Let's look at the door in more detail. The door system has its own links: the connection between the door frame and the walls, the lock, the door itself, the bolts that keep the door in the door frame, and the hinges. We could continue by discussing individual lines of attack on the lock, one of which is to acquire a key, which in turn leads to a whole tree about stealing the key in some way.

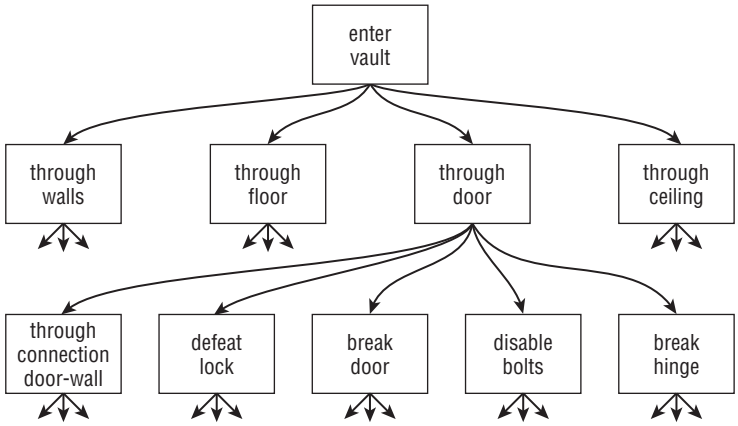

Figure 1.1: Example attack tree for a vault

We can analyze each link and split it up into other links until we are left with single components. Doing this for a real system can be an enormous amount of work. If we were concerned about an attacker stealing the diamonds stored in the vault, then Figure 1.1 is also just one piece of a larger attack tree; an attacker could trick an employee into removing the diamonds from the vault and steal them once removed. Attack trees provide valuable insight as to possible lines of attack. Trying to secure a system without first doing such an analysis very often leads to useless work. In this book, we work only on limited components—the ones that can be solved with cryptography—and we will not explicitly talk about their attack trees. But you should be certain to understand how to use an attack tree to study a larger system and to assess the role of cryptography in that system.

The weakest link property affects our work in many ways. For example, it is tempting to assume that users have proper passwords, but in practice they don't. They often choose simple short passwords. Users may go to almost any length not to be bothered by security systems. Writing a password on a sticky note and attaching it to their monitor is just one of many things they might do. You can never ignore issues like this because they always affect the end result. If you design a system that gives users a new 12-digit random password every week, you can be sure they will stick it on their monitors. This weakens an already weak link, and is bad for the overall security of the system.

Strictly speaking, strengthening anything but the weakest link is useless. In practice, things are not so clear-cut. The attacker may not know what the weakest link is and attack a slightly stronger one. The weakest link may be different for different types of attackers. The strength of any link depends on the attacker's skill and tools and access to the system. The link an attacker might exploit may also depend on the attacker's goals. So which link is the weakest depends on the situation. It is therefore worthwhile to strengthen any link that could in a particular situation be the weakest. Moreover, it's worth strengthening multiple links so that if one link does fail, the remaining links can still provide security—a property known as *defense in depth*.

1.3 The Adversarial Setting

One of the biggest differences between security systems and almost any other type of engineering is the adversarial setting. Most engineers have to contend with problems like storms, heat, and wear and tear. All of these factors affect designs, but their effect is fairly predictable to an experienced engineer. Not so in security systems. Our opponents are intelligent, clever, malicious, and devious; they'll do things nobody had ever thought of before. They don't play by the rules, and they are completely unpredictable. That is a much harder environment to work in.

Many of us remember the film in which the Tacoma Narrows suspension bridge wobbles and twists in a steady wind until it breaks and falls into the water. It is a famous piece of film, and the collapse taught bridge engineers a valuable lesson. Slender suspension bridges can have a resonance mode in which a steady wind can cause the whole structure to oscillate, and finally break. How do they prevent the same thing from happening with newer bridges? Making the bridge significantly stronger to resist the oscillations would be too expensive. The most common technique used is to change the aerodynamics of the bridge. The deck is made thicker, which makes it much harder for the wind to push up and down on the deck. Sometimes railings are used as spoilers to make the bridge deck behave less like a wing that lifts up in the wind. This works because wind is fairly predictable, and does not change its behavior in an active attempt to destroy the bridge.

A security engineer has to take a malicious wind into account. What if the wind blows up and down instead of just from the side, and what if it changes directions at the right frequency for the bridge to resonate? Bridge engineers will dismiss this kind of talk out of hand: ''Don't be silly, the wind doesn't blow that way.'' That certainly makes the bridge engineers' jobs much easier. Cryptographers don't have that luxury. Security systems are attacked by clever and malicious attackers. We have to consider all types of attack.

The adversarial setting is a very harsh environment to work in. There are no rules in this game, and the deck is stacked against us. We talk about an ''attacker'' in an abstract sense, but we don't know who she is, what she knows, what her goal is, when she will attack, or what her resources are. Since the attack may occur long after we design the system, she has the advantage of five or ten years' more research, and can use technology of the future that is not available to us. And with all those advantages, she only has to find a single weak spot in our system, whereas we have to protect all areas. Still, our mission is to build a system that can withstand it all. This creates a fundamental imbalance between the attacker of a system and the defender. This is also what makes the world of cryptography so exciting.

1.4 Professional Paranoia

To work in this field, you have to become devious yourself. You have to think like a malicious attacker to find weaknesses in your own work. This affects the rest of your life as well. Everybody who works on practical cryptographic systems has experienced this. Once you start thinking about how to attack systems, you apply that to everything around you. You suddenly see how you could cheat the people around you, and how they could cheat you. Cryptographers are professional paranoids. It is important to separate your professional paranoia from your real-world life so as to not go completely crazy. Most of us manage to preserve some sanity . . . we think.[1] In fact, we think that this practical paranoia can be a lot of fun. Developing this mindset will help you observe things about systems and your environment that most other people don't notice.

Paranoia is very useful in this work. Suppose you work on an electronic payment system. There are several parties involved in this system: the customer, the merchant, the customer's bank, and the merchant's bank. It can be very difficult to figure out what the threats are, so we use the paranoia model. For each participant, we assume that everybody else is part of a big conspiracy to defraud this one participant. And we also assume that the attacker might have any number of other goals, such as compromising the privacy of a participant's transactions or denying a participant's access to the system at a critical time. If your cryptographic system can survive the paranoia model, it has at least a fighting chance of surviving in the real world.

We will interchangeably refer to professional paranoia and the paranoia model as the security mindset.

[1] But remember: the fact that *you* are not paranoid doesn't mean *they* are not out to get you or compromise your system.

1.4.1 Broader Benefits

Once you develop a sense of professional paranoia, you will never look at systems the same way. This mindset will benefit you throughout your career, regardless of whether you become a cryptographer or not. Even if you don't become a cryptographer, you may someday find yourself working on the design, implementation, or evaluation of new computer software or hardware systems. If you have the security mindset, then you will be constantly thinking about what an attacker might try to do to your system. This will nicely position you to identify potential security problems with these systems early. You may not always be able to fix all of the security problems by yourself, but that's all right. The most important thing is to realize that a security problem might exist. Once you do that, it becomes a straightforward task to find others to help you fix the problem. But without the security mindset, you might never realize that your system has security problems and, therefore, you obviously can't protect against those problems in a principled way.

Technologies also change very rapidly. This means that some hot security mechanisms of today may be outdated in 10 or 15 years. But if you can learn how to think about security issues and have an appreciation for adversaries, then you can take that security mindset with you for the rest of your life and apply it to new technologies as they evolve.

1.4.2 Discussing Attacks

Professional paranoia is an essential tool of the trade. With any new system you encounter, the first thing you think of is how you can break it. The sooner you find a weak spot, the sooner you learn more about the new system. Nothing is worse than working on a system for years, only to have somebody come up and say: ''But how about if I attack it this way . . . ?'' You really don't want to experience that ''Oops'' moment.

In this field, we make a very strict distinction between attacking somebody's work and attacking somebody personally. Any work is fair game. If somebody proposes something, it is an automatic invitation to attack it. If you break one of our systems, we will applaud the attack and tell everybody about it.[2] We constantly look for weaknesses in any system because that is the only way to learn how to make more secure systems. This is one thing you will have to learn: an attack on your work is not an attack on you. Also, when you attack a system, always be sure to criticize the system, not the designers. Personal attacks in cryptography will get you the same negative response as anywhere else.

But be aware that this acceptance of attacks may not extend to everyone working on a system—particularly if they are not familiar with the field

[2]Depending on the attack, we might kick ourselves for not finding the weakness ourselves, but that is a different issue.

of cryptography and computer security. Without experience in the security community, it is very easy for people to take criticism of their work as a personal attack, with all the resulting problems. It is therefore important to develop a diplomatic approach, even if it makes it initially difficult to get the message across. Being too vague and saying something like ''There might be some issues with the security aspects'' may not be productive, since it may get a noncommittal response like ''Oh, we'll fix it,'' even if the basic design is fundamentally flawed. Experience has shown us that the best way to get the message across technically is to be specific and say something like ''If you do this and this, then an attacker could do this,'' but such a statement may be felt as harsh by the recipient. Instead, you could begin by asking, ''Have you thought about what might happen if someone did this?'' You could then ease the designers of the system into a discussion of the attack itself. You might also consider complimenting them on the remaining strengths of their system, observe the challenges to building secure systems, and offer to help them fix their security problems if possible.

So the next time someone attacks the security of your system, try not to take it personally. And make sure that when you attack a system, you only focus on the technology, you don't criticize the people behind it, and you are sensitive to the fact that the designers may not be familiar with the culture of constructive criticism in the security community.

1.5 Threat Model

Every system can be attacked. There is no such thing as perfect security. The whole point of a security system is to provide access to some people and not to others. In the end, you will always have to trust some people in some way, and these people may still be able to attack your system.

It is very important to know what you are trying to protect, and against whom you wish to protect it. What are the assets of value? What are the threats? These sound like simple questions, but it turns out to be a much harder problem than you'd think. Since there's really no such thing as perfect security, when we say that a system is ''secure,'' what we are really saying is that it provides a sufficient level of security for our assets of interest against certain classes of threats. We need to assess the security of a system under the designated threat model.

Most companies protect their LAN with a firewall, but many of the really harmful attacks are performed by insiders, and a firewall does not protect against insiders at all. It doesn't matter how good your firewall is; it won't protect against a malicious employee. This is a mismatch in the threat model.

Another example is SET. SET is a protocol for online shopping with a credit card. One of its features is that it encrypts the credit card number so that

an eavesdropper cannot copy it. That is a good idea. A second feature—that not even the merchant is shown the customer's credit-card number—works less well.

The second property fails because some merchants use the credit card number to look up customer records or to charge surcharges. Entire commerce systems have been based on the assumption that the merchant has access to the customer's credit card number. And then SET tries to take this access away. When Niels worked with SET in the past, there was an option for sending the credit card number twice—once encrypted to the bank, and once encrypted to the merchant so that the merchant would get it too. (We have not verified whether this is still the case.)

But even with this option, SET doesn't solve the whole problem. Most credit card numbers that are stolen are not intercepted while in transit between the consumer and the merchant. They are stolen from the merchant's database. SET only protects the information while it is in transit.

SET makes another, more serious, mistake. Several years ago Niels's bank in the Netherlands offered a SET-enabled credit card. The improved security for online purchases was one of the major selling points. But this turned out to be a bogus argument. It is quite safe to order online with a normal credit card. Your credit card number is not a secret. You give it to every salesperson you buy something from. The real secret is your signature. That is what authorizes the transaction. If a merchant leaks your credit card number, then you might get spurious charges, but as long as there is no handwritten signature (or PIN code) there is no indication of acceptance of the transaction, and therefore no legal basis for the charge. In most jurisdictions you simply complain and get your money back. There might be some inconvenience involved in getting a new credit card with a different number, but that is the extent of the user's exposure. With SET, the situation is different. SET uses a digital signature (explained in Chapter 12) by the user to authorize the transaction. That is obviously more secure than using just a credit card number. But think about it. Now the user is liable for any transaction performed by the SET software on his PC. This opens the user up to huge liabilities. What if a virus infects his PC and subverts the SET software? The software might sign the wrong transaction, and cause the user to lose money.

So from the user's point of view, SET offers *worse* security than a plain credit card. Plain credit cards are safe for online shopping because the user can always get his money back from a fraudulent transaction. Using SET increases the user's exposure. So although the overall payment system is better secured, SET transfers the residual risk from the merchant and/or bank to the user. It changes the user's threat model from ''It will only cost me money if they forge my signature well enough'' to ''It will only cost me money if they forge my signature well enough, or if a clever virus infects my PC.''

Threat models are important. Whenever you start on a cryptographic security project, sit down and think about what your assets are and against which threats you wish to protect them. A mistake in your threat analysis can render an entire project meaningless. We won't talk a lot about threat analysis in this book, as we are discussing the limited area of cryptography here, but in any real system you should never forget the threat analysis for each of the participants.

1.6 Cryptography Is Not the Solution

Cryptography is not the solution to your security problems. It might be part of the solution, or it might be part of the problem. In some situations, cryptography starts out by making the problem worse, and it isn't at all clear that using cryptography is an improvement. The correct use of cryptography must therefore be carefully considered. Our previous discussion of SET is an example of this.

Suppose you have a secret file on your computer that you don't want others to read. You could just protect the file system from unauthorized access. Or you could encrypt the file and protect the key. The file is now encrypted, and human nature being what it is, you might not protect the file very well. You might store it on a USB stick and not worry if that USB stick is lost or stolen. But where can you store the key? A good key is too long to remember. Some programs store the key on the disk—the very place the secret file was stored in the first place. But an attack that could recover the secret file in the first situation can now recover the key, which in turn can be used to decrypt the file. Further, we have introduced a new point of attack: if the encryption system is insecure or the amount of randomness in the key is too low, then the attacker could break the encryption system itself. Ultimately, the overall security has been reduced. Therefore, simply encrypting the file is not the entire solution. It might be part of the solution, but by itself it can create additional issues that need to be solved.

Cryptography has many uses. It is a crucial part of many good security systems. It can also make systems weaker when used in inappropriate ways. In many situations, it provides only a feeling of security, but no actual security. It is tempting to stop there, since that is what most users want: to *feel* secure. Using cryptography in this manner can also make a system comply with certain standards and regulations, even if the resulting system isn't actually secure. In situations like this (which are all too common), any voodoo that the customer believes in would provide the same feeling of security and would work just as well.

1.7 Cryptography Is Very Difficult

Cryptography is fiendishly difficult. Even seasoned experts design systems that are broken a few years later. This is common enough that we are not surprised when it happens. The weakest-link property and the adversarial setting conspire to make life for a cryptographer—or any security engineer—very hard.

Another significant problem is the lack of testing. There is no known way of testing whether a system is secure. In the security and cryptography research community, for example, what we try to do is publish our systems and then get other experts to look at them. Note that the second part is not automatic; there are many published systems that nobody has even glanced at after they were published, and the conference and journal review process alone isn't sufficient to preemptively identify all potential security issues with a system prior to publication. Even with many seasoned eyes looking at the system, security deficiencies may not be uncovered for years.

There are some small areas of cryptography that we as a community understand rather well. This doesn't mean they are simple; it just means that we have been working on them for a few decades now, and we think we know the critical issues. This book is mostly about those areas. What we have tried to do in this book is to collect the information that we have about designing and building practical cryptographic systems, and bring it all together in one place.

For some reason, many people still seem to think that cryptography is easy. It is not. This book will help you understand the challenges to cryptography engineering and help propel you on the road to overcoming those challenges. But don't go out and build a new cryptographic voting machine or other critical security system right away. Instead, take what you learn here and work with others—especially seasoned cryptography experts—to design and analyze your new system. Even we, despite our years of experience in cryptography and security, ask other cryptography and security experts to review the systems that we design.

1.8 Cryptography Is the Easy Part

Even though cryptography itself is difficult, it is still one of the easy parts of a security system. Like a lock, a cryptographic component has fairly well-defined boundaries and requirements. An entire security system is much more difficult to clearly define, since it involves many more aspects. Issues like the organizational procedures used to grant access and the procedures used to check that the other procedures are being followed are much harder to deal

with, as the situation is always changing. Another huge problem in computer security is the quality of much software. Security software cannot be effective if the software on the machine contains numerous bugs that lead to security holes.

Cryptography is the easy part, because there are people who know how to do a reasonably good job. There are experts for hire who will design a cryptographic system for you. They are not cheap, and they are often a pain to work with. They insist on changing other parts of the system to achieve the desired security properties. Still, for all practical purposes, cryptography poses problems that we know how to solve, and this book will give you a sense for how to go about solving them.

The rest of the security system contains problems we don't know how to solve. Key management and key storage is crucial to any cryptographic system, but most computers have no secure place to store a key. Poor software quality is another problem. Network security is even harder. And when you add users to the mix, the problem becomes harder still.

1.9 Generic Attacks

It is also important to realize that some security problems can't be solved. There are black box or generic attacks against certain types of systems. A classic example of this is the analog hole for digital rights management (DRM) systems. These DRM systems try to control the copying of digital materials, such as a picture, song, movie, or book. But no technology—cryptography or otherwise—can protect against a generic attack outside the system. For example, an attacker could take a photo of a computer screen to create a copy of the picture, or use a microphone to re-record the song.

It is important to identify what the generic attacks against a system are. Otherwise, you might spend a lot of time trying to fix an unfixable problem. Similarly, when someone claims that they've secured a system against a generic attack, you know to be skeptical.

1.10 Security and Other Design Criteria

Security is never the only design criterion for a system. Instead, security is but one of many criteria.

1.10.1 Security Versus Performance

The bridge over the Firth of Forth in Scotland has to be seen to be believed. A 19th-century engineering marvel, it is mind-numbingly large (and therefore expensive) compared to the trains that cross it. It is so incredibly

over-engineered it is hard to believe your eyes. Yet the designers did the right thing. They were confronted with a problem they had not solved successfully before: building a large steel bridge. They did an astoundingly good job. They succeeded spectacularly; their bridge is still in use today over a century later. That's what good engineering looks like.

Over the years, bridge designers have learned how to build such bridges much more cheaply and efficiently. But the first priority is always to get a bridge that is safe and that works. Efficiency, in the form of reducing cost, is a secondary issue.

We have largely reversed these priorities in the computer industry. The primary design objective all too often includes very strict efficiency demands. The first priority is always speed, even in areas where speed is not important. Here speed might be the speed of the system itself, or it might be the speed with which the system can be brought to market. This leads to security cost-cutting. The result is generally a system that is somewhat efficient, yet is not sufficiently secure.

There is another side to the Firth of Forth bridge story. In 1878, Thomas Bouch completed the then-longest bridge in the world across the Firth of Tay at Dundee. Bouch used a new design combining cast iron and wrought iron, and the bridge was considered to be an engineering marvel. On the night of December 28, 1879, less than two years later, the bridge collapsed in a heavy storm as a train with 75 people on board crossed the bridge. All perished. It was the major engineering disaster of the time.[3] So when the Firth of Forth bridge was designed a few years later, the designers put in a lot more steel, not only to make the bridge safe but also to make it *look* safe to the public.

We all know that engineers will sometimes get a design wrong, especially when they do something new. And when they get it wrong, bad things can happen. But here is a good lesson from Victorian engineers: if it fails, back off and become more conservative. The computer industry has largely forgotten this lesson. When we have very serious security failures in our computer systems, and we have them all too frequently, it is very easy to just plod along, accepting it as if it were fate. We rarely go back to the drawing board and design something more conservative. We just keep throwing a few patches out and hoping this will solve the problem.

By now, it will be quite clear to you that we will choose security over efficiency any time. How much CPU time are we willing to spend on security? Almost all of it. We wouldn't care if 90% of our CPU cycles were spent on a reliable security system if the alternative was a faster but insecure system. The lack of computer security is a real hindrance to us, and to most users. That is

[3]William McGonagall wrote a famous poem about the Tay Bridge disaster, ending with the lines *For the stronger we our houses do build/The less chance we have of being killed.* This advice is still highly relevant today.

why people still have to send pieces of paper around with signatures, and why they have to worry about viruses and other attacks on our computers. Digital crooks of the future will know much more and be much better equipped, and computer security will become a larger and larger problem. We are still only seeing the beginnings of the digital crime wave. We need to secure our computers much better.

There are of course many ways of achieving security. But as Bruce extensively documented in *Secrets and Lies*, good security is always a mixture of prevention, detection, and response [114]. The role for cryptography is primarily in the prevention part, which has to be very good to ensure that the detection and response parts (which can and should include manual intervention) are not overwhelmed. Cryptography can, however, be used to provide more secure detection mechanisms, such as strong cryptographic audit logs. Cryptography is what this book is about, so we'll concentrate on that aspect.

Yes, yes, we know, 90% still sounds like a lot. But there is some consolation. Remember first that we are willing to spend 90% of our CPU on security if the alternative is an insecure system. Fortunately, in many cases the costs of security can be hidden from the user. We can only type around 10 characters per second—on a good day—and even the slow machines of a decade ago had no trouble keeping up with that. Today's machines are over a thousand times faster. If we use 90% of the CPU for security, the computer will appear one-tenth as fast. That is about the speed that computers were five years ago. And those computers were more than fast enough for us to get our work done. We may not always have to spend so many cycles on security. But we're willing to, and that's the point.

There are only a few situations in which we have to wait on the computer. These include waiting for Web pages, printing data, starting certain programs, booting the machine, etc. A good security system would not slow down any of these activities. Modern computers are so fast that it is hard to figure out how to use the cycles in a useful manner. Sure, we can use alpha-blending on screen images, 3D animations, or even voice recognition. But the number-crunching parts of these applications do not perform any security-related actions, so they would not be slowed down by a security system. It is the rest of the system, which is already as fast as it can possibly get on a human time scale, that will have the overhead. And we don't care if it goes at one-tenth the speed if it increases security. Most of the time, you wouldn't even notice the overhead. Even in situations where the overhead would be significant, that is just the cost of doing business.

It will be clear by now that our priorities are security first, second, and third, and performance somewhere way down the list. Of course, we still want the system to be as efficient as possible, but not at the expense of security. We understand that this design philosophy is not always possible in the real world. Often the realities of the marketplace trump the need for security.

Systems can rarely be developed from scratch, and often need to be secured incrementally or after deployment. Systems need to be backward-compatible with existing, insecure, systems. The three of us have designed many security systems under these constraints, and we can tell you that it's practically impossible to build a good security system that way. The design philosophy of this book is security first and security foremost. It's one we'd like to see adopted more in commercial systems.

1.10.2 Security Versus Features

Complexity is the worst enemy of security, and it almost always comes in the form of features or options.

Here is the basic argument. Imagine a computer program with 20 different options, each of which can be either on or off. That is more than a million different configurations. To get the program to work, you only need to test the most common combination of options. To make the program secure, you must evaluate each of the million possible configurations that the program can have, and check that each configuration is secure against every possible form of attack. That is impossible to do. And most programs have considerably more than 20 options. The best way to have confidence in building something secure is to keep it simple.

A simple system is not necessarily a small system. You can build large systems that are still fairly simple. Complexity is a measure of how many things interact at any one point. If the effect of an option is limited to a small part of the program, then it cannot interact with an option whose effect is limited to another part of the program. To make a large, simple system you have to provide a very clear and simple interface between different parts of the system. Programmers call this modularization. This is all basic software engineering. A good simple interface isolates the rest of the system from the details of a module. And that should include any options or features of the module.

One of the things we have tried to do in this book is define simple interfaces for cryptographic primitives. No features, no options, no special cases, no extra things to remember, just the simplest definition we could come up with. Some of these definitions are new; we developed them while writing the book. They have helped us shape our thinking about good security systems, and we hope they will help you, too.

1.10.3 Security Versus Evolving Systems

One of the other biggest problems for security is that the full system continues to evolve even after the underlying security mechanisms are put in place. This means that the designer of the security mechanism needs not only to exhibit

professional paranoia and consider a wide range of attackers and attack goals, but also to anticipate and prepare for future uses of the system. This can also create enormous challenges, and is an issue that systems designers need to keep in mind.

1.11 Further Reading

Anyone interested in cryptography should read Kahn's *The Codebreakers* [67]. This is a history of cryptography, from ancient times to the 20th century. The stories provide many examples of the problems engineers of cryptographic systems face. Another good historical text, and a pleasurable read, is *The Code Book* [120].

In some ways, the book you're holding is a sequel to Bruce's first book, *Applied Cryptography* [112]. *Applied Cryptography* covers a much broader range of subjects, and includes the specifications of all the algorithms it discusses. However, it does not go into the engineering details that we talk about in this book.

For facts and precise results, you can't beat the *Handbook of Applied Cryptography*, by Menezes, van Oorschot, and Vanstone [90]. It is an encyclopedia of cryptography and an extremely useful reference book; but just like an encyclopedia, it's hardly a book to learn the field from.

If you're interested in the theory of cryptography, an excellent sequence of texts is *Foundations of Cryptography*, by Goldreich [55, 56]. Another excellent text is *Introduction to Modern Cryptography*, by Katz and Lindell [68]. There are also numerous excellent university course notes available online, such as the course notes from Bellare and Rogaway [10].

Bruce's previous book *Secrets and Lies* [114] is a good explanation of computer security in general, and how cryptography fits into that larger picture. And there's no better book on security engineering than Ross Anderson's *Security Engineering* [2]. Both are essential to understand the context of cryptography.

There are a number of good online resources for staying abreast of recent issues in cryptography and computer security. We suggest subscribing to Bruce's Crypto-Gram newsletter, `http://www.schneier.com/crypto-gram.html`, and reading Bruce's blog, `http://www.schneier.com/blog/`.

1.12 Exercises for Professional Paranoia

They say that one of the best ways to learn a foreign language is to immerse yourself in it. If you want to learn French, move to France. This book is designed to immerse you in the language and mindset of cryptography and

computer security. The following exercises will help immerse you further. They will force you to think about security on a regular basis, such as when you're reading news articles, talking with friends about current events, or reading the description of a new product on Slashdot. Thinking about security will no longer be a chore relegated to the time when you are specifically tasked with thinking about security. You may even start thinking about security while you're out walking your dog, in the shower, or at a movie. In short, you will be developing the professional paranoia mindset and will start thinking like a security professional.

It is also extremely important for a computer security practitioner (and, actually, all computer scientists) to be aware of the broader contextual issues surrounding technology. Technologies don't exist in isolation. Rather, they are one small aspect of a larger ecosystem consisting of people, economics, ethics, cultural differences, politics, law, and so on. These exercises will also give you an opportunity to discuss and explore these bigger picture issues as they relate to security.

We suggest that you regularly return to the exercises below. Try to do these exercises as often as possible. For example, you might do these exercises every week for a month straight, or after you finish every few chapters in this book, whichever is more frequent. The exercises might seem laborious and tedious at first. But if you're dedicated to this practice, you will soon find yourself doing these exercises automatically whenever you encounter a security-related news article or see a new product. This is the professional paranoia mindset. Further, if you continue to do these exercises as you read this book, you will notice that your ability to evaluate the technical properties of systems will mature over time.

We also recommend doing the exercises with a friend or, if you are in a class, with a classmate as part of group instruction. Discussing security issues with others can be very enlightening—you will soon realize firsthand that security is incredibly subtle and that it is very easy to overlook critical weaknesses.

Obviously, if you're not taking a class and doing the formal exercises, then you may choose to conduct these exercises in your head rather than actually producing written reports. Still, we suggest producing a written report at least once; doing so will force you to really think through the relevant issues completely.

1.12.1 Current Event Exercises

For these exercises, you should critically analyze some event currently in the news. The event you choose should somehow relate to computer security. Maybe improved computer security mechanisms would have thwarted the event. Maybe the event motivates the design of new security mechanisms or policies.

The current events retrospective that you write should be short, concise, very thoughtful, and well written. Assume a general audience. Your goal should be to write an article that will help the reader learn about and understand the computer security field and how it fits into the broader context.

You should summarize the current event, discuss why the current event arose, reflect on what could have been done differently prior to the event arising (to perhaps prevent, deter, or alter the consequences of the event), describe the broader issues surrounding the current event (such as ethical issues or societal issues), and propose possible reactions to the current event (e.g., how the public, policy makers, corporations, the media, or others should respond).

1.12.2 Security Review Exercises

These exercises deal with developing your security mindset in the context of real products or systems. Your goal with the security reviews is to evaluate the potential security and privacy issues of new technologies, evaluate the severity of those issues, and discuss how to address those security and privacy issues. These reviews should reflect deeply on the technology that you're discussing, and should therefore be significantly longer than your current event exercises.

Each security review should contain:

- Summary of the technology that you're evaluating. You may choose to evaluate a specific product (like a recently introduced wireless implantable drug pump) or a class of products with some common goal (like the set of all implantable medical devices). This summary should be at a high level, around one or two paragraphs in length. State the aspects of the technology that are relevant to your observations in the following bullets.

  For these exercises, it is acceptable to make assumptions about how the products work. However, if you do make assumptions about a product, then you should make it clear that you are doing so, and you should explicitly state what those assumptions are.

  Being able to clearly summarize a product (even with explicitly stated assumptions) is very important. If you don't understand the technology well enough to provide a crisp and clear summary, then you probably don't understand the technology well enough to evaluate its security and privacy.

- State at least two assets and, for each asset, a corresponding security goal. Explain why the security goals are important. You should produce around one or two sentences per asset/goal.

- State at least two possible threats, where a threat is defined as an action by an adversary aimed at compromising an asset. Give an example adversary for each threat. You should have around one or two sentences per threat/adversary.
- State at least two potential weaknesses. Again, justify your answer using one or two sentences per weakness. For the purposes of these exercises, you don't need to fully verify whether these potential weaknesses are also actual weaknesses.
- State potential defenses. Describe potential defenses that the system could use or might already be using to address the potential weaknesses you identified in the previous bullet.
- Evaluate the risks associated with the assets, threats, and potential weaknesses that you describe. Informally, how serious do you think these combinations of assets, threats, and potential weaknesses are?
- Conclusions. Provide some thoughtful reflections on your answers above. Also discuss relevant ''bigger picture'' issues (ethics, likelihood the technology will evolve, and so on).

Some examples of past security reviews are online at `http://www.schneier.com/ce.html`.

1.13 General Exercises

Exercise 1.1 Create an attack tree for stealing a car. For this and the other attack tree exercises, you can present your attack tree as a figure (like Figure 1.1), or you can present your attack tree as a list numbered in outline form (e.g., 1, 1.1, 1.2, 1.2.1, 1.2.2, 1.3, ...).

Exercise 1.2 Create an attack tree for getting into a gym without paying.

Exercise 1.3 Create an attack tree for getting food from a restaurant without paying.

Exercise 1.4 Create an attack tree for learning someone's online banking account name and password.

Exercise 1.5 Create an attack tree for reading someone else's e-mail.

Exercise 1.6 Create an attack tree for preventing someone from being able to read his own e-mail.

Exercise 1.7 Create an attack tree for sending e-mail as someone else. Here, the attacker's goal is to convince an e-mail recipient that an e-mail she receives is from someone else (say, Bob), when in fact Bob never sent that e-mail.

Exercise 1.8 Find a new product or system that was announced or released within the last three months. Conduct a security review of that product or system as described in Section 1.12. Pick one of the assets that you identified and construct an attack tree for compromising that asset.

Exercise 1.9 Provide a concrete example, selected from media reports or your personal experiences, in which attackers compromised a system by exploiting something other than the weakest link. Describe the system, describe what you view the weakest link of the system to be and why, and describe how the system was compromised.

Exercise 1.10 Describe a concrete example, excluding the ones given in this chapter, where improving the security of a system against one type of attack can increase the likelihood of other attacks.