charge_payment_card function is invoked in whatever infrastructure is available

whenever a card needs to be processed. If that function is successful, it invokes the decrease_inventory function, again, in whatever infrastructure is available, and so on. After each function terminates, it simply evaporates so that no more resources are consumed than are absolutely needed. If there's a sudden spike in demand, the orchestrator spins up whatever additional resources are needed to run as many functions as are required.

Server-based

Microservices

Serverless

Client

Client

Client

Web server
Web Server

Web server

Charge
purchase
card

View
items
App server
Place
order

Database

Database
Database

Decrease
inventory

DB

DB

Cloud-Based Systems

If you were asked to install a brand-new server room for your organization, you would probably have to clear your calendar for weeks (or longer) to address the many tasks that would be involved. From power and environmental controls to hardware acquisition, installation, and configuration to software builds, the list is long and full of headaches.

Now, imagine that you can provision all the needed servers in minutes using a simple graphical interface or a short script and that you can get rid of them just as quickly when you no longer need them. This is one of the benefits of cloud computing.

Cloud computing is the use of shared, remote computing devices for the purpose of providing improved efficiencies, performance, reliability, scalability, and security. These devices are usually based on virtual machines running on shared infrastructure and can be outsourced to a third-party cloud service provider (CSP) on a public cloud or provided in-house on a private cloud. If you don't feel comfortable sharing infrastructure with random strangers (though this is done securely), there is also a virtual private cloud (VPC) model in which you get your own walled garden inside an otherwise public cloud.

Web server

Generally speaking, there are three models for cloud computing services:

- Software as a Service (SaaS) The user of SaaS is allowed to use a specific application that executes on the CSP's environment. Examples of SaaS are Microsoft 365 and Google Apps, which you use via a web interface but someone else provisions and maintains everything for you.
- Platform as a Service (PaaS) In this model, the user gets access to a computing platform that is typically built on a server operating system. An example of this would be spawning an instance of Windows Server 2019 to provide a web server. The CSP is normally responsible for configuring and securing the platform, however, so the user normally doesn't get administrative privileges over the entire platform.
- Infrastructure as a Service (IaaS) If you want full, unfettered access to (and responsibility for securing) a cloud-based VM, you would want to use the IaaS model. Following up on the previous example, this would allow you to manage the patching of the Windows Server 2019 instance. The catch is that the CSP has no responsibility for security; it's all on you.

If you are a user of IaaS, you probably won't do things too differently than you already
do to secure your systems. The only exception is that you wouldn't have physical access
to the computers if a CSP hosts them. If, on the other hand, you use SaaS or PaaS, the
security of your systems will almost always rely on the policies and contracts that you
put into place. The policies will dictate how your users interact with the cloud services.
This would include the information classification levels that would be allowed on those
services, terms of use, and other policies. The contracts will specify the quality of service
and what the CSP will do with or for you in responding to security events.
CAUTION It is imperative that you carefully review the terms of service when
evaluating a potential contract for cloud services and consider them in the
context of your organization's security. Though the industry is getting better
all the time, security provisions are oftentimes lacking in these contracts
at this time.

Software as a Service
SaaS is pervasively used by most enterprises. According to some estimates, the average
company uses nearly 2,000 unique cloud services for everything from writing memos to
managing their sales pipeline. The whole idea is that, apart from a fairly small amount of
allowed customization, you just pay for the licenses and the vendor takes care of making
sure all your users have access to the software, regardless of where they are.
Given the popularity of SaaS solutions, cloud service providers such as Microsoft,
Amazon, Cisco, and Google often dedicate large teams to securing all aspects of their
service infrastructure. Increasingly, however, most security incidents involving SaaS
occur at the data-handling level, where these infrastructure companies do not have the

responsibility or visibility required to take action. For example, how could the CSP be
held liable when one of your employees shares a confidential file with an unauthorized
third party?
So, visibility is one of our main concerns as security professionals when it comes to
SaaS. Do you know what assets you have and how they are being used? The "McAfee
2019 Cloud Adoption and Risk Report" describes the disconnect between the number
of cloud services that organizations believe are being accessed by their users and the

number of cloud services that are actually being accessed. The discrepancy, according to the report, can be several orders of magnitude. As we have mentioned before, you can't protect what you don't know you have. This is where solutions like cloud access security brokers (CASBs) and data loss prevention (DLP) systems can come in very handy.

NOTE We already covered CASBs and DLP systems in Chapter 6.

PART III

Platform as a Service

What if, instead of licensing someone else's application, you have developed your own and need a place to host it for your users? You'd want to have a fair amount of flexibility in terms of configuring the hosting environment, but you probably could use some help in terms of provisioning and securing it. You can secure the app, for sure, but would like someone else to take care of things like hardening the host, patching the underlying OS, and maybe even monitoring access to the VM. This is where PaaS comes in.

PaaS has a similar set of functionalities as SaaS and provides many of the same benefits in that the CSP manages the foundational technologies of the stack in a manner transparent to the end user. You simply tell your provider, "I'd like a Windows Server 2019 with 64 gigabytes of RAM and eight cores," and, voilà, there it is. You get direct access to a development or deployment environment that enables you to build and host your own solutions on a cloud-based infrastructure without having to build your own infrastructure.

PaaS solutions, therefore, are optimized to provide value focused on software development.

PaaS, by its very nature, is designed to provide an organization with tools that interact directly with what may be its most important asset: its source code.

At the physical infrastructure, in PaaS, service providers assume the responsibility of maintenance and protection and employ a number of methods to deter successful exploits at this level. This often means PaaS providers require trusted sources for hardware, use strong physical security for its data centers, and monitor access to the physical servers and connections to and from them. Additionally, PaaS providers often enhance their protection against distributed denial-of-service (DDoS) attacks using network-based technologies that require no additional configuration from the user.

While the PaaS model makes a lot of provisioning, maintenance, and security problems go away for you, it is worth noting that it does nothing to protect the software systems you host there. If you build and deploy insecure code, there is very little your CSP will be able to do to keep it protected. PaaS providers focus on the infrastructure on which the

service runs, but you still have to ensure that the software is secure and the appropriate controls are in place. We'll dive into how to build secure code in Chapters 24 and 25.

Infrastructure as a Service

Sometimes, you just have to roll up your sleeves, get your hands dirty, and build your own servers from the ground up. Maybe the applications and services you have developed require your IT and security teams to install and configure components at the OS level that would not be accessible to you in the PaaS model. You don't need someone to make platforms that they manage available to you; you need to build platforms from the ground up yourself. IaaS gives you just that. You upload an image to the CSP's environment and build your own hosts however you need them.

As a method of efficiently assigning hardware through a process of constant assignment and reclamation, IaaS offers an effective and affordable way for organizations to get all of the benefits of managing their own hardware without incurring the massive overhead costs associated with acquisition, physical storage, and disposal of the hardware. In this service model, the vendor provides the hardware, network, and storage resources necessary for the user to install and maintain any operating system, dependencies, and applications they want. The vendor deals with all hardware issues for you, leaving you to focus on the virtual hosts.

In the IaaS model, the majority of the security controls (apart from physical ones) are your responsibility. Obviously, you want to have a robust security team to manage these. Still, there are some risks that are beyond your control and for which you rely on your vendor, such as any vulnerabilities that could allow an attacker to exploit flaws in hard disks, RAM, CPU caches, and GPUs. One attack scenario affecting IaaS cloud providers could enable a malicious actor to implant persistent back doors for data theft into baremetal cloud servers. A vulnerability either in the hypervisor supporting the visualization of various tenant systems or in the firmware of the hardware in use could introduce a vector for this attack. This attack would be difficult for the customer to detect because it would be possible for all services to appear unaffected at a higher level of the technology stack. Though the likelihood of a successful exploit of this kind of vulnerability is quite low, defects and errors at this level may still incur significant costs unrelated to an actual exploit. Take, for example, the 2014 hypervisor update performed by Amazon Web Services (AWS), which essentially forced a complete restart of a major cloud offering, the Elastic Compute Cloud (EC2). In response to the discovery of a critical security flaw in the open-source hypervisor Xen, Amazon forced EC2 instances globally to restart to ensure the patch would take correctly and that customers remained unaffected. In most cases, though, as with many other cloud services, attacks against IaaS environments are possible because of misconfiguration on the customer side.

Everything as a Service

It’s worth reviewing the basic premise of cloud service offerings: you save money by only paying for exactly the resources you actually use, while having the capacity to scale those up as much as you need to at a moment’s notice. If you think about it, this model can

apply to things other than applications and computers. Everything as a Service (XaaS) captures the trend to apply the cloud model to a large range of offerings, from entertainment (e.g., television shows and feature-length movies), to cybersecurity (e.g., Security as a Service), to serverless computing environments (e.g., Function as a Service). Get ready for the inevitable barrage of <fill-in-the-blank> as a Service offerings coming your way.

Cloud Deployment Models

By now you may be a big believer in the promise of cloud computing but may be wondering, “Where, exactly, is the cloud?” The answer, as in so many questions in our field, is “It depends.” There are four common models for deploying cloud computing resources, each with its own features and limitations:

Pervasive Systems

Cloud computing is all about the concentration of computing power so that it may be dynamically reallocated among customers. Going in the opposite conceptual direction, pervasive computing (also called ubiquitous computing or ubicomp) is the concept that small (even tiny) amounts of computing power are spread out everywhere and computing is embedded into everyday objects that communicate with each other, often with little or no user interaction, to do very specific things for particular customers. In this model, computers are everywhere and communicate on their own with each other, bringing really cool new features but also really thorny new security challenges.

• A public cloud is the most prevalent model, in which a vendor like AWS owns all the resources and provides them as a service to all its customers. Importantly, the resources are shared among all customers, albeit in a transparent and secure manner. Public cloud vendors typically also offer a virtual private cloud (VPC) as an option, in which increased isolation between users provides added security.
• A private cloud is owned and operated by the organization that uses its services. Here, you own, operate, and maintain the servers, storage, and networking needed to provide the services, which means you don't share resources with anyone. This approach can provide the best security, but the tradeoff might be higher costs and a cap on scalability.
• A community cloud is a private cloud that is co-owned (or at least shared) by a specific set of partner organizations. This approach is commonly implemented in large conglomerates where multiple firms report to the same higher-tier headquarters.
• A hybrid cloud combines on-premises infrastructure with a public cloud, with a significant effort placed in the management of how data and applications leverage each solution to achieve organizational goals. Organizations that use a hybrid model often derive benefits offered by both public and private models.

Embedded Systems

An embedded system is a self-contained computer system (that is, it has its own processor, memory, and input/output devices) designed for a very specific purpose. An embedded device is part of (or embedded into) some other mechanical or electrical device or system. Embedded systems typically are cheap, rugged, and small, and they use very little power.

They are usually built around microcontrollers, which are specialized devices that consist of a CPU, memory, and peripheral control interfaces. Microcontrollers have a very basic operating system, if they have one at all. A digital thermometer is an example of a very simple embedded system; other examples of embedded systems include traffic lights and factory assembly line controllers. As you can see from these examples, embedded systems are frequently used to sense and/or act on a physical environment. For this reason, they are sometimes called cyber-physical systems.

The main challenge in securing embedded systems is that of ensuring the security of the software that drives them. Many vendors build their embedded systems around commercially available microprocessors, but they use their own proprietary code that is difficult, if not impossible, for a customer to audit. Depending on the risk tolerance of your organization, this may be acceptable as long as the embedded systems are standalone. The problem, however, is that these systems are increasingly shipping with some sort of network connectivity. For example, some organizations have discovered that some of their embedded devices have “phone home” features that are not documented. In some cases, this has resulted in potentially sensitive information being transmitted to the manufacturer. If a full audit of the embedded device security is not possible, at a very minimum, you should ensure that you see what data flows in and out of it across any network.

Another security issue presented by many embedded systems concerns the ability to update and patch them securely. Many embedded devices are deployed in environments where they have no Internet connectivity. Even if this is not the case and the devices can check for updates, establishing secure communications or verifying digitally signed code, both of which require processor-intensive cryptography, may not be possible on a cheap device.

Internet of Things

The Internet of Things (IoT) is the global network of connected embedded systems. What distinguishes the IoT is that each node is connected to the Internet and is uniquely addressable. By some accounts, this network is expected to reach 31 billion devices by 2025, which makes this a booming sector of the global economy. Perhaps the most visible aspect of this explosion is in the area of smart homes in which lights, furnaces, and even refrigerators collaborate to create the best environment for the residents. With this level of connectivity and access to physical devices, the IoT poses many security challenges. Among the issues to address by anyone considering adoption of IoT devices are the following:

• Authentication Embedded devices are not known for incorporating strong authentication support, which is the reason why most IoT devices have very poor (if any) authentication.

• Encryption Cryptography is typically expensive in terms of processing power and memory requirements, both of which are very limited in IoT devices. The fallout of this is that data at rest and data in transit can be vulnerable in many parts of the IoT.

• Updates Though IoT devices are networked, many vendors in this fast-moving sector do not provide functionality to automatically update their software and firmware when patches are available.

Distributed Systems

A distributed system is one in which multiple computers work together to do something. The earlier section “Server-Based Systems” already covered a specific example of a fourtier distributed system. It is this collaboration that more generally defines a distributed system. A server-based system is a specific kind of distributed system in which devices in one group (or tier) act as clients for devices in an adjacent group. A tier-1 client cannot work directly with the tier-4 database, as shown earlier in Figure 7-1. We could then say that a distributed system is any system in which multiple computing nodes, interconnected by a network, exchange information for the accomplishment of collective tasks.

Not all distributed systems are hierarchical like the example in Figure 7-1. Another approach to distributed computing is found in peer-to-peer systems, which are systems in which each node is considered an equal (as opposed to a client or a server) to all others. There is no overarching structure, and nodes are free to request services from any other node. The result is an extremely resilient structure that fares well even when large numbers of nodes become disconnected or otherwise unavailable. If you had a typical client/server model and you lost your server, you’d be down for the count. In a peer-topeer system, you could lose multiple nodes and still be able to accomplish whatever task you needed to. Clearly, not every application lends itself to this model, because some tasks are inherently hierarchical or centralized. Popular examples of peer-to-peer systems are file sharing systems like BitTorrent, anonymizing networks like The Onion Router (TOR), and cryptocurrencies like bitcoin.

One of the most important issues in securing distributed systems is network communications, which are essential to these systems. While the obvious approach would be to encrypt all traffic, it can be challenging to ensure all nodes are using cryptography that is robust enough to mitigate attacks. This is particularly true when the

Perhaps the most dramatic illustration to date of what can happen when millions of insecure IoT devices are exploited by an attacker is the Mirai botnet. Mirai is a malware strain that infects IoT devices and was behind one of the largest and most effective botnets in recent history. The Mirai botnet took down major websites via massive DDoS attacks against several sites and service providers using hundreds of thousands of compromised IoT devices. In October 2016, a Mirai attack targeted the popular DNS provider Dyn, which provided name resolution to many popular websites such as Airbnb, Amazon, GitHub, HBO, Netflix, PayPal, Reddit, and Twitter. After taking down Dyn, Mirai left millions of users unable to access these sites for hours.

system includes IoT or OT components that may not have the same crypto capabilities as traditional computers.

Even if you encrypt all traffic (and you really should) in a distributed system, there's still the issue of trust. How do we ensure that every user and every node is trustworthy? How could you tell if part of the system was compromised? Identity and access management is another key area to address, as is the ability to isolate users or nodes from the system should they become compromised.

NOTE We will discuss identity and access management (IAM) in Chapter 16.

Edge Computing Systems

An interesting challenge brought about by the proliferation of IoT devices is how to service them in a responsive, scalable, and cost-effective manner. To understand the problem, let’s first consider a server-based example. Suppose you enjoy playing a massively multiplayer online game (MMOG) on your web browser. The game company would probably host the backend servers in the cloud to allow massive scalability, so the processing power is not an issue. Now suppose all these servers were provisioned in the eastern United States. Gamers in New York would have no problem enjoying the game, but those in Japan would probably have noticeable network latency issues because every one of their commands would have to be sent literally around the world to be processed by the U.S. servers, and then the resulting graphics sent back around the world to the player in Japan. That player would probably lose interest in the game really quickly.

Now, suppose that the company kept its main servers in the United States but provisioned regional servers, with one of them in, say, Singapore. Most of the commands are processed in the regional server, which means that the user experience of players in Japan is a lot better, while the global leaderboard is maintained centrally in the United States. This is an example of edge computing.

Edge computing is an evolution of content distribution networks (CDNs), which were designed to bring web content closer to its clients. CDNs helped with internationalization of websites but were also very good for mitigating the effects of DDoS attacks. Edge computing is a distributed system in which some computational and data storage assets are deployed close to where they are needed in order to reduce latency and network traffic.

As shown in Figure 7-7, an edge computing architecture typically has three layers: end devices, edge devices, and cloud infrastructure. The end devices can be anything from smart thermometers to self-driving cars. They have a requirement for processing data in real time, which means there are fairly precise time constraints. Think of a thermal sensor in one of your data centers and how you would need to have an alarm within minutes (at most) of it detecting rising or excessive heat.

Global
cloud
services
Data center - West

Data center - East

Edge
device

Fire
alarms

Thermal
sensors

Door
sensors

Fire
alarms

Thermal
sensors

Figure 7-7 A sample edge computing architecture for facility management

To reduce the turnaround time for these computing requirements, we deploy edge devices that are closer to, and in some cases embedded within, the end devices. Returning to the thermometer example, suppose you have several of these devices in each of your two data centers. You also have a multitude of other sensors such as fire alarms and door sensors. Rather than configuring an alarm to sound whenever the data center gets too hot, you integrate all these sensors to develop an understanding of what is going in the facility. For example, maybe the temperature is rising because someone left the back door open on a hot summer day. If it keeps going up, you want to sound a door alarm, not necessarily a temperature alarm, and do it while there is still time for the cooling system to keep the ambient temperature within tolerance. The sensors (including the thermometer) would send their data to the edge device, which is located near or in the same facility. This reduces the time needed to compute solutions and also provides a degree of protection against network outages. The determination to sound the door alarm (and when) is made there, locally, at the edge device. All (or maybe some of) the data from all the sensors at both data centers is also sent to the global cloud services infrastructure. There, we can take our time and run data analytics to discover useful patterns that could tell us how to be more efficient in how we use our resources around

the world.
NOTE As increased computing power finds its way into IoT devices, these too are becoming edge devices in some cases.

Door
sensors

Edge
device

Chapter Review

Central to securing our systems is understanding their components and how they interact with each other—in other words, their architectures. While it may seem that architectural terminology overlaps quite a bit, in reality each approach brings some unique challenges and some not-so-unique challenges. As security professionals, we need to understand where architectures are similar and where they differ. We can mix and match, of course, but must also do so with a clear understanding of the underlying issues. In this chapter, we've classified the more common system architectures and discussed what makes them unique and what specific security challenges they pose. Odds are that you will encounter devices and systems in most, if not all, of the architectures we've covered here.

Quick Review

• Client-based systems execute all their core functions on the user's device and don't require network connectivity.
• Server-based systems require that a client make requests from a server across a network connection.
• Transactions are sequences of actions required to properly change the state of a database.
• Database transactions must be atomic, consistent, isolated, and durable (ACID).
• Aggregation is the act of combining information from separate sources and is a security problem when it allows unauthorized individuals to piece together sensitive information.
• Inference is deducing a whole set of information from a subset of its aggregated components. This is a security problem when it allows unauthorized individuals to infer sensitive information.
• High-performance computing (HPC) is the aggregation of computing power in ways that exceed the capabilities of general-purpose computers for the specific

purpose of solving large problems.
• Industrial control systems (ICS) consist of information technology that is specifically
designed to control physical devices in industrial processes.
• Any system in which computers and physical devices collaborate via the exchange
of inputs and outputs to accomplish a task or objective is an embedded or cyberphysical system.
• The two main types of ICS are distributed control systems (DCS) and supervisory
control and data acquisition (SCADA) systems. The main difference between them is that a DCS controls local processes while SCADA is used to control things remotely.
• ICS should always be logically or physically isolated from public networks.
• Virtualized systems are those that exist in software-simulated environments.
• Virtual machines (VMs) are systems in which the computing hardware has been virtualized for the operating systems running in them.

Questions
Please remember that these questions are formatted and asked in a certain way for a
reason. Keep in mind that the CISSP exam is asking questions at a conceptual level.
Questions may not always have the perfect answer, and the candidate is advised against
always looking for the perfect answer. Instead, the candidate should look for the best
answer in the list.
1. Which of the following lists two foundational properties of database transactions?
A. Aggregation and inference
B. Scalability and durability
C. Consistency and performance
D. Atomicity and isolation

• Containers are systems in which the operating systems have been virtualized for
the applications running in them.
• Microservices are software architectures in which features are divided into multiple
separate components that work together in a distributed manner across a network.
• Containers and microservices don’t have to be used together but it’s very common to do so.
• In a serverless architecture, the services offered to end users can be performed
without a requirement to set up any dedicated server infrastructure.
• Cloud computing is the use of shared, remote computing devices for the purpose of

providing improved efficiencies, performance, reliability, scalability, and security.

• Software as a Service (SaaS) is a cloud computing model that provides users access to a specific application that executes in the service provider's environment.

• Platform as a Service (PaaS) is a cloud computing model that provides users access to a computing platform but not to the operating system or to the virtual machine on which it runs.

• Infrastructure as a Service (IaaS) is a cloud computing model that provides users unfettered access to a cloud device, such as an instance of a server, which includes both the operating system and the virtual machine on which it runs.

• An embedded system is a self-contained, typically ruggedized, computer system with its own processor, memory, and input/output devices that is designed for a very specific purpose.

• The Internet of Things (IoT) is the global network of connected embedded systems.

• A distributed system is a system in which multiple computing nodes, interconnected by a network, exchange information for the accomplishment of collective tasks.

• Edge computing is a distributed system in which some computational and data storage assets are deployed close to where they are needed in order to reduce latency and network traffic.

2. Which of the following is not true about containers?
A. They are embedded systems.
B. They are virtualized systems.
C. They commonly house microservices.
D. They operate in a sandbox.

3. What is the term that describes a database attack in which an unauthorized user is able to combine information from separate sources to learn sensitive information to which the user should not have access?
A. Aggregation
B. Containerization
C. Serialization
D. Collection

4. What is the main difference between a distributed control system (DCS) and supervisory control and data acquisition (SCADA)?
A. SCADA is a type of industrial control system (ICS), while a DCS is a type of bus.
B. SCADA controls systems in close proximity, while a DCS controls physically distant ones.
C. A DCS controls systems in close proximity, while SCADA controls physically distant ones.
D. A DCS uses programmable logic controllers (PLCs), while SCADA uses remote terminal units (RTUs).

5. What is the main purpose of a hypervisor?
A. Virtualize hardware resources and manage virtual machines
B. Virtualize the operating system and manage containers
C. Provide visibility into virtual machines for access control and logging
D. Provide visibility into containers for access control and logging
6. Which cloud service model provides customers direct access to hardware, the network, and storage?
A. SaaS
B. PaaS
C. IaaS
D. FaaS
7. Which cloud service model do you recommend to enable access to developers to write custom code while also providing all employees access from remote offices?
A. PaaS
B. SaaS

C. FaaS
D. IaaS

8. Which of the following is not a major issue when securing embedded systems?
A. Use of proprietary code
B. Devices that “phone home”
C. Lack of microcontrollers
D. Ability to update and patch them securely

9. Which of the following is true about edge computing?
A. Uses no centralized computing resources, pushing all computation to the edge
B. Pushes computation to the edge while retaining centralized data management
D. Is an evolution of content distribution networks

Use the following scenario to answer Questions 10–12. You were just hired as director of cybersecurity for an electric power company with facilities around your country. Carmen is the director of operations and offers to give you a tour so you can see the security measures that are in place on the operational technology (OT).
10. What system would be used to control power generation, distribution, and delivery to all your customers?
A. Supervisory control and data acquisition (SCADA)
B. Distributed control system (DCS)
C. Programmable logic controller
D. Edge computing system

11. You see a new engineer being coached remotely by a more senior member of the staff in the use of the human-machine interface (HMI). Carmen tells you that senior engineers are allowed to access the HMI from their personal computers at home to facilitate this sort of impromptu training. She asks what you think of this policy. How should you respond?
A. Change the policy. They should not access the HMI with their personal

computers, but they could do so using a company laptop, assuming they also

use a virtual private network (VPN).
B. Change the policy. ICS devices should always be isolated from the Internet.
C. It is acceptable because the HMI is only used for administrative purposes and not operational functions.
D. It is acceptable because safety is the fundamental concern in ICS, so it is best
to let the senior engineers be available to train other staff from home.

C. Typically consists of two layers: end devices and cloud infrastructure

12. You notice that several ICS devices have never been patched. When you ask why,
Carmen tells you that those are mission-critical devices, and her team has no way
of testing the patches before patching these production systems. Fearing that patching them could cause unexpected outages or, worse, injure someone, she has authorized them to remain as they are. Carmen asks whether you agree. How could you respond?
A. Yes. As long as we document the risk and ensure the devices are as isolated

and as closely monitored as possible.
B. Yes. Safety and availability trump all other concerns when it comes to ICS security.
C. No. You should stand up a testing environment so you can safely test the patches and then deploy them to all devices.
D. No. These are critical devices and should be patched as soon as possible.

Answers

1. D. The foundational properties of database transactions are atomicity, consistency,
isolation, and durability (ACID).
2. A. Containers are virtualized systems that commonly (though not always) house microservices and run in sandboxes. It would be highly unusual to implement a container as an embedded system.
3. A. Aggregation happens when a user does not have the clearance or permission to access specific information, but she does have the permission to access components of this information. She can then figure out the rest and obtain restricted information.
4. C. The main difference is that a DCS controls devices within fairly close proximity,
while SCADA controls large-scale physical processes involving nodes separated by significant distances. They both can (and frequently use) PLCs, but RTUs are almost always seen in SCADA systems.
5. A. Hypervisors are almost always used to virtualize the hardware on which virtual
machines run. They can also provide visibility and logging, but these are secondary
functions. Containers are the equivalents of hypervisors, but they work at a higher

level by virtualizing the operating system.
6. C. Infrastructure as a Service (IaaS) offers an effective and affordable way for
organizations to get all the benefits of managing their own hardware without the massive overhead costs associated with acquisition, physical storage, and disposal
of the hardware.
7. A. Platform as a Service (PaaS) solutions are optimized to provide value focused
on software development, offering direct access to a development environment to enable an organization to build its own solutions on the cloud infrastructure,
rather than providing its own infrastructure.

8. C. Embedded systems are usually built around microcontrollers, which are specialized devices that consist of a CPU, memory, and peripheral control interfaces. All the other answers are major issues in securing embedded systems.
9. D. Edge computing is an evolution of content distribution networks, which were designed to bring web content closer to its clients. It is a distributed system
in which some computational and data storage assets are deployed close to where they are needed in order to reduce latency and network traffic. Accordingly, some
computing and data management is handled in each of three different layers: end devices, edge devices, and cloud infrastructure.
10. A. SCADA was designed to control large-scale physical processes involving nodes
separated by significant distances, as is the case with electric power providers.

12. A. It is all too often the case that organizations can afford neither the risk of
pushing untested patches to ICS devices nor the costs of standing up a testing environment. In these conditions, the best strategy is to isolate and monitor the
devices as much as possible.

11. B. It is a best practice to completely isolate ICS devices from Internet access.
Sometimes this is not possible for operational reasons, so remote access through a VPN could be allowed even though it is not ideal.

🡅This page intentionally left blank

🡅CHAPTER

Cryptology
This chapter presents the following:
• Principles of cryptology

- Symmetric cryptography
- Asymmetric cryptography
- Public key infrastructure
- Cryptanalytic attacks

Three can keep a secret, if two of them are dead.
—Benjamin Franklin

Now that you have a pretty good understanding of system architectures from Chapter 7, we turn to a topic that is central to protecting these architectures. Cryptography is the practice of storing and transmitting information in a form that only authorized parties can understand. Properly designed and implemented, cryptography is an effective way to protect sensitive data throughout its life cycle. However, with enough time, resources, and motivation, hackers can successfully attack most cryptosystems and reveal the information. So, a more realistic goal of cryptography is to make obtaining the information too work intensive or time consuming to be worthwhile to the attacker.

Cryptanalysis is the name collectively given to techniques that aim to weaken or defeat cryptography. This is what the adversary attempts to do to thwart the defender's use of cryptography. Together, cryptography and cryptanalysis comprise cryptology. In this chapter, we'll take a good look at both sides of this topic. This is an important chapter in the book, because we can't defend our information systems effectively without understanding applied cryptology.

The History of Cryptography

Cryptography has roots in antiquity. Around 600 ▯.▯., Hebrews invented a cryptographic method called atbash that required the alphabet to be flipped so each letter in the original message was mapped to a different letter in the flipped, or shifted, message. An example of an encryption key used in the atbash encryption scheme is shown here:

ABCDEFGHIJKLMNOPQRSTUVWXYZ
ZYXWVUTSRQPONMLKJIHGFEDCBA

If you want to encrypt the word "security" you would instead use "hvxfirgb." Atbash is an example of a substitution cipher because each character is replaced with another character. This type of substitution cipher is referred to as a monoalphabetic substitution cipher because it uses only one alphabet, whereas a polyalphabetic substitution cipher uses multiple alphabets.

TIP

Cipher is another term for algorithm.

Around 400 ▯.▯., the Spartans used a system of encrypting information in which they would write a message on a sheet of papyrus (a type of paper) that was wrapped around a staff (a stick or wooden rod), which was then delivered and wrapped around a different staff by the recipient. The message was only readable if it was wrapped around the correct size staff, which made the letters properly match up, as shown in Figure 8-1.

When the papyrus was not wrapped around the staff, the writing appeared as just a bunch of random characters. This approach, known as the scytale cipher, is an example of a transposition cipher because it relies on changing the sequence of the characters to obscure their meaning. Only someone who knows how to rearrange them would be able to recover the original message.

Later, in Rome, Julius Caesar (100–44 ▯.▯.) developed a simple method of shifting letters of the alphabet, similar to the atbash scheme. He simply shifted the alphabet by three positions. The following example shows a standard alphabet and a shifted alphabet. The alphabet serves as the algorithm, and the key is the number of locations it has been shifted during the encryption and decryption process.

• Standard alphabet:
ABCDEFGHIJKLMNOPQRSTUVWXYZ
• Cryptographic alphabet:
DEFGHIJKLMNOPQRSTUVWXYZABC

As an example, suppose we need to encrypt the message "MISSION ACCOMPLISHED." We take the first letter of this message, M, and shift up three locations within the alphabet. The encrypted version of this first letter is P, so we write

Figure 8-1
The scytale was used by the Spartans to decipher encrypted messages.

that down. The next letter to be encrypted is I, which matches L when we shift three spaces. We continue this process for the whole message. Once the message is encrypted, a carrier takes the encrypted version to the destination, where the process is reversed.

- Original message:
MISSION ACCOMPLISHED

- Encrypted message:
PLVVLRQ DFFRPSOLVKHG

Today, this technique seems too simplistic to be effective, but in the time of Julius Caesar, not very many people could read in the first place, so it provided a high level of protection. The Caesar cipher, like the atbash cipher, is an example of a monoalphabetic cipher. Once more people could read and reverse-engineer this type of encryption process, the cryptographers of that day increased the complexity by creating polyalphabetic ciphers.

In the 16th century in France, Blaise de Vigenère developed a polyalphabetic substitution cipher for Henry III. This was based on the Caesar cipher, but it increased the difficulty of the encryption and decryption process. As shown in Figure 8-2, we have a message that needs to be encrypted, which is SYSTEM SECURITY AND CONTROL. We have a key with the value of SECURITY. We also have a Vigenère table, or algorithm, which is really the Caesar cipher on steroids. Whereas the Caesar cipher used a single shift alphabet (letters were shifted up three places), the Vigenère cipher has 27 shift alphabets and the letters are shifted up only one place.

So, looking at the example in Figure 8-2, we take the first value of the key, S, and starting with the first alphabet in our algorithm, trace over to the S column. Then we look at the first character of the original message that needs to be encrypted, which is S, and go down to the S row. We follow the column and row and see that they intersect on the value K. That is the first encrypted value of our message, so we write down K.

Then we go to the next value in our key, which is E, and the next character in the original message, which is Y. We see that the E column and the Y row intersect at the cell with the value of C. This is our second encrypted value, so we write that down.

We continue this process for the whole message (notice that the key repeats itself, since the message is longer than the key). The result is an encrypted message that is sent to the destination. The destination must have the same algorithm (Vigenère table) and the same key (SECURITY) to properly reverse the process to obtain a meaningful message.

The evolution of cryptography continued as countries refined it using new methods, tools, and practices with varying degrees of success. Mary, Queen of Scots, lost her life in the 16th century when an encrypted message she sent was intercepted. During the American Revolutionary War, Benedict Arnold used a codebook cipher to exchange information on troop movement and strategic military advancements. By the late 1800s, cryptography was commonly used in the methods of communication between military factions.

During World War II, encryption devices were used for tactical communication, which drastically improved with the mechanical and electromechanical technology that provided the world with telegraphic and radio communication. The rotor cipher machine, which is a device that substitutes letters using different rotors within the

Vigenére Table

Repeated
key

SECURITY
SYSTEMSE
SECURITY
CURITYAN
SECURITY
DCONTROL

KCUNVULCUYTCKGTLVGQHKZHJ

Key: SECURITY
Original message:
SYSTEM SECURITY AND CONTROL
Encrypted message:
KCUNVULCUYTCKGTLVGQHKZHJ

Figure 8-2

Polyalphabetic algorithms were developed to increase encryption complexity.

machine, was a huge breakthrough in military cryptography that provided complexity that proved difficult to break. This work gave way to the most famous cipher machine in history to date: Germany's Enigma machine. The Enigma machine had separate rotors, a plug board, and a reflecting rotor.

The originator of the message would configure the Enigma machine to its initial settings before starting the encryption process. The operator would type in the first letter of the message, and the machine would substitute the letter with a different letter and present it to the operator. This encryption was done by moving the rotors a predefined number of times. So, if the operator typed in a T as the first character, the Enigma machine might present an M as the substitution value. The operator would write down the letter M on his sheet. The operator would then advance the rotors and enter the next letter. Each time a new letter was to be encrypted, the operator would advance the rotors to a new setting.

This process was followed until the whole message was encrypted. Then the encrypted text was transmitted over the airwaves, most likely to a German U-boat. The chosen

Cryptography Definitions and Concepts

Encryption is a method of transforming readable data, called plaintext, into a form that appears to be random and unreadable, which is called ciphertext. Plaintext is in a form that can be understood either by a person (a document) or by a computer (executable code). Once plaintext is transformed into ciphertext, neither human nor machine can properly process it until it is decrypted. This enables the transmission of confidential information over insecure channels without unauthorized disclosure. When sensitive data is stored on a computer, it is usually protected by logical and physical access controls. When this same sensitive information is sent over a network, it no longer has the advantage of these controls and is in a much more vulnerable state.

Plaintext

Encryption

Ciphertext

Decryption

Plaintext

A system or product that provides encryption and decryption is referred to as a cryptosystem and can be created through hardware components or program code in an application. The cryptosystem uses an encryption algorithm (which determines how simple or complex the encryption process will be), keys, and the necessary software components and protocols. Most algorithms are complex mathematical formulas that are applied in a specific sequence to the plaintext. Most encryption methods use a secret value called a key (usually a long string of bits), which works with the algorithm to encrypt and decrypt the text.
The algorithm, the set of rules also known as the cipher, dictates how enciphering and deciphering take place. Many of the mathematical algorithms used in computer systems today are publicly known and are not the secret part of the encryption process. If the internal mechanisms of the algorithm are not a secret, then something must be: the key.

PART III

substitution for each letter was dependent upon the rotor setting, so the crucial and secret part of this process (the key) was the initial setting and how the operators advanced the rotors when encrypting and decrypting a message. The operators at each end needed to know this sequence of increments to advance each rotor in order to enable the German military units to properly communicate.
When computers were invented, the possibilities for encryption methods and devices expanded exponentially and cryptography efforts increased dramatically. This era brought unprecedented opportunity for cryptographic designers to develop new encryption techniques. A well-known and successful project was Lucifer, which was developed at IBM.
Lucifer introduced complex mathematical equations and functions that were later adopted and modified by the U.S. National Security Agency (NSA) to establish the U.S.
Data Encryption Standard (DES) in 1976, a federal government standard. DES was used worldwide for financial and other transactions, and was embedded into numerous commercial applications. Though it was cracked in the late 1990s and is no longer

considered secure, DES represented a significant advancement for cryptography. It was replaced a few years later by the Advanced Encryption Standard (AES), which continues to protect sensitive data to this day.

A common analogy used to illustrate this point is the use of locks you would purchase from your local hardware store. Let’s say 20 people bought the same brand of lock. Just because these people share the same type and brand of lock does not mean they can now unlock each other’s doors and gain access to their private possessions. Instead, each lock comes with its own key, and that one key can open only that one specific lock.

In encryption, the key (also known as cryptovariable) is a value that comprises a large sequence of random bits. Is it just any random number of bits crammed together? Not really. An algorithm contains a keyspace, which is a range of values that can be used to construct a key. When the algorithm needs to generate a new key, it uses random values from this keyspace. The larger the keyspace, the more available values that can be used to represent different keys—and the more random the keys are, the harder it is for intruders to figure them out. For example, if an algorithm allows a key length of 2 bits, the keyspace for that algorithm would be 4, which indicates the total number of different keys that would be possible. (Remember that we are working in binary and that 22 equals 4.) That would not be a very large keyspace, and certainly it would not take an attacker very long to find the correct key that was used.

A large keyspace allows for more possible keys. (Today, we are commonly using key sizes of 128, 256, 512, or even 1,024 bits and larger.) So a key size of 512 bits would provide 2512 possible combinations (the keyspace). The encryption algorithm should use the entire keyspace and choose the values to make up the keys as randomly as possible. If a smaller keyspace were used, there would be fewer values to choose from when generating a key, as shown in Figure 8-3. This would increase an attacker’s chances of figuring out the key value and deciphering the protected information.

Keys

Keyspace

Keyspace

Keys

Figure 8-3 Larger keyspaces permit a greater number of possible key values.

Encrypted message
askfjaoiwenh220va8fjsdnv jaksfue92v8ssk

Intruder obtains the
message but its encryption
makes it useless to her.

askfjaoiwenh220va8fjsdnv jaksfue92v8ssk

Figure 8-4 Without the right key, the captured message is useless to an attacker.

If an eavesdropper captures a message as it passes between two people, she can view the message, but it appears in its encrypted form and is therefore unusable. Even if this attacker knows the algorithm that the two people are using to encrypt and decrypt their information, without the key, this information remains useless to the eavesdropper, as shown in Figure 8-4.

Cryptosystems

A cryptosystem encompasses all of the necessary components for encryption and decryption to take place. Pretty Good Privacy (PGP) is just one example of a cryptosystem.

A cryptosystem is made up of at least the following:

- Software
- Protocols
- Algorithms
- Keys

Cryptosystems can provide the following services:

- Confidentiality Renders the information unintelligible except by authorized entities.
- Integrity Ensures that data has not been altered in an unauthorized manner since it was created, transmitted, or stored.
- Authentication Verifies the identity of the user or system that created the information.

PART III

Intruder

• Authorization Provides access to some resource to the authenticated user or system.
• Nonrepudiation Ensures that the sender cannot deny sending the message.

As an example of how these services work, suppose your boss sends you an e-mail message stating that you will be receiving a raise that doubles your salary. The message is encrypted, so you can be sure it really came from your boss (authenticity), that someone did not alter it before it arrived at your computer (integrity), that no one else was able to read it as it traveled over the network (confidentiality), and that your boss cannot deny sending it later when he comes to his senses (nonrepudiation).

Different types of messages and transactions require higher or lower degrees of one or all of the services that cryptography methods can supply. Military and intelligence agencies are very concerned about keeping information confidential, so they would choose encryption mechanisms that provide a high degree of secrecy. Financial institutions care about confidentiality, but they also care about the integrity of the data being transmitted, so the encryption mechanism they would choose may differ from the military's encryption methods. If messages were accepted that had a misplaced decimal point or zero, the ramifications could be far reaching in the financial world.

Legal agencies may care most about the authenticity of the messages they receive. If information received ever needed to be presented in a court of law, its authenticity would certainly be questioned; therefore, the encryption method used must ensure authenticity, which confirms who sent the information.

NOTE If David sends a message and then later claims he did not send it, this is an act of repudiation. When a cryptography mechanism provides nonrepudiation, the sender cannot later deny he sent the message (well, he can try to deny it, but the cryptosystem proves otherwise).

The types and uses of cryptography have increased over the years. At one time, cryptography was mainly used to keep secrets secret (confidentiality), but today we use cryptography to ensure the integrity of data, to authenticate messages, to confirm that a message was received, to provide access control, and much more.

Kerckhoffs' Principle

Auguste Kerckhoffs published a paper in 1883 stating that the only secrecy involved with a cryptography system should be the key. He claimed that the algorithm should be publicly known. He asserted that if security were based on too many secrets, there would be more vulnerabilities to possibly exploit.

So, why do we care what some guy said almost 140 years ago? Because this debate is still going on. Cryptographers in certain sectors agree with Kerckhoffs' principle, because making an algorithm publicly available means that many more people can view the source code, test it, and uncover any type of flaws or weaknesses. It is the attitude of "many heads are better than one." Once someone uncovers some type of flaw, the developer can fix the issue and provide society with a much stronger algorithm.

But not everyone agrees with this philosophy. Governments around the world create their own algorithms that are not released to the public. Their stance is that if a smaller number of people know how the algorithm actually works, then a smaller number of people will know how to possibly break it. Cryptographers in the private sector do not agree with this practice and do not commonly trust algorithms they cannot examine. It is basically the same as the open-source versus compiled software debate that is in full force today.

The Strength of the Cryptosystem

One-Time Pad

A one-time pad is a perfect encryption scheme because it is considered unbreakable if implemented properly. It was invented by Gilbert Vernam in 1917, so sometimes it is referred to as the Vernam cipher.

This cipher does not use shift alphabets, as do the Caesar and Vigenère ciphers discussed earlier, but instead uses a pad made up of random values, as shown in Figure 8-5. Our plaintext message that needs to be encrypted has been converted into bits, and our one-time

The strength of an encryption method comes from the algorithm, the secrecy of the key, the length of the key, and how they all work together within the cryptosystem.

When
strength is discussed in encryption, it refers to how hard it is to figure out the algorithm
or key, whichever is not made public. Attempts to break a cryptosystem usually involve
processing an amazing number of possible values in the hopes of finding the one value
(key) that can be used to decrypt a specific message. The strength of an encryption
method correlates to the amount of necessary processing power, resources, and time
required to break the cryptosystem or to figure out the value of the key.
Breaking a cryptosystem can be accomplished by a brute-force attack, which means trying
every possible key value until the resulting plaintext is meaningful. Depending on the
algorithm and length of the key, this can be an easy task or one that is close to impossible. If
a key can be broken with an Intel Core i5 processor in three hours, the cipher is not strong
at all. If the key can only be broken with the use of a thousand multiprocessing systems over
1.2 million years, then it is pretty darned strong. The introduction of commodity cloud
computing has really increased the threat of brute-force attacks.
The goal when designing an encryption method is to make compromising it too
expensive or too time consuming. Another name for cryptography strength is work
factor, which is an estimate of the effort and resources it would take an attacker to
penetrate a cryptosystem.
Even if the algorithm is very complex and thorough, other issues within encryption
can weaken encryption methods. Because the key is usually the secret value needed to
actually encrypt and decrypt messages, improper protection of the key can weaken the
encryption. Even if a user employs an algorithm that has all the requirements for strong
encryption, including a large keyspace and a large and random key value, if she shares her
key with others, the strength of the algorithm becomes almost irrelevant.
Important elements of encryption are to use an algorithm without flaws, use a large key
size, use all possible values within the keyspace selected as randomly as possible, and protect
the actual key. If one element is weak, it could be the link that dooms the whole process.

Hello Mom,
I’ve dropped out of

school and decided
to travel. Please
send money.

One-time pad

Message

Ciphertext

Hello Mom,
I’ve dropped out of
school and decided
to travel. Please
send money.

Ciphertext

One-time pad

Message

Figure 8-5 A one-time pad

pad is made up of random bits. This encryption process uses a binary mathematic function
called exclusive-OR, usually abbreviated as XOR.
XOR is an operation that is applied to 2 bits and is a function commonly used in binary mathematics and encryption methods. When combining the bits, if both values
are the same, the result is 0 (1 XOR 1 = 0). If the bits are different from each other, the
result is 1 (1 XOR 0 = 1). For example:
Message stream:
1
Keystream:
0
Ciphertext stream: 1

0
0
0

0
1
1

1
1
0

0
1
1

1
0
1

0
1
1

1
0
1

1
1
0

1
0
1

So in our example, the first bit of the message is XORed to the first bit of the one-time
pad, which results in the ciphertext value 1. The second bit of the message is XORed with

the second bit of the pad, which results in the value 0. This process continues until the
whole message is encrypted. The result is the encrypted message that is sent to the receiver.

In Figure 8-5, we also see that the receiver must have the same one-time pad to decrypt
the message by reversing the process. The receiver takes the first bit of the encrypted
message and XORs it with the first bit of the pad. This results in the plaintext value. The
receiver continues this process for the whole encrypted message until the entire message
is decrypted.

The one-time pad encryption scheme is deemed unbreakable only if the following
things are true about the implementation process:

NOTE Generating truly random numbers is very difficult. Most systems use an algorithmic pseudorandom number generator (PRNG) that takes as its input a seed value and creates a stream of pseudorandom values from it. Given the same seed, a PRNG generates the same sequence of values. Truly random numbers must be based on natural phenomena such as thermal noise and quantum mechanics.

Although the one-time pad approach to encryption can provide a very high degree of

security, it is impractical in most situations because of all of its different requirements.
Each possible pair of entities that might want to communicate in this fashion must
receive, in a secure fashion, a pad that is as long as, or longer than, the actual message.
This type of key management can be overwhelming and may require more overhead than
it is worth. The distribution of the pad can be challenging, and the sender and receiver
must be perfectly synchronized so each is using the same pad.
EXAM TIP The one-time pad, though impractical for most modern
applications, is the only perfect cryptosystem.

• The pad must be used only one time. If the pad is used more than one time, this
might introduce patterns in the encryption process that will aid the eavesdropper
in his goal of breaking the encryption.
• The pad must be at least as long as the message. If it is not as long as the message,
the pad will need to be reused to cover the whole message. This would be the
same thing as using a pad more than one time, which could introduce patterns.
• The pad must be securely distributed and protected at its destination. This is a very
cumbersome process to accomplish, because the pads are usually just individual
pieces of paper that need to be delivered by a secure courier and properly guarded
at each destination.
• The pad must be made up of truly random values. This may not seem like a difficult
task, but even our computer systems today do not have truly random number
generators; rather, they have pseudorandom number generators.

One-Time Pad Requirements
For a one-time pad encryption scheme to be considered unbreakable, each pad in the scheme must be

• Made up of truly random values
• Used only one time
• Securely distributed to its destination
• Secured at sender's and receiver's sites
• At least as long as the message

Cryptographic Life Cycle
Since most of us will probably not be using one-time pads (the only "perfect" system)
to defend our networks, we have to consider that cryptography, like a fine steak, has a

limited shelf life. Given enough time and resources, any cryptosystem can be broken, either through analysis or brute force. The cryptographic life cycle is the ongoing process of identifying your cryptography needs, selecting the right algorithms, provisioning the needed capabilities and services, and managing keys. Eventually, you determine that your cryptosystem is approaching the end of its shelf life and you start the cycle all over again.

How can you tell when your algorithms (or choice of keyspaces) are about to go stale? You need to stay up to date with the cryptologic research community. They are the best source for early warning that things are going sour. Typically, research papers postulating weaknesses in an algorithm are followed by academic exercises in breaking the algorithm under controlled conditions, which are then followed by articles on how it is broken in general cases. When the first papers come out, it is time to start looking for replacements.

Cryptographic Methods

By far, the most commonly used cryptographic methods today are symmetric key cryptography, which uses symmetric keys (also called secret keys), and asymmetric key cryptography, which uses two different, or asymmetric, keys (also called public and private keys).

Asymmetric key cryptography is also called public key cryptography because one of its keys can be made public. As we will see shortly, public key cryptography typically uses powers of prime numbers for encryption and decryption. A variant of this approach uses elliptic curves, which allows much smaller keys to be just as secure and is (unsurprisingly) called elliptic curve cryptography (ECC). Though you may not know it, it is likely that you've used ECC at some point to communicate securely on the Web. (More on that later.)

Though these three cryptographic methods are considered secure today (given that you use good keys), the application of quantum computing to cryptology could dramatically change this situation. The following sections explain the key points of these four methods of encryption.

Symmetric Key Cryptography

$N(N - 1)/2$ = number of keys

The security of the symmetric encryption method is completely dependent on how

well users protect their shared keys. This should raise red flags for you if you have ever
had to depend on a whole staff of people to keep a secret. If a key is compromised, then
all messages encrypted with that key can be decrypted and read by an intruder. This
is complicated further by how symmetric keys are actually shared and updated when
necessary. If Dan wants to communicate with Norm for the first time, Dan has to figure
out how to get the right key to Norm securely. It is not safe to just send it in an e-mail
Figure 8-6
When using
symmetric
algorithms,
the sender and
receiver use
the same key
for encryption
and decryption
functions.

Symmetric encryption uses the same keys.

Encrypt
message
Message

Decrypt
message
Message

Message

PART III

In a cryptosystem that uses symmetric key cryptography, the sender and receiver use two
instances of the same key for encryption and decryption, as shown in Figure 8-6. So the
key has dual functionality in that it can carry out both encryption and
decryption processes. Symmetric keys are also called secret keys, because this type of encryption relies on
each user to keep the key a secret and properly protected. If an intruder were to get this
key, he could decrypt any intercepted message encrypted with it.
Each pair of users who want to exchange data using symmetric key encryption must
have two instances of the same key. This means that if Dan and Iqqi want to communicate,
both need to obtain a copy of the same key. If Dan also wants to communicate using
symmetric encryption with Norm and Dave, he needs to have three separate keys, one

for each friend. This might not sound like a big deal until Dan realizes that he may
communicate with hundreds of people over a period of several months, and keeping track and using the correct key that corresponds to each specific receiver can become
a daunting task. If 10 people needed to communicate securely with each other using
symmetric keys, then 45 keys would need to be kept track of. If 100 people were going
to communicate, then 4,950 keys would be involved. The equation used to calculate the
number of symmetric keys needed is

Symmetric Key Cryptosystems Summary

The following outlines the strengths and weaknesses of symmetric key algorithms.

Strengths:

• Much faster (less computationally intensive) than asymmetric systems.
• Hard to break if using a large key size.

Weaknesses:

• Requires a secure mechanism to deliver keys properly.
• Each pair of users needs a unique key, so as the number of individuals increases, so does the number of keys, possibly making key management overwhelming.
• Provides confidentiality but not authenticity or nonrepudiation.

Examples:

• Advanced Encryption Standard (AES)
• ChaCha20

message, because the key is not protected and can be easily intercepted and used by
attackers. Thus, Dan must get the key to Norm through an out-of-band method. Dan can
save the key on a thumb drive and walk over to Norm's desk, or have a secure courier
deliver it to Norm. This is a huge hassle, and each method is very clumsy and insecure.

Because both users employ the same key to encrypt and decrypt messages, symmetric
cryptosystems can provide confidentiality, but they cannot provide authentication or
nonrepudiation. There is no way to prove through cryptography who actually sent a
message if two people are using the same key.

If symmetric cryptosystems have so many problems and flaws, why use them at all?
Because they are very fast and can be hard to break. Compared with asymmetric systems,
symmetric algorithms scream in speed. They can encrypt and decrypt relatively quickly
large amounts of data that would take an unacceptable amount of time to encrypt

and
decrypt with an asymmetric algorithm. It is also difficult to uncover data encrypted
with a symmetric algorithm if a large key size is used. For many of our applications that
require encryption, symmetric key cryptography is the only option.
The two main types of symmetric algorithms are block ciphers, which work on blocks
of bits, and stream ciphers, which work on one bit at a time.

Block Ciphers
When a block cipher is used for encryption and decryption purposes, the message is
divided into blocks of bits. These blocks are then put through mathematical functions,
one block at a time. Suppose you need to encrypt a message you are sending to your

mother and you are using a block cipher that uses 64 bits. Your message of 640 bits is
chopped up into 10 individual blocks of 64 bits. Each block is put through a succession
of mathematical formulas, and what you end up with is 10 blocks of encrypted text.

Message

110011 110101
001011 111010
111100 110101
110101 101000

Second block
of plaintext

Third block
of plaintext

100101
110101

100101
100101

101000
101010

Encryption

Encryption

Encryption

010011
101010

010101
101100

101010
001011

First block
of ciphertext

Second block
of ciphertext

Third block
of ciphertext

001010 011010
101000 110101
Message

You send this encrypted message to your mother. She has to have the same block cipher and key, and those 10 ciphertext blocks go back through the algorithm in the reverse sequence and end up in your plaintext message.
A strong cipher contains the right level of two main attributes: confusion and diffusion.
Confusion is commonly carried out through substitution, while diffusion is carried out by using transposition. For a cipher to be considered strong, it must contain both of these attributes to ensure that reverse-engineering is basically impossible. The randomness of the key values and the complexity of the mathematical functions dictate the level of confusion and diffusion involved.
In algorithms, diffusion takes place as individual bits of a block are scrambled, or diffused, throughout that block. Confusion is provided by carrying out complex substitution functions so the eavesdropper cannot figure out how to substitute the right values and come up with the original plaintext. Suppose you have 500 wooden blocks with individual letters written on them. You line them all up to spell out a paragraph (plaintext). Then you substitute 300 of them with another set of 300 blocks (confusion through substitution). Then you scramble all of these blocks (diffusion through transposition) and leave them in a pile. For someone else to figure out your original message, they would have to substitute the correct blocks and then put them back

in the
right order. Good luck.
Confusion pertains to making the relationship between the key and resulting
ciphertext as complex as possible so the key cannot be uncovered from the
ciphertext.
Each ciphertext value should depend upon several parts of the key, but this
mapping
between the key values and the ciphertext values should seem completely random
to
the observer.
Diffusion, on the other hand, means that a single plaintext bit has influence
over
several of the ciphertext bits. Changing a plaintext value should change many
ciphertext

Did you know
that Dave dropped
out of college and
joined the circus?
He asked his mom
for money to buy
a tiger, but she
only sent enough
to buy the stripes!

First block
of plaintext

values, not just one. In fact, in a strong block cipher, if one plaintext bit is
changed, it
will change every ciphertext bit with the probability of 50 percent. This means
that if one
plaintext bit changes, then about half of the ciphertext bits will change.
A very similar concept of diffusion is the avalanche effect. If an algorithm
follows
strict avalanche effect criteria, this means that if the input to an algorithm
is slightly
modified, then the output of the algorithm is changed significantly. So a small
change to
the key or the plaintext should cause drastic changes to the resulting
ciphertext. The ideas
of diffusion and avalanche effect are basically the same—they were just derived
from
different people. Horst Feistel came up with the avalanche term, while Claude
Shannon
came up with the diffusion term. If an algorithm does not exhibit the necessary
degree
of the avalanche effect, then the algorithm is using poor randomization. This
can make

it easier for an attacker to break the algorithm.
Block ciphers use diffusion and confusion in their methods. Figure 8-7 shows a conceptual example of a simplistic block cipher. It has four block inputs, and each block
is made up of 4 bits. The block algorithm has two layers of 4-bit substitution boxes called
Message (plaintext)—YX

1 0 1 1
Key determines
which S-boxes
are used
and how.

1 0 0 1

1 0 1 1

0 0 0 1

S-box

S-box

S-box

S-box

S-box

S-box

S-box

S-box

0 0 0 1

0 1 1 1

0 0 0 1

1 1 0 0

Lookup table
1. XOR bit with
1 then 0
2. XOR result
with 0,1,1
3. XOR result
with 1,0
4. XOR result
with 0,0

Encrypted message (ciphertext)—B9

Figure 8-7 A message is divided into blocks of bits, and substitution and transposition functions
are performed on those blocks.

S-boxes. Each S-box contains a lookup table used by the algorithm as instructions on how
the bits should be encrypted.
Figure 8-7 shows that the key dictates what S-boxes are to be used when scrambling
the original message from readable plaintext to encrypted nonreadable ciphertext. Each
S-box contains the different substitution methods that can be performed on each block.
This example is simplistic—most block ciphers work with blocks of 32, 64, or 128 bits
in size, and many more S-boxes are usually involved.

Stream Ciphers

NOTE This process is very similar to the one-time pad explained earlier. The
individual bits in the one-time pad are used to encrypt the individual bits
of the message through the XOR function, and in a stream algorithm the
individual bits created by the keystream generator are used to encrypt the
bits of the message through XOR also.

In block ciphers, it is the key that determines what functions are applied to the plaintext
and in what order. The key provides the randomness of the encryption process. As stated
earlier, most encryption algorithms are public, so people know how they work. The secret
to the secret sauce is the key. In stream ciphers, the key also provides randomness, so that
the stream of bits that is XORed to the plaintext is as random as possible. This concept

Figure 8-8
With stream
ciphers, the bits
generated by
the keystream
generator are
XORed with
the bits of
the plaintext
message.

Keystream
generator

1
0
1
1
0
1
0
0
Plaintext message

XOR
Ciphertext message

PART III

As stated earlier, a block cipher performs mathematical functions on blocks of bits. A stream cipher, on the other hand, does not divide a message into blocks. Instead, a stream cipher treats the message as a stream of bits and performs mathematical functions on each bit individually.
When using a stream cipher, a plaintext bit will be transformed into a different ciphertext bit each time it is encrypted. Stream ciphers use keystream generators, which produce a stream of bits that is XORed with the plaintext bits to produce ciphertext, as shown in Figure 8-8.

Key

Keystream
generator

Keystream
generator

Keystream
Plaintext

Keystream
Ciphertext

Encrypt

Key

Plaintext
Decrypt

Figure 8-9 The sender and receiver must have the same key to generate the same keystream.

is shown in Figure 8-9. As you can see in this graphic, both the sending and receiving ends must have the same key to generate the same keystream for proper encryption and decryption purposes.

Initialization Vectors

Initialization vectors (IVs) are random values that are used with algorithms to ensure patterns are not created during the encryption process. They are used with keys and do not need to be encrypted when being sent to the destination. If IVs are not used, then two identical plaintext values that are encrypted with the same key will create the same ciphertext. Providing attackers with these types of patterns can make their job easier in breaking the encryption method and uncovering the key. For example, if we have the plaintext value of “See Spot run” two times within our message, we need to make sure that even though there is a pattern in the plaintext message, a pattern in the resulting ciphertext will not be created. So the IV and key are both used by the algorithm to provide more randomness to the encryption process.

A strong and effective stream cipher contains the following characteristics:

• Easy to implement in hardware Complexity in the hardware design makes it more difficult to verify the correctness of the implementation and can slow it down.
• Long periods of no repeating patterns within keystream values Bits generated by the keystream are not truly random in most cases, which will eventually lead to the emergence of patterns; we want these patterns to be rare.
• A keystream not linearly related to the key If someone figures out the keystream values, that does not mean she now knows the key value.
• Statistically unbiased keystream (as many zeroes as ones) There should be no dominance in the number of zeroes or ones in the keystream.

Stream ciphers require a lot of randomness and encrypt individual bits at a time. This requires more processing power than block ciphers require, which is why stream ciphers

are better suited to be implemented at the hardware level. Because block ciphers do not require as much processing power, they can be easily implemented at the software level.

Asymmetric Key Cryptography

Asymmetric systems use two different keys
for encryption and decryption purposes.

Figure 8-10
An asymmetric
cryptosystem

Public
key

Private
key
Encrypt
message

Message

Decrypt message
with different key

Message

PART III

In symmetric key cryptography, a single secret key is used between entities, whereas in public key systems, each entity has different, asymmetric keys. The two different asymmetric keys are mathematically related. If a message is encrypted by one key, the other key is required in order to decrypt the message. One key is called public and the other one private. The public key can be known to everyone, and the private key must be known and used only by the owner. Many times, public keys are listed in directories and databases of e-mail addresses so they are available to anyone who wants to use these keys to encrypt or decrypt data when communicating with a particular person. Figure 8-10 illustrates the use of the different keys.

The public and private keys of an asymmetric cryptosystem are mathematically related, but if someone gets another person's public key, she should not be able to figure out the corresponding private key. This means that if an eavesdropper gets a copy of Bob's public key, she can't employ some mathematical magic and find out Bob's private key. But if someone gets Bob's private key, then there is big trouble—no one other than the owner should have access to a private key.

If Bob encrypts data with his private key, the receiver must have a copy of Bob's
public key to decrypt it. The receiver can decrypt Bob's message and decide to reply to
Bob in an encrypted form. All the receiver needs to do is encrypt her reply with Bob's
public key, and then Bob can decrypt the message with his private key. It is not possible
to encrypt and decrypt using the same key when using an asymmetric key encryption
technology because, although mathematically related, the two keys are not the same key,
as they are in symmetric cryptography. Bob can encrypt data with his private key, and the
receiver can then decrypt it with Bob's public key. By decrypting the message with Bob's

public key, the receiver can be sure the message really came from Bob. A message can be
decrypted with a public key only if the message was encrypted with the corresponding
private key. This provides authentication, because Bob is the only one who is supposed
to have his private key. However, it does not truly provide confidentiality because anyone
with the public key (which is, after all, public) can decrypt it. If the receiver wants to
make sure Bob is the only one who can read her reply, she will encrypt the response with
his public key. Only Bob will be able to decrypt the message because he is the only one
who has the necessary private key.

The receiver can also choose to encrypt data with her private key instead of using
Bob's public key. Why would she do that? Authentication—she wants Bob to know that
the message came from her and no one else. If she encrypted the data with Bob's public
key, it does not provide authenticity because anyone can get Bob's public key. If she uses
her private key to encrypt the data, then Bob can be sure the message came from her and
no one else. Symmetric keys do not provide authenticity, because the same key is used
on both ends. Using one of the secret keys does not ensure the message originated from
a specific individual.

If confidentiality is the most important security service to a sender, she would encrypt
the file with the receiver's public key. This is called a secure message format because it can

only be decrypted by the person who has the corresponding private key.
If authentication is the most important security service to the sender, then she would
encrypt the data with her private key. This provides assurance to the receiver that the only
person who could have encrypted the data is the individual who has possession of that
private key. If the sender encrypted the data with the receiver's public key, authentication
is not provided because this public key is available to anyone.
Encrypting data with the sender's private key is called an open message format
because anyone with a copy of the corresponding public key can decrypt the message.
Confidentiality is not ensured.
Each key type can be used to encrypt and decrypt, so do not get confused and think
the public key is only for encryption and the private key is only for decryption. They
both have the capability to encrypt and decrypt data. However, if data is encrypted with
a private key, it cannot be decrypted with a private key. If data is encrypted with a private
key, it must be decrypted with the corresponding public key.
An asymmetric algorithm works much more slowly than a symmetric algorithm,
because symmetric algorithms carry out relatively simplistic mathematical functions on
the bits during the encryption and decryption processes. They substitute and scramble
(transposition) bits, which is not overly difficult or processor intensive. The reason it is
hard to break this type of encryption is that the symmetric algorithms carry out this type
of functionality over and over again. So a set of bits will go through a long series of being
substituted and scrambled.
Asymmetric algorithms are slower than symmetric algorithms because they use much
more complex mathematics to carry out their functions, which requires more processing
time. Although they are slower, asymmetric algorithms can provide authentication and
nonrepudiation, depending on the type of algorithm being used. Asymmetric systems
also provide for easier and more manageable key distribution than symmetric systems and
do not have the scalability issues of symmetric systems. The reason for these differences

Asymmetric Key Cryptosystems Summary
The following outlines the strengths and weaknesses of asymmetric key algorithms.
Strengths:

• Better key distribution than symmetric systems.
• Better scalability than symmetric systems.
• Can provide authentication and nonrepudiation.

Weaknesses:

Examples:

• Rivest-Shamir-Adleman (RSA)
• Elliptic curve cryptography (ECC)
• Digital Signature Algorithm (DSA)

is that, with asymmetric systems, you can send out your public key to all of the people you need to communicate with, instead of keeping track of a unique key for each one of them. The “Hybrid Encryption Methods” section later in this chapter shows how these two systems can be used together to get the best of both worlds.

TIP Public key cryptography is asymmetric cryptography. The terms can be used interchangeably.

Table 8-1 summarizes the differences between symmetric and asymmetric algorithms.

Diffie-Hellman Algorithm

The first group to address the shortfalls of symmetric key cryptography decided to attack the issue of secure distribution of the symmetric key. Whitfield Diffie and Martin Hellman worked on this problem and ended up developing the first asymmetric key agreement algorithm, called, naturally, Diffie-Hellman.

To understand how Diffie-Hellman works, consider an example. Let’s say that Tanya and Erika would like to communicate over an encrypted channel by using Diffie-Hellman. They would both generate a private and public key pair and exchange public keys. Tanya’s software would take her private key (which is just a numeric value) and Erika’s public key (another numeric value) and put them through the Diffie-Hellman algorithm. Erika’s software would take her private key and Tanya’s public key and insert them into the Diffie-Hellman algorithm on her computer. Through this process, Tanya and Erika derive the same shared value, which is used to create instances of symmetric keys.

• Works much more slowly than symmetric systems.
• Mathematically intensive tasks.

Attribute	Symmetric	Asymmetric
Keys	One key is shared between two or more entities.	One entity has a public key, and the other entity has the corresponding private key.
Key exchange	Out-of-band through secure mechanisms.	A public key is made available to everyone, and a private key is kept secret by the owner.
Speed	The algorithm is less complex and faster.	The algorithm is more complex and slower.
Use	Bulk encryption, which means encrypting files and communication paths.	Key distribution and digital signatures.
Security service provided	Confidentiality.	Confidentiality, authentication, and nonrepudiation.

Table 8-1 Differences Between Symmetric and Asymmetric Systems

So, Tanya and Erika exchanged information that did not need to be protected (their public keys) over an untrusted network, and in turn generated the exact same

symmetric key on each system. They both can now use these symmetric keys to
encrypt, transmit, and decrypt information as they communicate with each other.
NOTE The preceding example describes key agreement, which is different from key exchange, the functionality used by the other asymmetric algorithms that will be discussed in this chapter. With key exchange functionality, the sender encrypts the symmetric key with the receiver's public key before transmission.

The Diffie-Hellman algorithm enables two systems to generate a symmetric key securely without requiring a previous relationship or prior arrangements. The algorithm allows for key distribution, but does not provide encryption or digital signature functionality. The algorithm is based on the difficulty of calculating discrete logarithms in a finite field.

The original Diffie-Hellman algorithm is vulnerable to a man-in-the-middle attack, because no authentication occurs before public keys are exchanged. In our example, when Tanya sends her public key to Erika, how does Erika really know it is Tanya's public key? What if Lance spoofed his identity, told Erika he was Tanya, and sent over his key? Erika would accept this key, thinking it came from Tanya. Let's walk through the steps of how this type of attack would take place, as illustrated in Figure 8-11:

1. Tanya sends her public key to Erika, but Lance grabs the key during transmission so it never makes it to Erika.
2. Lance spoofs Tanya's identity and sends over his public key to Erika. Erika now thinks she has Tanya's public key.

Figure 8-11
A man-in-themiddle attack
against a
Diffie-Hellman
key agreement

S1

S2

Tanya

Lance
S1

S2

3. Erika sends her public key to Tanya, but Lance grabs the key during transmission
so it never makes it to Tanya.
4. Lance spoofs Erika's identity and sends over his public key to Tanya. Tanya now
thinks she has Erika's public key.
5. Tanya combines her private key and Lance's public key and creates symmetric key S1.
6. Lance combines his private key and Tanya's public key and creates symmetric key S1.
7. Erika combines her private key and Lance's public key and creates symmetric key S2.
8. Lance combines his private key and Erika's public key and creates symmetric key S2.
9. Now Tanya and Lance share a symmetric key (S1) and Erika and Lance share
a different symmetric key (S2). Tanya and Erika think they are sharing a key
between themselves and do not realize Lance is involved.
10. Tanya writes a message to Erika, uses her symmetric key (S1) to encrypt the
message, and sends it.
11. Lance grabs the message and decrypts it with symmetric key S1, reads or modifies
the message and re-encrypts it with symmetric key S2, and then sends it to Erika.
12. Erika takes symmetric key S2 and uses it to decrypt and read the message.

The countermeasure to this type of attack is to have authentication take place before
accepting someone's public key. The basic idea is that we use some sort of certificate to
attest the identity of the party on the other side before trusting the data we receive from
it. One of the most common ways to do this authentication is through the use of the RSA
cryptosystem, which we describe next.

RSA
RSA, named after its inventors Ron Rivest, Adi Shamir, and Leonard Adleman, is a
public key algorithm that is the most popular when it comes to asymmetric algorithms. RSA
is a worldwide de facto standard and can be used for digital signatures, key exchange,
and encryption. It was developed in 1978 at MIT and provides authentication as well as
key encryption.
The security of this algorithm comes from the difficulty of factoring large numbers
into their original prime numbers. The public and private keys are functions of a pair
of large prime numbers, and the necessary activity required to decrypt a message

from
ciphertext to plaintext using a private key is comparable to factoring a product into two
prime numbers.
NOTE A prime number is a positive whole number whose only factors
(i.e., integer divisors) are 1 and the number itself.

One advantage of using RSA is that it can be used for encryption and digital signatures.
Using its one-way function, RSA provides encryption and signature verification, and the
inverse direction performs decryption and signature generation.
RSA has been implemented in applications; in operating systems; and at the hardware
level in network interface cards, secure telephones, and smart cards. RSA can be used as a
key exchange protocol, meaning it is used to encrypt the symmetric key to get it securely to
its destination. RSA has been most commonly used with the symmetric algorithm AES.
So, when RSA is used as a key exchange protocol, a cryptosystem generates a symmetric
key to be used with the AES algorithm. Then the system encrypts the symmetric key
with the receiver's public key and sends it to the receiver. The symmetric key is protected
because only the individual with the corresponding private key can decrypt and extract
the symmetric key.
Diving into Numbers Cryptography is really all about using mathematics to scramble
bits into an undecipherable form and then using the same mathematics in reverse to put the
bits back into a form that can be understood by computers and people. RSA's mathematics
are based on the difficulty of factoring a large integer into its two prime factors. Put on your
nerdy hat with the propeller and let's look at how this algorithm works.
The algorithm creates a public key and a private key from a function of large prime
numbers. When data is encrypted with a public key, only the corresponding private key
can decrypt the data. This act of decryption is basically the same as factoring the product
of two prime numbers. So, let's say Ken has a secret (encrypted message), and for you to

be able to uncover the secret, you have to take a specific large number and factor it and
come up with the two numbers Ken has written down on a piece of paper. This may
sound simplistic, but the number you must properly factor can be 22048 in size.