Cryptographic controls: The heart of cyber security

Abstract: Cryptography lies at the heart of all cyber security technologies and appreciating the role it plays is vital to understanding how to secure cyberspace. In this chapter, we review the basic tools provided by cryptography, and how they are used to provide the core security services of confidentiality, data integrity and entity authentication in cyberspace.

Introduction

Cryptography provides a set of mathematical tools for providing the fundamental security required in cyberspace. From these basic tools, we are able to construct more complex security protocols, such as secure messaging, secure access to web sites, secure payment systems, etc. Without cryptography, none of this would be remotely conceivable.

To appreciate what cryptography does, it is worth considering how information is secured in the physical world. This is done through a combination of physical mechanisms, which include protection of an object containing information (locking a document in a safe), modification detection techniques (watermarks on a bank note) and physical identification (face and voice recognition). Security is also supported by the context in which things happen in the physical world (to break into an office you need to physically get to the office and circumvent any access controls). In cyberspace we lose both the ability to deploy physical security mechanisms and the physical context. In theory, a computer can be broken into by anyone, anywhere, who is connected to the same network.

In order to provide security in cyberspace we thus need very different mechanisms from which notions of security can be constructed. Cryptography provides these mechanisms. Each mechanism implements a specific *security service*. The most important of these services are:

- *Confidentiality*, which is the assurance that data cannot be viewed by an unauthorized entity. This is sometimes known as *secrecy*, and allows the control of who can see what information in cyberspace.
- *Data integrity*, which is the assurance that data has not been altered in an unauthorized manner since the point at which it was created. Cryptography provides a range of different tools for this, depending on how strong an assurance is required.
- *Entity authentication,* which is the assurance that a given entity is involved and active in a communication session in cyberspace. This provides answers to the question ``who is out there?'' in cyberspace.

We first explain the core principles behind cryptography, then review the most common cryptographic tools for implementing these security services. We also discuss the wider system within which these tools need to be deployed in order to provide cyber security.

Fundamentals of cryptography

Cryptographic fundamentals are best illustrated by considering the features of a physical lock.

Firstly, the mechanical methods behind the workings of a physical lock tend to be well-known. This allows everyone to understand whether a physical lock is of good quality or not, as well as to develop a notion of what it would take to defeat the lock. The cryptographic equivalent of this is a *cryptographic algorithm*, which provides a specification of how a particular cryptographic tool works. Most cryptographic algorithms are published and developers can use this specification to implement them. Everyone can inspect a cryptographic algorithm and then reason about its security.

To provide security, however, there must be something that differentiates one instantiation of a security technology from another. For a physical lock, this differentiator is the key. Front doors might all have the same type of lock fitted, which all work in the same way, but any specific lock requires a unique key to open it. So, too, does a cryptographic algorithm typically involve a key. A *cryptographic key* is a (normally) secret piece of data which serves as an input into the cryptographic algorithm. Inputting two different cryptographic keys into the same cryptographic algorithm results in a different output, thus allowing cryptographic keys to be used to provide entities in cyberspace with unique capabilities that others do not have.

To break a physical lock, an attacker needs to find a key with the right settings to push down the tumblers that will free the bolt. A good physical lock will have so many possible settings that this task is difficult. The same is true for a cryptographic key. However, computers are much faster at conducting computations than burglars of a physical lock, so the number of possible cryptographic keys that could be input into a cryptographic algorithm needs to be massive. As an illustration, many of the cryptographic tools we use today have around 2^{128} possible keys, which is so big that the world's fastest supercomputer would take billions of years to try them all out.

Confidentiality tools

Possibly the most well-known use of cryptography is the provision of confidentiality, through a process often referred to as *encryption*. The basic model of an encryption mechanism is shown in Figure 1.

The goal of encryption is to render some data, the *plaintext*, only readable to anyone who has been given the capability to do so. The data is thus converted into an unreadable form, known as the *ciphertext*, by feeding it through an *encryption algorithm*. The encryption algorithm takes as input the plaintext and an *encryption key*. The resulting ciphertext can now be passed over an insecure channel, such as the Internet, since it appears to be random data. Only someone with the corresponding *decryption key* can reconvert the ciphertext back into plaintext by feeding the ciphertext and the matching *decryption key* into a *decryption algorithm*.

The encryption and decryption algorithms are essentially the ``recipes'' for scrambling and unscrambling the data, thus are closely related to one another. In most cases these algorithms

are open standards, as is the case for the *Advanced Encryption Standard* (AES), which is the most widely deployed encryption algorithm. The critical secret, knowledge of which distinguishes a desired recipient of a confidential message from everyone else, is the decryption key. This value must be carefully protected. In modern technologies, decryption keys are usually securely stored in hardware such as smartcards, tokens, key fobs, or sometimes generated from passwords or passphrases.

There is, however, no inherent reason why an encryption key needs to be a secret value. Indeed, whether it is secret or not is the distinguishing feature of two major classes of encryption mechanism:

- In *symmetric* encryption, the encryption and decryption keys are the same value, which means the encryption key also needs to be kept secret. Symmetric encryption emulates the type of physical locking mechanism that requires the same key to both lock and unlock.
- In *public-key* encryption, the encryption key can be made publicly available. Public-key encryption emulates the likes of a padlock, where anyone can snap it shut but only the key holder can unlock.

These two types of encryption mechanism have very different features. Symmetric encryption requires the secret key to be distributed in advance to both the sender of a confidential message and the intended recipient (this is known as the *key distribution problem*). Public-key encryption is simpler in this regard, since the encryption key does not require secure distribution. However, symmetric encryption algorithms are much faster to run than public-key encryption algorithms, making them far more desirable to implement in practice.

The advantages and disadvantages of these two different types of encryption mechanism tend to lead to two different ways of deploying them in real systems:

1. If the key distribution problem is relatively easy to overcome then only the faster symmetric encryption will be used. This is the case, for example, in mobile phone and payment card networks, where symmetric keys can be issued in advance to users on their SIM and payment cards.
2. In applications where it is not straightforward to solve the key distribution problem, such as when setting up a secure connection with a website (where the user may not have any prior relationship with the site), then a hybrid solution is adopted. The relatively slow public-key encryption is used to encrypt a symmetric key, which then encrypts the bulk data using symmetric encryption. The recipient first decrypts the symmetric key, then decrypts the main message.

Symmetric encryption algorithms are themselves classified into *block ciphers*, which encrypt chunks of data (typically 128 bits) at a time, and *stream ciphers*, which encrypt each bit of data individually. The AES is the most common block cipher and is a good general purpose encryption mechanism. Stream ciphers have special properties that make them more desirable for applications such as protecting voice traffic and streaming. An example is the A5 family of

stream ciphers used for encrypting mobile phone calls. Well-known public-key encryption algorithms include RSA and algorithms based on elliptic-curves.

Data integrity tools

For many applications, including financial transactions, it is arguably more important to ensure the data is correct than keep it secret. A data integrity mechanism does not prevent data from being altered, but does enable modification to be detected. Cryptography provides a number of different types of data integrity mechanism, each of which subtly vary in the assurances they provide.

The weakest tool is a cryptographic *hash function*. This tool simply generates a short digest of some data, known as a *hash*. In other words, a cryptographic hash function takes an arbitrary amount of data as an input, and compresses this data into a short hash. The hash function is carefully designed so that even a single bit change to the input will result in a completely different hash being output.

In principle, this means that hash functions could be used to provide a degree of data integrity. If we send some data alongside its hash, the recipient can recalculate the hash locally and check that the received hash matches the locally computed one. This process will detect accidental errors. However, hash functions are unusual cryptographic algorithms because they do not involve a secret key, which means that anyone can compute the hash of some data. Hence, an attacker intercepting some data could modify it, compute a new hash, and then replace the original hash with the modified one. This is not a flaw, because cryptographic hash functions are not explicitly designed to be data integrity tools. In fact they have many different uses in cryptography, mainly as components of other cryptographic algorithms and protocols. Examples of hash functions include MD5, SHA-2 and SHA-3.

A hash function can be converted into a more useful data integrity tool by introducing a secret key. This is the idea behind a *message authentication code* (MAC), which is illustrated in Figure 2.

As for symmetric encryption, a MAC requires the sender and receiver of some data to share a secret key in advance. The sender inputs the data and key into the MAC algorithm, which outputs a MAC. The data is then sent to the receiver alongside the MAC. The receiver inputs the data and the key into the same MAC algorithm and checks whether the output matches the MAC that was sent. If it does then the receiver can be confident that the data has not been altered, since nobody else knows the key and so could not have computed a false MAC. Note that a MAC not only provides data integrity, but also provides a clear indication of who sent the message, since only a legitimate key holder can compute a MAC. This property is often called *data origin authentication*. Examples of commonly deployed MACs include CMAC and HMAC.

A sometimes undesirable feature of a MAC is that it cannot be used to distinguish between legitimate holders of the key. While it provides data integrity, the creator of the MAC can be

any of the holders of the MAC key. On occasions where we wish to link data explicitly to a single entity, then a *digital signature scheme* can be used. This the public-key equivalent of a MAC, where a *digital signature* on some data is created by inputting the data (or, more commonly, a hash of the data) and a secret *signature key* into a signature algorithm. The recipient uses a public *verification key* to check whether the received signature matches the message that was sent. Since the signature key is not shared by any other parties, successful verification indicates that the data is not just unchanged, but links the data to the unique holder of the signature key. A digital signature can thus, in theory, be presented before a judge as evidence that a particular entity sent it. This property is sometimes called *non-repudiation*. The most commonly deployed digital signature algorithms are RSA and DSA.

Entity authentication tools

The invisibility of cyberspace makes it is extremely important to know who is on the end of a communication channel. Thus almost all uses of cryptography begin with some form of entity authentication. While not all entity authentication mechanisms, such as passwords and biometrics, are cryptographic, most of the strongest are.

The commonest way of providing entity authentication using cryptography is to require the authenticating party to provide evidence that they know a secret cryptographic key. For example, this is how mobile operators authenticate customers. The authenticating party is issued with a key in advance (in this case on a SIM card). The customer (phone) is then asked to perform a cryptographic computation using the key that was issued to them. The operator then checks whether this computation is correct. If it is, they believe that it must be the entity they issued the key to that is currently trying to communicate. A similar principle applies to accessing an online back account, opening a modern car door, etc. There are many different ways in which this basic idea is instantiated in practice.

Cryptosystems

We have discussed a number of cryptographic mechanisms that can be used to provide specific security services in cyberspace. In practice, real applications typically combine these in different ways. For example, securing a web connection involves establishing entity authentication of the server, then encrypting and providing data origin authentication for all subsequently exchanged messages.

It is important to recognize that cryptography will only deliver these security services if considered as part of a wider system. Consider, for example, the case of two parties using encryption to provide confidentiality of a file attachment exchanged by email. Assuming they have chosen a good encryption algorithm, the true security of this process also depends on:

- *End point security*. The file is only encrypted during transit, but will potentially exist in unencrypted form at both ends of the communication.

- *Implementation*. If the encryption scheme is not implemented correctly then the file protection might not be as secure as intended.
- *Key management*. If the decryption key management is compromised in any way (leaked during distribution or stolen from storage) then an attacker might also be able to decrypt the file.

Taking this system view of how cryptography is deployed is critical. Cryptography alone does not secure cyberspace. But cyberspace cannot be secure without cryptography.

Further reading

Piper and Murphy [1] is a brief and accessible introduction to cryptography. A more comprehensive introduction, aimed at security practitioners and avoiding the mathematical detail, is Martin [2]. Another good guide for security practitioners is Ferguson, Schneier and Kohno [3]. Paar and Pelzl [4] is one of the more accessible introductions that includes some of the mathematics behind cryptography.

[1] F. Piper and S. Murphy, Cryptography: A Very Short Introduction, Oxford, 2002.
[2] K.M. Martin, Everyday Cryptography, Oxford, 2nd Edition, 2017.
[3] N. Ferguson, B. Schneier and T. Kohno, Cryptography Engineering, Wiley, 2010.
[4] C.Paar and J. Pelzl, Understanding Cryptography, Springer, 2011.