Infosec Overview

Information security (infosec) is a vast field. The field has grown and evolved greatly in the last few years. It offers many specializations, including but not limited to:

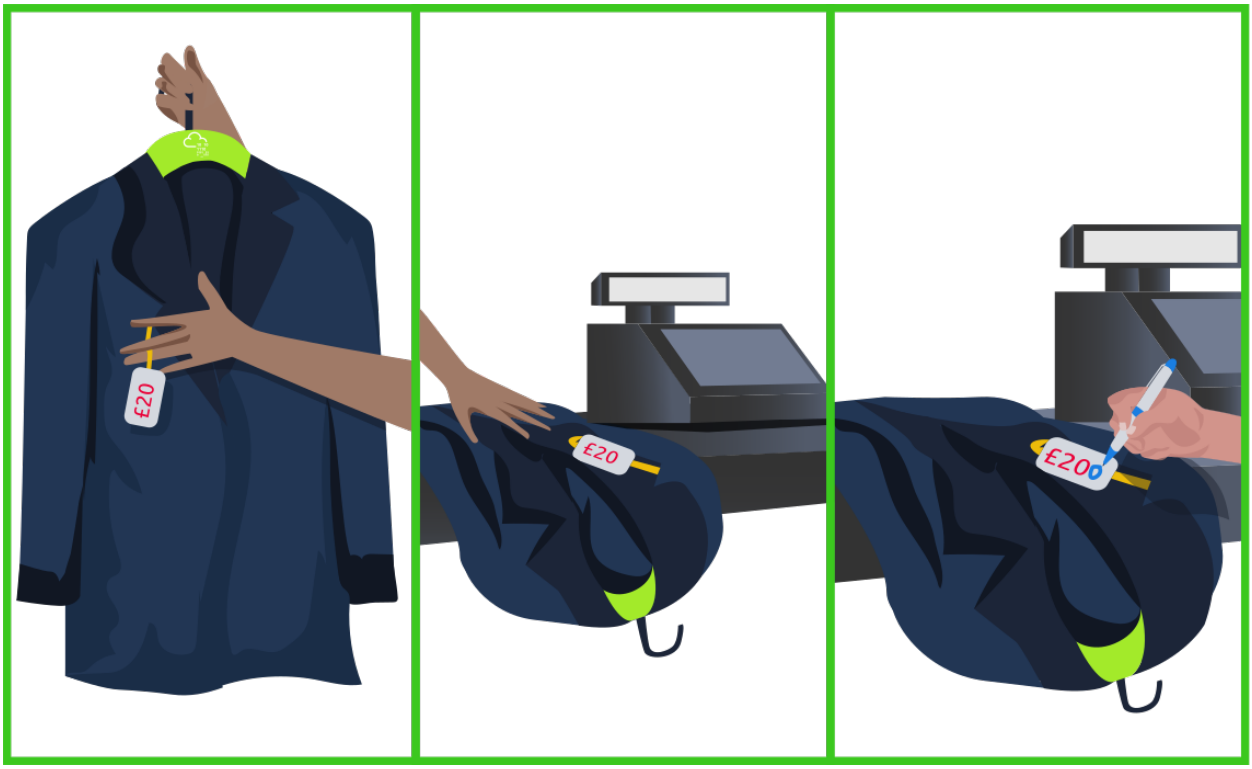
- Network and infrastructure security
- Application security
- Security testing
- Systems auditing
- Business continuity planning
- Digital forensics
- Incident detection and response

In a nutshell, infosec is the practice of protecting data from unauthorized access, changes, unlawful use, disruption, etc. Infosec professionals also take actions to reduce the overall impact of any such incident.

Data can be electronic or physical and tangible (e.g., design blueprints) or intangible (knowledge). A common phrase that will come up many times in our infosec career is protecting the "confidentiality, integrity, and availability of data," or the CIA triad.

Before we can describe something as *secure*, we need to consider better what makes up security. When you want to judge the security of a system, you need to think in terms of the security triad: confidentiality, integrity, and availability (CIA).

- **Confidentiality** ensures that only the intended persons or recipients can access the data.
- **Integrity** aims to ensure that the data cannot be altered; moreover, we can detect any alteration if it occurs.
- **Availability** aims to ensure that the system or service is available when needed.



Let's consider the CIA security triad in the case of placing an order for online shopping:

- **Confidentiality**: During online shopping, you expect your credit card number to be disclosed only to the entity that processes the payment. If you doubt that your credit card information will be disclosed to an untrusted party, you will most likely refrain from continuing with the transaction. Moreover, if a data breach results in the disclosure of personally identifiable information, including credit cards, the company will incur huge losses on multiple levels.
- **Integrity**: After filling out your order, if an intruder can alter the shipping address you have submitted, the package will be sent to someone else. Without data integrity, you might be very reluctant to place your order with this seller.
- **Availability**: To place your online order, you will either browse the store's website or use its official app. If the service is unavailable, you won't be able to browse the products or place an order. If you continue to face such technical issues, you might eventually give up and start looking for a different online store.

Let's consider the CIA as it relates to patient records and related systems:

- **Confidentiality**: According to various laws in modern countries, healthcare providers must ensure and maintain the confidentiality of medical records. Consequently, healthcare providers can be held legally accountable if they illegally disclose their patients' medical records.
- **Integrity**: If a patient record is accidentally or maliciously altered, it can lead to the wrong treatment being administered, which, in turn, can lead to a life-threatening situation. Hence, the system would be useless and potentially harmful without ensuring the integrity of medical records.
- **Availability**: When a patient visits a clinic to follow up on their medical condition, the system must be available. An unavailable system would mean that the medical practitioner cannot access the patient's records and consequently won't know if any current symptoms are related to the patient's medical history. This situation can make the medical diagnosis more challenging and error-prone.

The emphasis does not need to be the same on all three security functions. One example would be a university announcement; although it is usually not confidential, the document's integrity is critical.

Beyond CIA

Going one more step beyond the CIA security triad, we can think of:

- **Authenticity**: Authentic means not fraudulent or counterfeit. Authenticity is about ensuring that the document/file/data is from the claimed source.
- **Nonrepudiation**: Repudiate means refusing to recognize the validity of something. Nonrepudiation ensures that the original source cannot deny that they are the source of a

particular document/file/data. This characteristic is indispensable for various domains, such as shopping, patient diagnosis, and banking.

These two requirements are closely related. The need to tell authentic files or orders from fake ones is indispensable. Moreover, ensuring that the other party cannot deny being the source is vital for many systems to be usable.

In online shopping, depending on your business, you might tolerate attempting to deliver a t-shirt with cash-on-delivery and learn later that the recipient never placed such an order. However, no company can tolerate shipping 1000 cars to discover that the order is fake. In the example of a shopping order, you want to confirm that the said customer indeed placed this order; that's authenticity. Moreover, you want to ensure they cannot deny placing this order; that's nonrepudiation.

As a company, if you receive a shipment order of 1000 cars, you need to ensure the authenticity of this order; moreover, the source should not be able to deny placing such an order. Without authenticity and nonrepudiation, the business cannot be conducted.

Parkerian Hexad

In 1998, Donn Parker proposed the Parkerian Hexad, a set of six security elements. They are:

1. Availability
2. Utility
3. Integrity
4. Authenticity
5. Confidentiality
6. Possession

We have already covered four of the above six elements. Let's discuss the remaining two elements:

- **Utility**: Utility focuses on the usefulness of the information. For instance, a user might have lost the decryption key to access a laptop with encrypted storage. Although the user still has the laptop with its disk(s) intact, they cannot access them. In other words, although still available, the information is in a form that is not useful, i.e., of no utility.
- **Possession**: This security element requires that we protect the information from unauthorized taking, copying, or controlling. For instance, an adversary might take a backup drive, meaning we lose possession of the information as long as they have the drive. Alternatively, the adversary might succeed in encrypting our data using ransomware; this also leads to the loss of possession of the data.

The security of a system is attacked through one of several means. It can be via the disclosure of secret data, alteration of data, or destruction of data.

- **Disclosure** is the opposite of confidentiality. In other words, disclosure of confidential data would be an attack on confidentiality.
- **Alteration** is the opposite of Integrity. For example, the integrity of a cheque is indispensable.
- **Destruction/Denial** is the opposite of Availability.

The opposite of the CIA Triad would be the DAD Triad: Disclosure, Alteration, and Destruction.

Consider the previous example of patient records and related systems:

- Disclosure: As in most modern countries, healthcare providers must maintain medical records' confidentiality. As a result, if an attacker succeeds in stealing some of these medical records and dumping them online to be viewed publicly, the health care provider will incur a loss due to this data disclosure attack.
- Alteration: Consider the gravity of the situation if the attacker manages to modify patient medical records. This alteration attack might lead to the wrong treatment being administered, and consequently, this alteration attack could be life-threatening.
- Destruction/Denial: Consider the case where a medical facility has gone completely paperless. If an attacker manages to make the database systems unavailable, the facility will not be able to function properly. They can go back to paper temporarily; however, the patient records won't be available. This denial attack would stall the whole facility.

Protecting against disclosure, alteration, and destruction/denial is of utter significance. This protection is equivalent to working to maintain confidentiality, integrity and availability.

Protecting confidentiality and integrity to an extreme can restrict availability, and increasing availability to an extreme can result in losing confidentiality and integrity. Good security principles implementation requires a balance between the three.

Fundamental Concepts of Security Models

We have learned that the security triad is represented by Confidentiality, Integrity, and Availability (CIA). One might ask, how can we create a system that ensures one or more security functions? The answer would be in using security models. In this task, we will introduce three foundational security models:

- Bell-LaPadula Model
- The Biba Integrity Model
- The Clark-Wilson Model

Bell-LaPadula Model

The Bell-LaPadula Model aims to achieve **confidentiality** by specifying three rules:

- **Simple Security Property**: This property is referred to as “no read up”; it states that a subject at a lower security level cannot read an object at a higher security level. This rule prevents access to sensitive information above the authorized level.
- **Star Security Property**: This property is referred to as “no write down”; it states that a subject at a higher security level cannot write to an object at a lower security level. This rule prevents the disclosure of sensitive information to a subject of lower security level.
- **Discretionary-Security Property**: This property uses an access matrix to allow read and write operations. An example access matrix is shown in the table below and used in conjunction with the first two properties.

Subjects	Object A	Object B
Subject 1	Write	No access
Subject 2	Read/Write	Read

The first two properties can be summarized as “write up, read down.” You can share confidential information with people of higher security clearance (write up), and you can receive confidential information from people with lower security clearance (read down).

There are certain limitations to the Bell-LaPadula model. For example, it was not designed to handle file-sharing.

Biba Model

The Biba Model aims to achieve **integrity** by specifying two main rules:

- **Simple Integrity Property**: This property is referred to as “no read down”; a higher integrity subject should not read from a lower integrity object.
- **Star Integrity Property**: This property is referred to as “no write up”; a lower integrity subject should not write to a higher integrity object.

These two properties can be summarized as “read up, write down.” This rule is in contrast with the Bell-LaPadula Model, and this should not be surprising as one is concerned with confidentiality while the other is with integrity.

Biba Model suffers from various limitations. One example is that it does not handle internal threats (insider threat).

Clark-Wilson Model

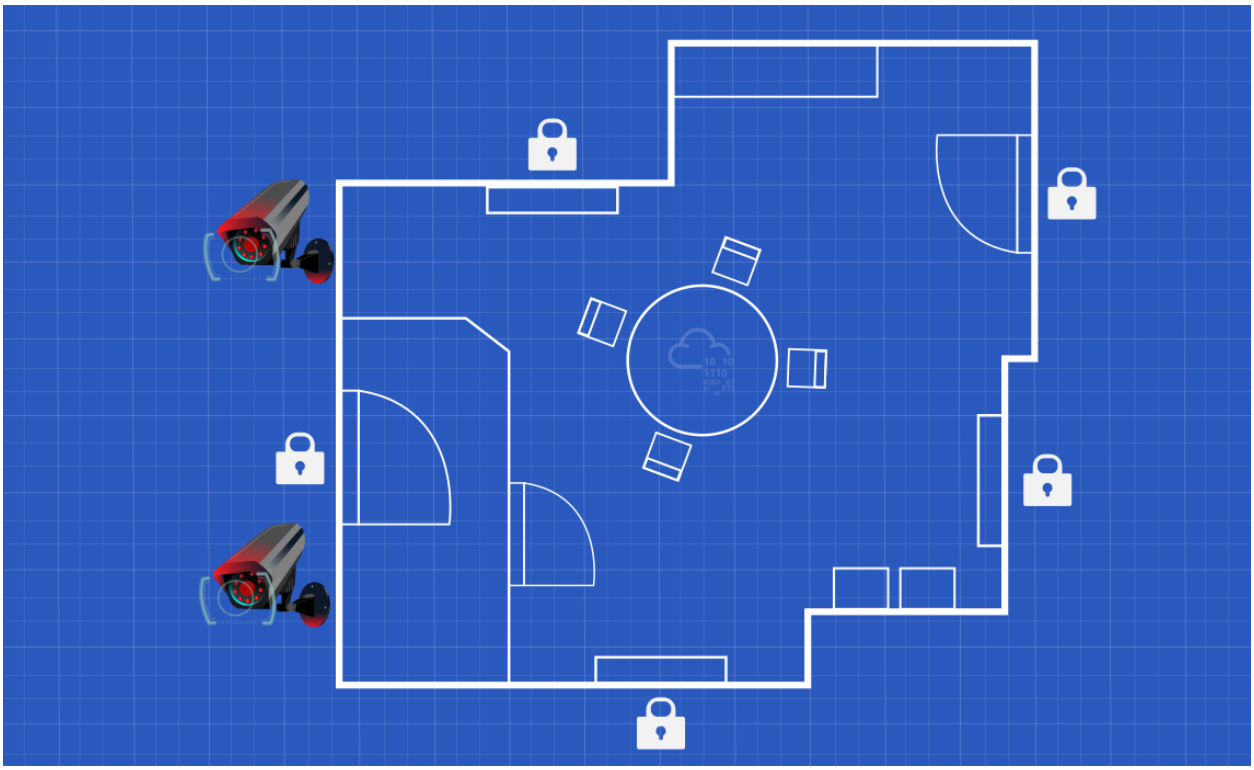
The Clark-Wilson Model also aims to achieve integrity by using the following concepts:

- **Constrained Data Item (CDI)**: This refers to the data type whose integrity we want to preserve.
- **Unconstrained Data Item (UDI)**: This refers to all data types beyond CDI, such as user and system input.
- **Transformation Procedures (TPs)**: These procedures are programmed operations, such as read and write, and should maintain the integrity of CDIs.
- **Integrity Verification Procedures (IVPs)**: These procedures check and ensure the validity of CDIs.

We covered only three security models. The reader can explore many additional security models. Examples include:

- Brewer and Nash model
- Goguen-Meseguer model
- Sutherland model
- Graham-Denning model
- Harrison-Ruzzo-Ullman model

Defence-in-Depth



Defence-in-Depth refers to creating a security system of multiple levels; hence it is also called Multi-Level Security.

Consider the following analogy: you have a locked drawer where you keep your important documents and pricey stuff. The drawer is locked; however, do you want this drawer lock to be the only thing standing between a thief and your expensive items? If we think of multi-level security, we would prefer that the drawer be locked, the relevant room be locked, the main door of the apartment be locked, the building gate be locked, and you might even want to throw in a few security cameras along the way. Although these multiple levels of security cannot stop every thief, they would block most of them and slow down the others.

ISO/IEC 19249

The International Organization for Standardization (ISO) and the International Electrotechnical Commission (IEC) have created the ISO/IEC 19249. In this task, we will brush briefly upon ISO/IEC 19249:2017 *Information technology - Security techniques - Catalogue of architectural and design principles for secure products, systems and applications*. The purpose is to have a better idea of what international organizations would teach regarding security principles.

ISO/IEC 19249 lists five *architectural* principles:

1. **Domain Separation**: Every set of related components is grouped as a single entity; components can be applications, data, or other resources. Each entity will have its own domain and be assigned a common set of security attributes. For example, consider the x86 processor privilege levels: the operating system kernel can run in ring 0 (the most privileged level). In contrast, user-mode applications can run in ring 3 (the least privileged level). Domain separation is included in the Goguen-Meseguer Model.
2. **Layering**: When a system is structured into many abstract levels or layers, it becomes possible to impose security policies at different levels; moreover, it would be feasible to validate the operation. Let's consider the OSI (Open Systems Interconnection) model with its seven layers in networking. Each layer in the OSI model provides specific services to the layer above it. This layering makes it possible to impose security policies and easily validate that the system is working as intended. Another example from the programming world is disk operations; a programmer usually uses the disk read and write functions provided by the chosen high-level programming language. The programming language hides the low-level system calls and presents them as more user-friendly methods. Layering relates to Defence in Depth.
3. **Encapsulation**: In object-oriented programming (OOP), we hide low-level implementations and prevent direct manipulation of the data in an object by providing specific methods for that purpose. For example, if you have a clock object, you would provide a method increment() instead of giving the user direct access to the seconds variable. The aim is to prevent invalid values for your variables. Similarly, in larger systems, you would use (or even design) a proper Application Programming Interface (API) that your application would use to access the database.
4. **Redundancy**: This principle ensures availability and integrity. There are many examples related to redundancy. Consider the case of a hardware server with two built-in power supplies: if one power supply fails, the system continues to function. Consider a RAID 5 configuration with three drives: if one drive fails, data remains available using the remaining two drives. Moreover, if data is improperly changed on one of the disks, it would be detected via the parity, ensuring the data's integrity.
5. **Virtualization**: With the advent of cloud services, virtualization has become more common and popular. The concept of virtualization is sharing a single set of hardware among multiple operating systems. Virtualization provides sandboxing capabilities that improve security boundaries, secure detonation, and observance of malicious programs.

ISO/IEC 19249 teaches five *design* principles:

1. **Least Privilege**: You can also phrase it informally as "need-to basis" or "need-to-know basis" as you answer the question, "who can access what?" The principle of least privilege teaches that

you should provide the least amount of permissions for someone to carry out their task and nothing more. For example, if a user needs to be able to view a document, you should give them read rights without write rights.

2. **Attack Surface Minimisation**: Every system has vulnerabilities that an attacker might use to compromise a system. Some vulnerabilities are known, while others are yet to be discovered. These vulnerabilities represent risks that we should aim to minimize. For example, in one of the steps to harden a Linux system, we would disable any service we don't need.
3. **Centralized Parameter Validation**: Many threats are due to the system receiving input, especially from users. Invalid inputs can be used to exploit vulnerabilities in the system, such as denial of service and remote code execution. Therefore, parameter validation is a necessary step to ensure the correct system state. Considering the number of parameters a system handles, the validation of the parameters should be centralized within one library or system.
4. **Centralized General Security Services**: As a security principle, we should aim to centralize all security services. For example, we would create a centralized server for authentication. Of course, you might take proper measures to ensure availability and prevent creating a single point of failure.
5. **Preparing for Error and Exception Handling**: Whenever we build a system, we should take into account that errors and exceptions do and will occur. For instance, in a shopping application, a customer might try to place an order for an out-of-stock item. A database might get overloaded and stop responding to a web application. This principle teaches that the systems should be designed to fail safe; for example, if a firewall crashes, it should block all traffic instead of allowing all traffic. Moreover, we should be careful that error messages don't leak information that we consider confidential, such as dumping memory content that contains information related to other customers.

Zero Trust versus Trust but Verify

Trust is a very complex topic; in reality, we cannot function without trust. If one were to think that the laptop vendor has installed spyware on the laptop, they would most likely end up rebuilding the system. If one were to mistrust the hardware vendor, they would stop using it completely. If we think of trust on a business level, things only become more sophisticated; however, we need some guiding security principles. Two security principles that are of interest to us regarding trust:

- Trust but Verify
- Zero Trust

Trust but Verify: This principle teaches that we should always verify even when we trust an entity and its behaviour. An entity might be a user or a system. Verifying usually requires setting up proper logging mechanisms; verifying indicates going through the logs to ensure everything is normal. In reality, it is not feasible to verify everything; just think of the work it takes to review all the actions taken by a single entity, such as Internet pages browsed by a single user. This requires automated security mechanisms, such as proxy, intrusion detection, and intrusion prevention systems.

Zero Trust: This principle treats trust as a vulnerability, and consequently, it caters to insider-related threats. After considering trust as a vulnerability, zero trust tries to eliminate it. It is teaching indirectly, “never trust, always verify.” In other words, every entity is considered adversarial until proven otherwise. Zero trust does not grant trust to a device based on its location or ownership. This approach contrasts with older models that would trust internal networks or enterprise-owned devices. Authentication and authorization are required before accessing any resource. As a result, if any breach occurs, the damage would be more contained if a zero trust architecture had been implemented.

Microsegmentation is one of the implementations used for Zero Trust. It refers to the design where a network segment can be as small as a single host. Moreover, communication between segments requires authentication, access control list checks, and other security requirements.

There is a limit to how much we can apply zero trust without negatively impacting a business; however, this does not mean that we should not apply it as long as it is feasible.

Threat versus Risk

There are three terms that we need to take note of to avoid any confusion.

- **Vulnerability**: Vulnerable means susceptible to attack or damage. In information security, a vulnerability is a weakness.
- **Threat**: A threat is a potential danger associated with this weakness or vulnerability.
- **Risk**: The risk is concerned with the likelihood of a threat actor exploiting a vulnerability and the consequent impact on the business.

Away from information systems, a showroom with doors and windows made of standard glass suffers a weakness, or *vulnerability*, due to the nature of glass. Consequently, there is a *threat* that the glass doors and windows can be broken. The showroom owners should contemplate the *risk*, i.e. the likelihood that a glass door or window gets broken and the resulting impact on the business.

Consider another example directly related to information systems. You work for a hospital that uses a particular database system to store all the medical records. One day, you are following the latest security news, and you learn that the used database system is not only vulnerable but also a proof-of-concept working exploit code has been released; the released exploit code indicates that the threat is real. With this knowledge, you must consider the resulting risk and decide the next steps.

Risk Management Process

Data protection must focus on efficient yet effective policy implementation without negatively affecting an organization's business operations and productivity. To achieve this, organizations must follow a process called the risk management process. This process involves the following five steps:

Step	Explanation
Identifying the Risk	Identifying risks the business is exposed to, such as legal, environmental, market, regulatory, and other types of risks.
Analyze the Risk	Analyzing the risks to determine their impact and probability. The risks should be mapped to the organization's various policies, procedures, and business processes.
Evaluate the Risk	Evaluating, ranking, and prioritizing risks. Then, the organization must decide to accept (unavoidable), avoid (change plans), control (mitigate), or transfer risk (insure).
Dealing with Risk	Eliminating or containing the risks as best as possible. This is handled by interfacing directly with the stakeholders for the system or process that the risk is associated with.
Monitoring Risk	All risks must be constantly monitored. Risks should be constantly monitored for any situational changes that could change their impact score, i.e., from low to medium or high impact.

As mentioned previously, the core tenet of infosec is information assurance, or maintaining the CIA of data and making sure that it is not compromised in any way, shape, or form when an incident occurs. An incident could be a natural disaster, system malfunction, or security incident.

Red Team vs. Blue Team

In infosec, we usually hear the terms red team and blue team. In the simplest terms, the red team plays the attackers' role, while the blue team plays the defenders' part.

Red teamers usually play an adversary role in breaking into the organization to identify any potential weaknesses real attackers may utilize to break the organization's defenses. The most common task on the red teaming side is penetration testing, social engineering, and other similar offensive techniques.

On the other hand, the blue team makes up the majority of infosec jobs. It is responsible for strengthening the organization's defenses by analyzing the risks, coming up with policies, responding to threats and incidents, and effectively using security tools and other similar tasks.

Role of Penetration Testers

A security assessor (network penetration tester, web application penetration tester, red teamer, etc.) helps an organization identify risks in its external and internal networks. These risks may include network or web application vulnerabilities, sensitive data exposure, misconfigurations, or issues that could lead to reputational harm. A good tester can work with a client to identify risks to their organization, provide information on how to reproduce these risks, and guidance on either mitigating or remediating the issues identified during testing.

Assessments can take many forms, from a white-box penetration test against all in-scope systems and applications to identify as many vulnerabilities as possible, to a phishing assessment to assess the risk or employee's security awareness, to a targeted red team assessment built around a scenario to emulate a real-world threat actor.

We must understand the bigger picture of the risks an organization faces and its environment to evaluate and rate vulnerabilities discovered during testing accurately. A deep understanding of the risk management process is critical for anyone starting in information security.

Getting Started with a Pentest Distro

Anyone looking to start a technical path in information security must become comfortable with a wide range of technologies and operating systems. As penetration testers, we must understand how to set up, maintain, and secure both Linux and Windows attack machines. Depending on the client environment or scope of the assessment, we may be using a Linux or Windows VM on our machine, our

base operating system, a cloud Linux box, a VM installed within the client's environment, or even perform testing directly from a client-owned workstation to simulate an insider threat (assume breach scenario).

Choosing a Distro

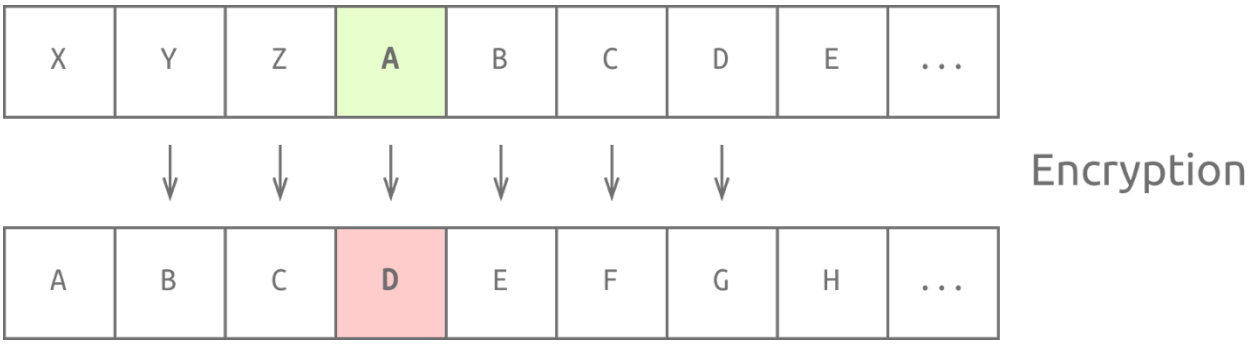
There are many Linux distributions (distros) for penetration testing. There are quite a few Debian-based pre-existing distros preloaded with many tools that we need to perform our assessments. Many of these tools are rarely required, and no distro contains every tool that we need to perform our assessments. As we learn and progress in our careers, we will gravitate to specific tools and have a list of "must-haves" to add to a new distro. As we progress, we may even prefer to fully customize our own pentesting VM from a Debian or Ubuntu base image, but building a fully custom VM is outside this module's scope.

The choice of a distro is individual, and, as mentioned, we can even choose to create and maintain our own from scratch. There are countless Linux distros out there that serve various purposes, some explicitly customized for penetration testing, others geared towards web application penetration testing, forensics, etc.

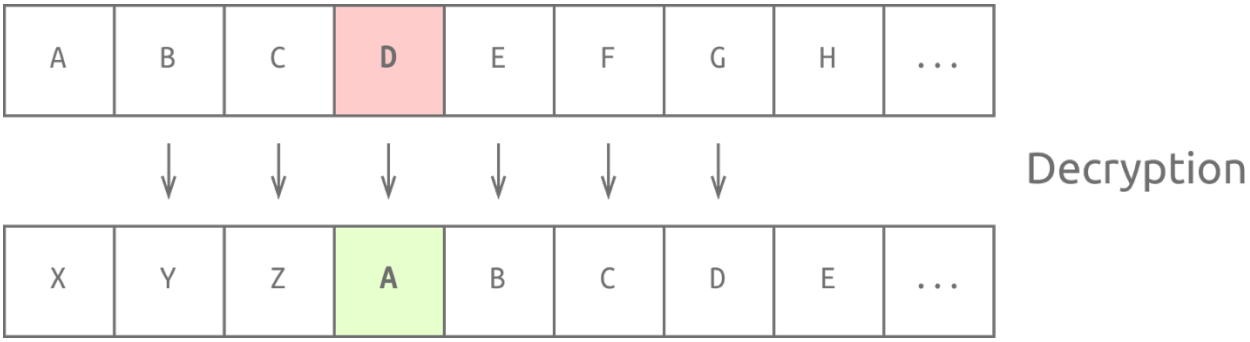
Cryptography: An Introduction

Suppose you want to send a message that no one can understand except the intended recipient. How would you do that?

One of the simplest ciphers is the Caesar cipher, used more than 2000 years ago. Caesar Cipher shifts the letter by a fixed number of places to the left or to the right. Consider the case of shifting by 3 to the right to encrypt, as shown in the figure below.



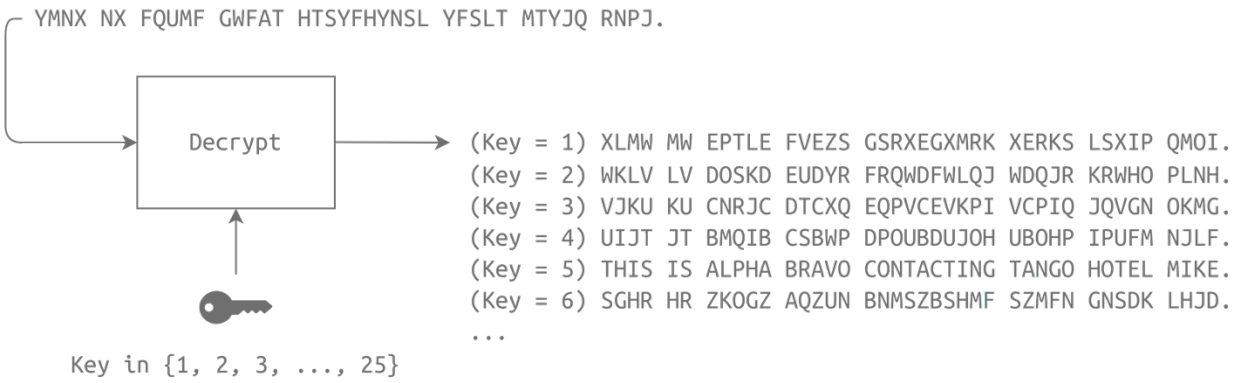
The recipient needs to know that the text was shifted by 3 to the right to recover the original message.



Using the same key to encrypt “TRY HACK ME”, we get “WUB KDFN PH”.

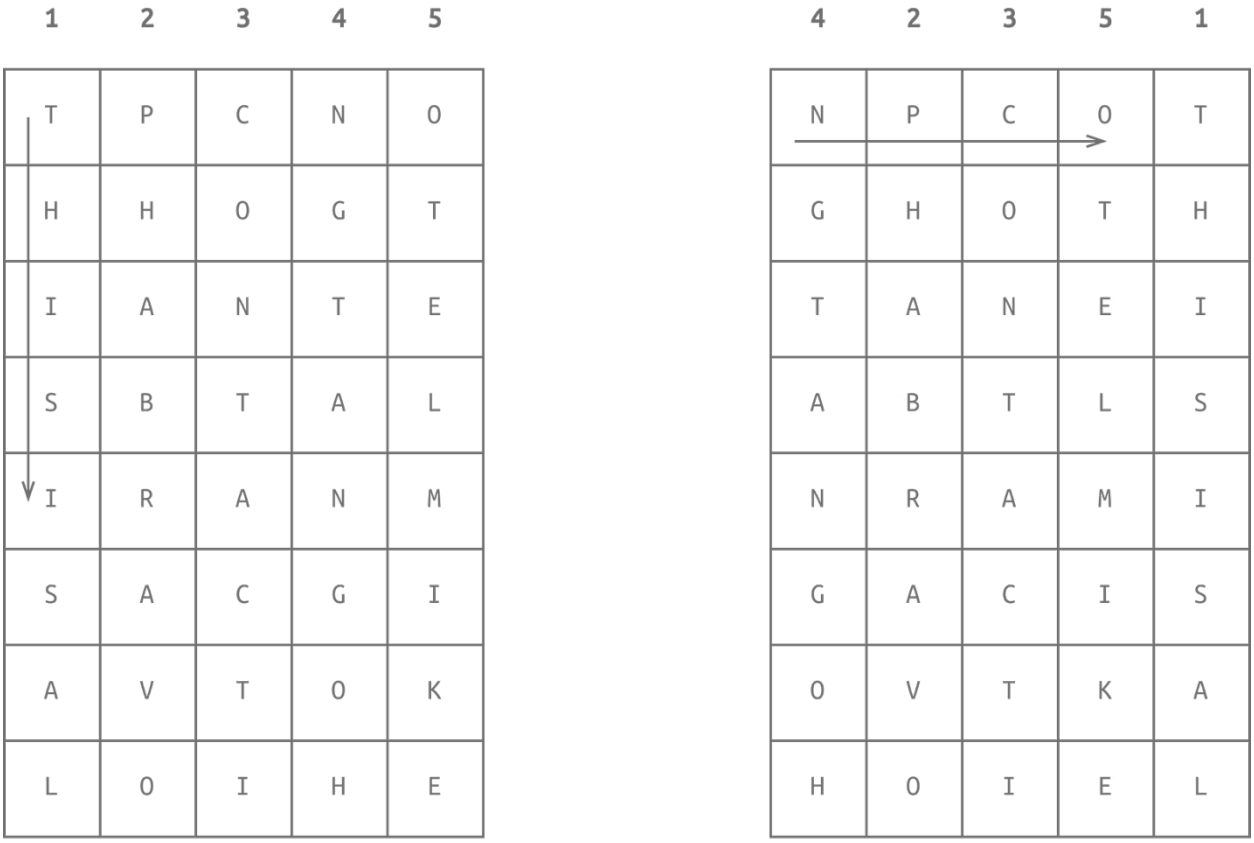
The Caesar Cipher that we have described above can use a key between 1 and 25. With a key of 1, each letter is shifted by one position, where A becomes B, and Z becomes A. With a key of 25, each letter is shifted by 25 positions, where A becomes Z, and B becomes A. A key of 0 means no change; moreover, a key of 26 will also lead to no change as it would lead to a full rotation. Consequently, we conclude that Caesar Cipher has a keyspace of 25; there are 25 different keys that the user can choose from.

Consider the case where you have intercepted a message encrypted using Caesar Cipher: “YMNX NX FQUMF GWFAT HTSYFHYNSL YFSLT MTYJQ RNPJ”. We are asked to decrypt it without knowledge of the key. We can attempt this by using brute force, i.e., we can try all the possible keys and see which one makes the most sense. In the following figure, we noticed that key being 5 makes the most sense, “THIS IS ALPHA BRAVO CONTACTING TANGO HOTEL MIKE.”



Caesar cipher is considered a **substitution cipher** because each letter in the alphabet is substituted with another.

Another type of cipher is called **transposition cipher**, which encrypts the message by changing the order of the letters. Let's consider a simple transposition cipher in the figure below. We start with the message, "THIS IS ALPHA BRAVO CONTACTING TANGO HOTEL MIKE", and the key 42351. After we write the letters of our message by filling one column after the other, we rearrange the columns based on the key and then read the rows. In other words, we write by columns and we read by rows. Also notice that we ignored all the space in the plaintext in this example. The resulting ciphertext "NPCOTGHOTH…" is read one row after the other. In other words, a transposition cipher simply rearranges the order of the letters, unlike the substitution cipher, which substitutes the letters without changing their order.



Plaintext

THIS IS ALPHA BRAVO
CONTACTING TANGO HOTEL MIKE

Ciphertext

NPCOTGHOTHTANEIABTLS
NRAMIGACISOVTKAHOIEL

This task introduced simple substitution and transposition ciphers and applied them to messages made of alphabetic characters. For an encryption algorithm to be considered **secure**, it should be infeasible to recover the original message, i.e., plaintext. (In mathematical terms, we need a **hard** problem, i.e., a problem that cannot be solved in polynomial time. A problem that we can solve in polynomial time is a problem that's feasible to solve even for large input, although it might take the computer quite some time to finish.)

If the encrypted message can be broken in one week, the encryption used would be considered insecure. However, if the encrypted message can be broken in 1 million years, the encryption would be considered practically secure.

Consider the mono-alphabetic substitution cipher, where each letter is mapped to a new letter. For example, in English, you would map "a" to one of the 26 English letters, then you would map "b" to one of the remaining 25 English letters, and then map "c" to one of the remaining 24 English letters, and so on.

For example, we might choose the letters in the alphabet “abcdefghijklmnopqrstuvwxyz” to be mapped to “xpatvrzyjhecsdikbfwunqgmol” respectively. In other words, “a” becomes “x”, “b” becomes “p”, and so on. The recipient needs to know the key, “xpatvrzyjhecsdikbfwunqgmol”, to decrypt the encrypted messages successfully.

This algorithm might look very secure, especially since trying all the possible keys is not feasible. However, different techniques can be used to break a ciphertext using such an encryption algorithm. One weakness of such an algorithm is letter frequency. In English texts, the most common letters are ‘e’, ‘t’, and ‘a’, as they appear at a frequency of 13%, 9.1%, and 8.2%, respectively. Moreover, in English texts, the most common first letters are ‘t’, ‘a’, and ‘o’, as they appear at 16%, 11.7% and 7.6%, respectively. Add to this the fact that most of the message words are dictionary words, and you will be able to break an encrypted text with the alphabetic substitution cipher in no time.

We don’t really need to use the encryption key to decrypt the received ciphertext, “Uyv sxd gyi siqvw x sinduxjd pvzjdw po axffojdz xgxo wsxcc wuidvw.” As shown in the figure below, using a website such as quipqiup, it will take a moment to discover that the original text was “The man who moves a mountain begins by carrying away small stones.” This example clearly indicates that this algorithm is broken and should not be used for confidential communication.

---

quipqiup beta3

quipqiup is a fast and automated cryptogram solver by Edwin Olson. It can solve simple substitution ciphers often found in newspapers, including puzzles like cryptoquips (in which word boundaries are preserved) and patristocrats (inwhi chwor dboun darie saren t).

Puzzle:

"Uyv sxd gyi siqvw x sinduxjd pvzjdw po axffojdz xgxo wsxcc wuidvw."

Clues: For example G=R QVW=THE

Solve

```
0   -1.916   "The man who moves a mountain begins by carrying away small stones."
1   -2.679   "The man who moves a mountain pekins pr jaffrink awar small stones."
2   -2.753   "The man who moves a mountain jedins jr kaggrind awar small stones."
```

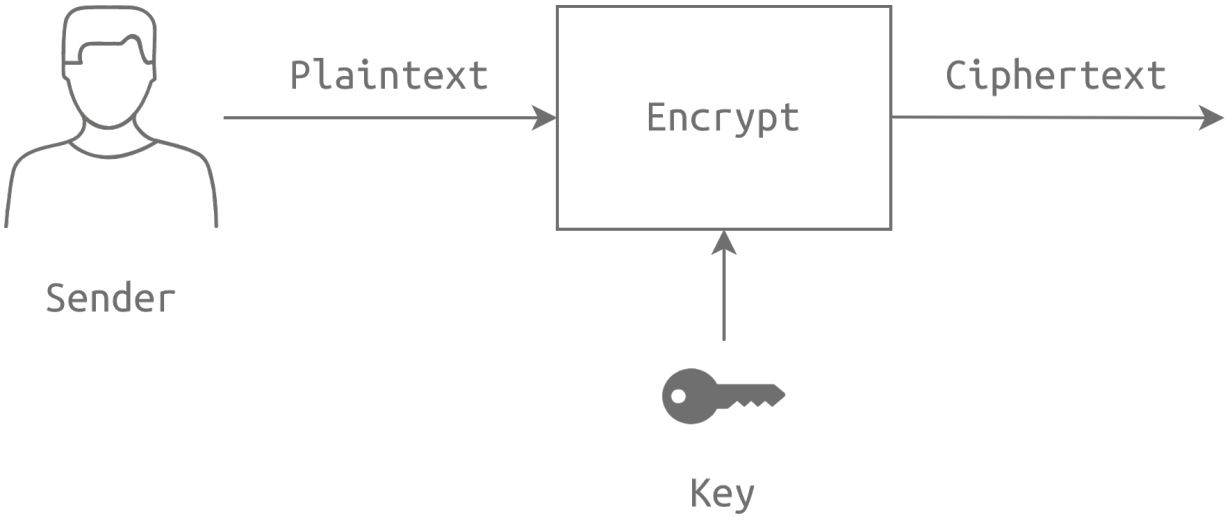
Let’s review some terminology:

- **Cryptographic Algorithm** or **Cipher**: This algorithm defines the encryption and decryption processes.
- **Key**: The cryptographic algorithm needs a key to convert the plaintext into ciphertext and vice versa.
- **Plaintext** is the original message that we want to encrypt
- **Ciphertext** is the message in its encrypted form

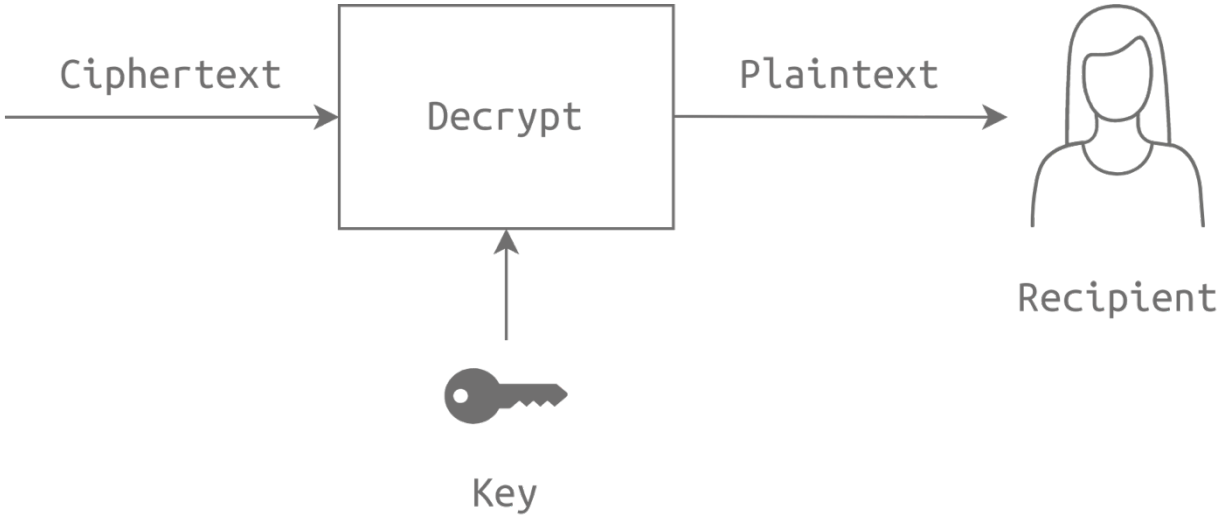
Symmetric Algorithm

A symmetric encryption algorithm uses the same key for encryption and decryption. Consequently, the communicating parties need to agree on a secret key before being able to exchange any messages.

In the following figure, the sender provides the *encrypt* process with the plaintext and the key to get the ciphertext. The ciphertext is usually sent over some communication channel.



On the other end, the recipient provides the *decrypt* process with the same key used by the sender to recover the original plaintext from the received ciphertext. Without knowledge of the key, the recipient won't be able to recover the plaintext.

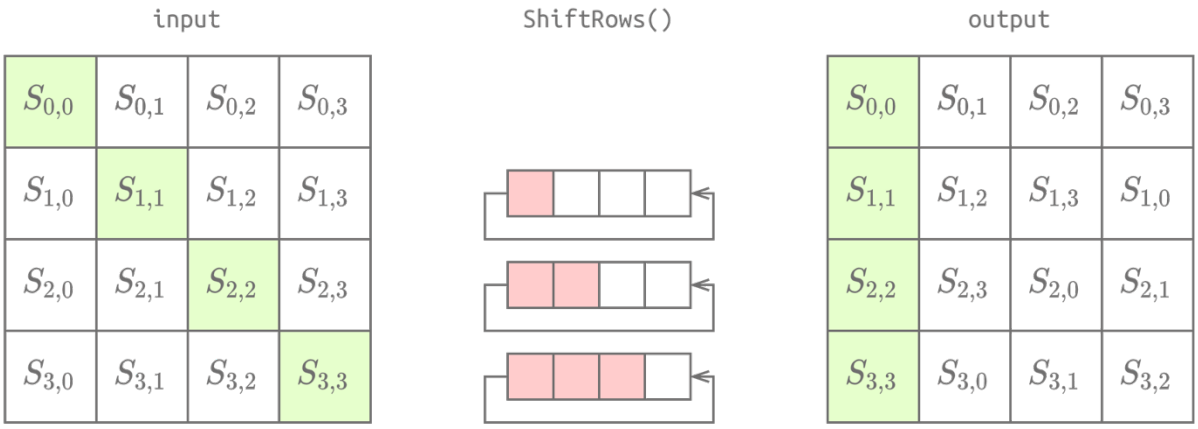


National Institute of Standard and Technology (NIST) published the Data Encryption Standard (DES) in 1977. DES is a symmetric encryption algorithm that uses a key size of 56 bits. In 1997, a challenge to break a message encrypted using DES was solved. Consequently, it was demonstrated that it had

become feasible to use a brute-force search to find the key and break a message encrypted using DES. In 1998, a DES key was broken in 56 hours. These cases indicated that DES could no longer be considered secure.

NIST published the Advanced Encryption Standard (AES) in 2001. Like DES, it is a symmetric encryption algorithm; however, it uses a key size of 128, 192, or 256 bits, and it is still considered secure and in use today. AES repeats the following four transformations multiple times:

1. SubBytes(state): This transformation looks up each byte in a given substitution table (S-box) and substitutes it with the respective value. The state is 16 bytes, i.e., 128 bits, saved in a 4 by 4 array.
2. ShiftRows(state): The second row is shifted by one place, the third row is shifted by two places, and the fourth row is shifted by three places. This is shown in the figure below.
3. MixColumns(state): Each column is multiplied by a fixed matrix (4 by 4 array).
4. AddRoundKey(state): A round key is added to the state using the XOR operation.



The total number of transformation rounds depends on the key size.

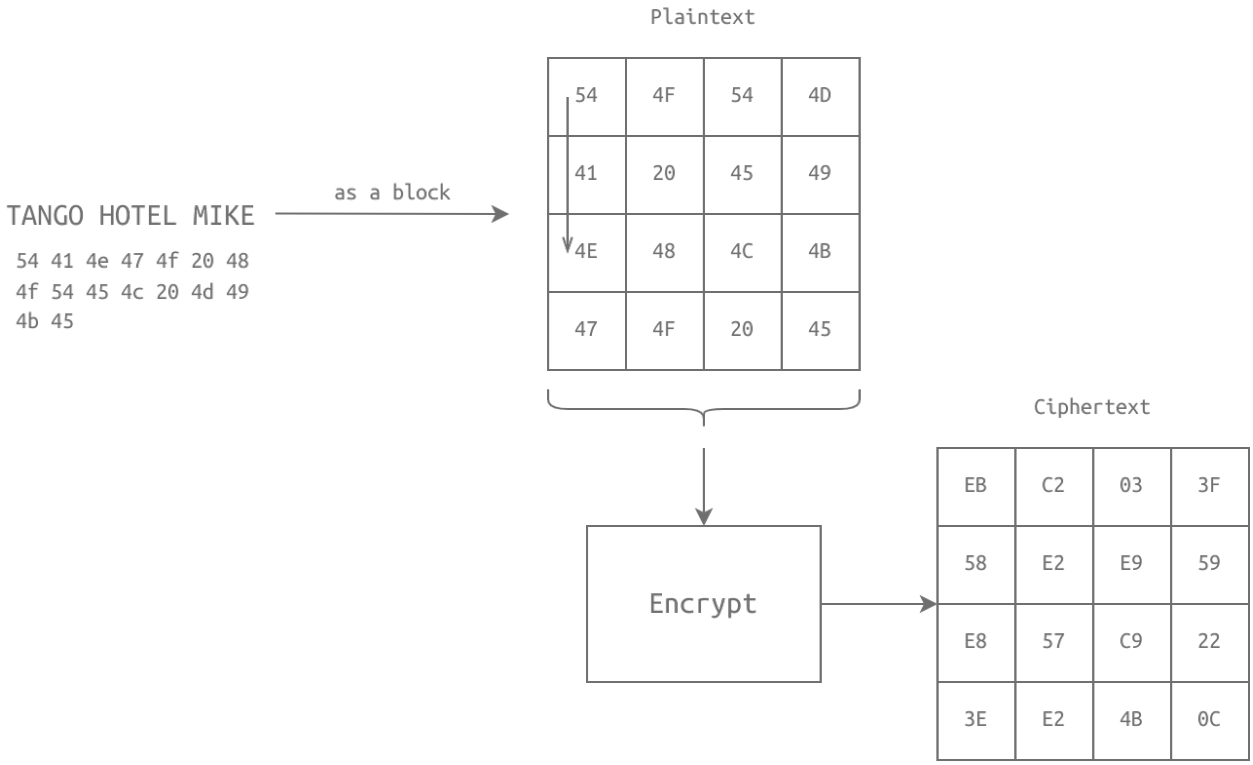
Don't worry if you find this cryptic because it is! Our purpose is not to learn the details of how AES works nor to implement it as a programming library; the purpose is to appreciate the difference in complexity between ancient encryption algorithms and modern ones. If you are curious to dive into details, you can check the AES specifications, including pseudocode and examples in its published standard, FIPS PUB 197.

In addition to AES, many other symmetric encryption algorithms are considered secure. Here is a list of symmetric encryption algorithms supported by GPG (GnuPG) 2.37.7, for example:

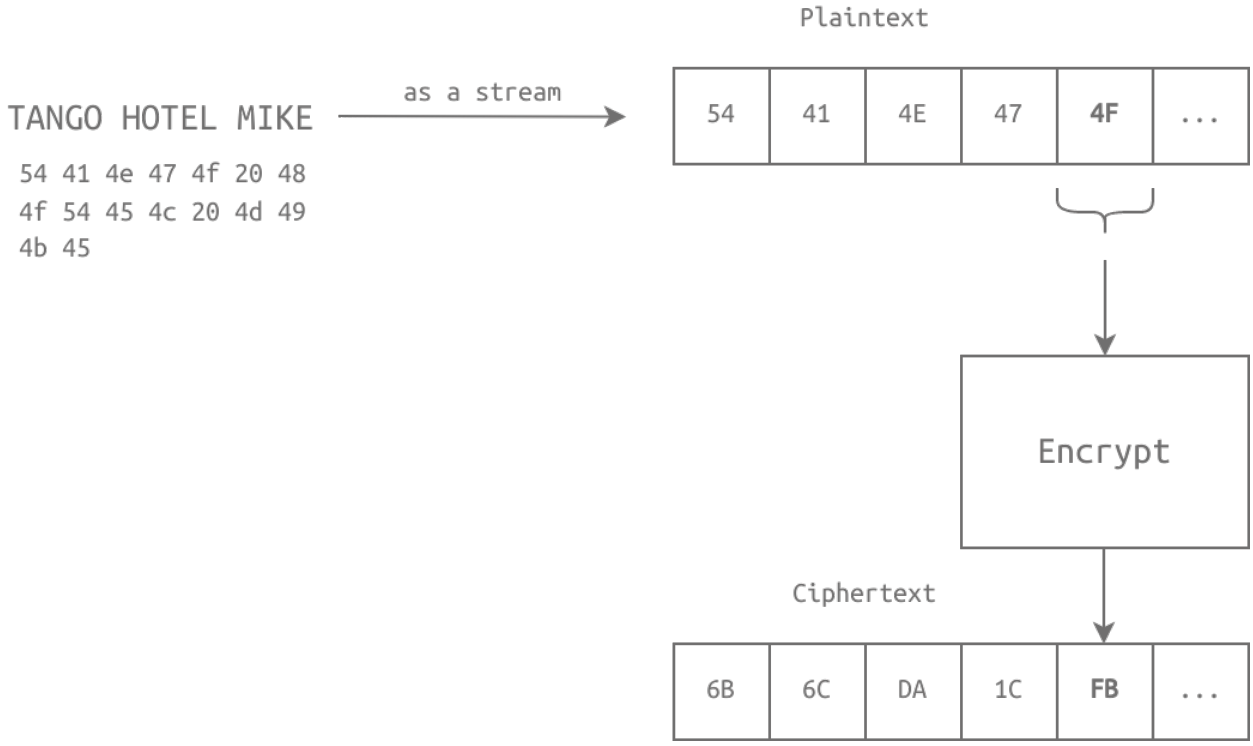
Encryption Algorithm	Notes
AES, AES192, and AES256	AES with a key size of 128, 192, and 256 bits
IDEA	International Data Encryption Algorithm (IDEA)

Encryption Algorithm	Notes
3DES	Triple DES (Data Encryption Standard) and is based on DES. We should note that 3DES will be deprecated in 2023 and disallowed in 2024.
CAST5	Also known as CAST-128. Some sources state that CAST stands for the names of its authors: Carlisle Adams and Stafford Tavares.
BLOWFISH	Designed by Bruce Schneier
TWOFISH	Designed by Bruce Schneier and derived from Blowfish
CAMELLIA128, CAMELLIA192, and CAMELLIA256	Designed by Mitsubishi Electric and NTT in Japan. Its name is derived from the flower camellia japonica.

All the algorithms mentioned so far are block cipher symmetric encryption algorithms. A block cipher algorithm converts the input (plaintext) into blocks and encrypts each block. A block is usually 128 bits. In the figure below, we want to encrypt the plaintext “TANGO HOTEL MIKE”, a total of 16 characters. The first step is to represent it in binary. If we use ASCII, “T” is 0x54 in hexadecimal format, “A” is 0x41, and so on. Every two hexadecimal digits constitute 8 bits and represent one byte. A block of 128 bits is practically 16 bytes and is represented in a 4 by 4 array. The 128-bit block is fed as one unit to the encryption method.



The other type of symmetric encryption algorithm is stream ciphers, which encrypt the plaintext byte by byte. Consider the case where we want to encrypt the message “TANGO HOTEL MIKE”; each character needs to be converted to its binary representation. If we use ASCII, “T” is 0x54 in hexadecimal, while “A” is 0x41, and so on. The encryption method will process one byte at a time. This is represented in the figure below.



Symmetric encryption solves many security problems discussed in the Security Principles room. Let’s say that Alice and Bob met and chose an encryption algorithm and agreed on a specific key. We assume that the selected encryption algorithm is secure and that the secret key is kept safe. Let’s take a look at what we can achieve:

- **Confidentiality**: If Eve intercepted the encrypted message, she wouldn’t be able to recover the plaintext. Consequently, all messages exchanged between Alice and Bob are confidential as long as they are sent encrypted.
- **Integrity**: When Bob receives an encrypted message and decrypts it successfully using the key he agreed upon with Alice, Bob can be sure that no one could tamper with the message across the channel. When using secure modern encryption algorithms, any minor modification to the ciphertext would prevent successful decryption or would lead to gibberish as plaintext.
- **Authenticity**: Being able to decrypt the ciphertext using the secret key also proves the authenticity of the message because only Alice and Bob know the secret key.

We are just getting started, and we know how to maintain confidentiality, check the integrity and ensure the authenticity of the exchanged messages. More practical and efficient approaches will be presented in later tasks. The question, for now, is whether this is scalable.

With Alice and Bob, we needed one key. If we have Alice, Bob, and Charlie, we need three keys: one for Alice and Bob, another for Alice and Charlie, and a third for Bob and Charlie. However, the number of keys grows quickly; communication between 100 users requires almost 5000 different secret keys. (If you are curious about the mathematics behind it, that's 99 + 98 + 97 + … + 1 = 4950).

Moreover, if one system gets compromised, they need to create new keys to be used with the other 99 users. Another problem would be finding a secure channel to exchange the keys with all the other users. Obviously, this quickly grows out of hand.

Asymmetric Algorithm

Symmetric encryption requires the users to find a secure channel to exchange keys. By secure channel, we are mainly concerned with confidentiality and integrity. In other words, we need a channel where no third party can eavesdrop and read the traffic; moreover, no one can change the sent messages and data.

Asymmetric encryption makes it possible to exchange encrypted messages without a secure channel; we just need a reliable channel. By reliable channel, we mean that we are mainly concerned with the channel's integrity and not confidentiality.

When using an asymmetric encryption algorithm, we would generate a key pair: a public key and a private key. The public key is shared with the world, or more specifically, with the people who want to communicate with us securely. The private key must be saved securely, and we must never let anyone access it. Moreover, it is not feasible to derive the private key despite the knowledge of the public key.

How does this key pair work?

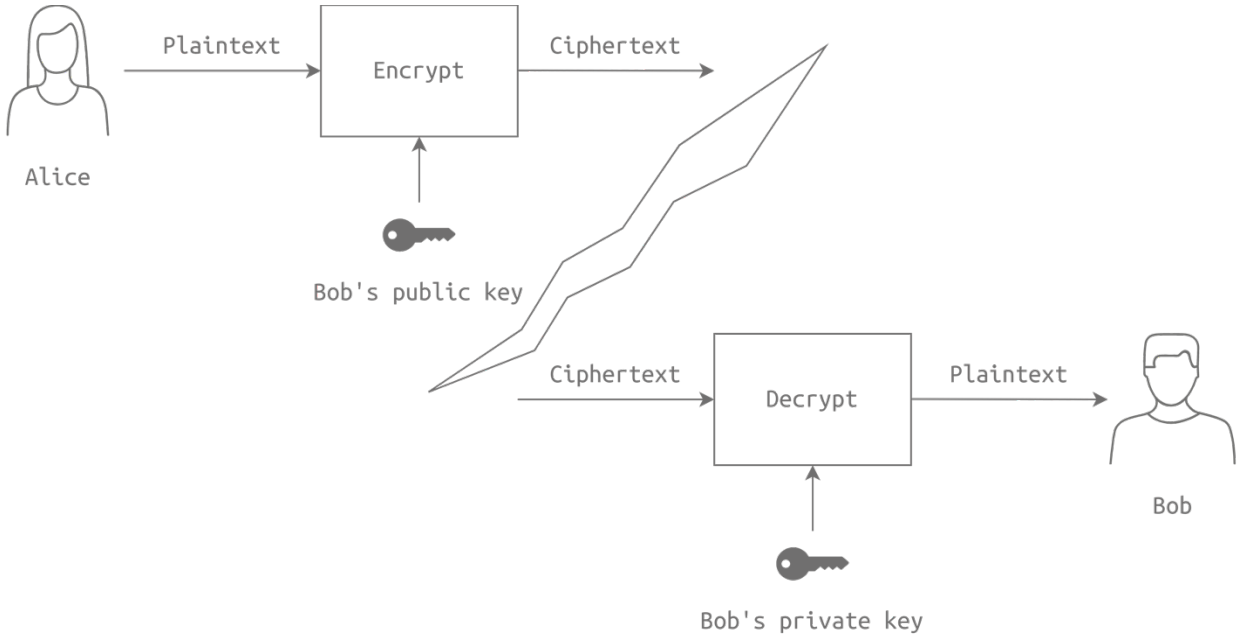
If a message is encrypted with one key, it can be decrypted with the other. In other words:

- If Alice encrypts a message using Bob's public key, it can be decrypted only using Bob's private key.
- Reversely, if Bob encrypts a message using his private key, it can only be decrypted using Bob's public key.

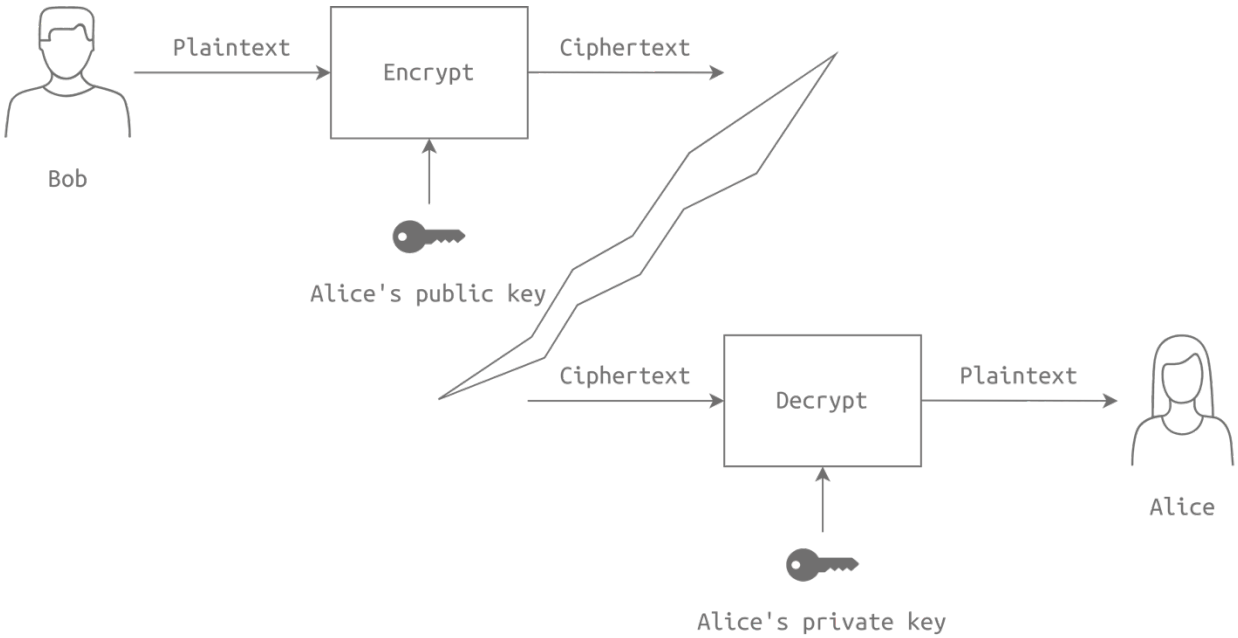
Confidentiality

We can use asymmetric encryption to achieve confidentiality by encrypting the messages using the recipient's public key. In the following two figures, we can see that:

Alice wants to ensure confidentiality in her communication with Bob. She encrypts the message using Bob's public key, and Bob decrypts them using his private key. Bob's public key is expected to be published on a public database or on his website, for instance.



When Bob wants to reply to Alice, he encrypts his messages using Alice's public key, and Alice can decrypt them using her private key.



In other words, it becomes easy to communicate with Alice and Bob while ensuring the confidentiality of the messages. The only requirement is that all parties have their public keys available for interested senders.

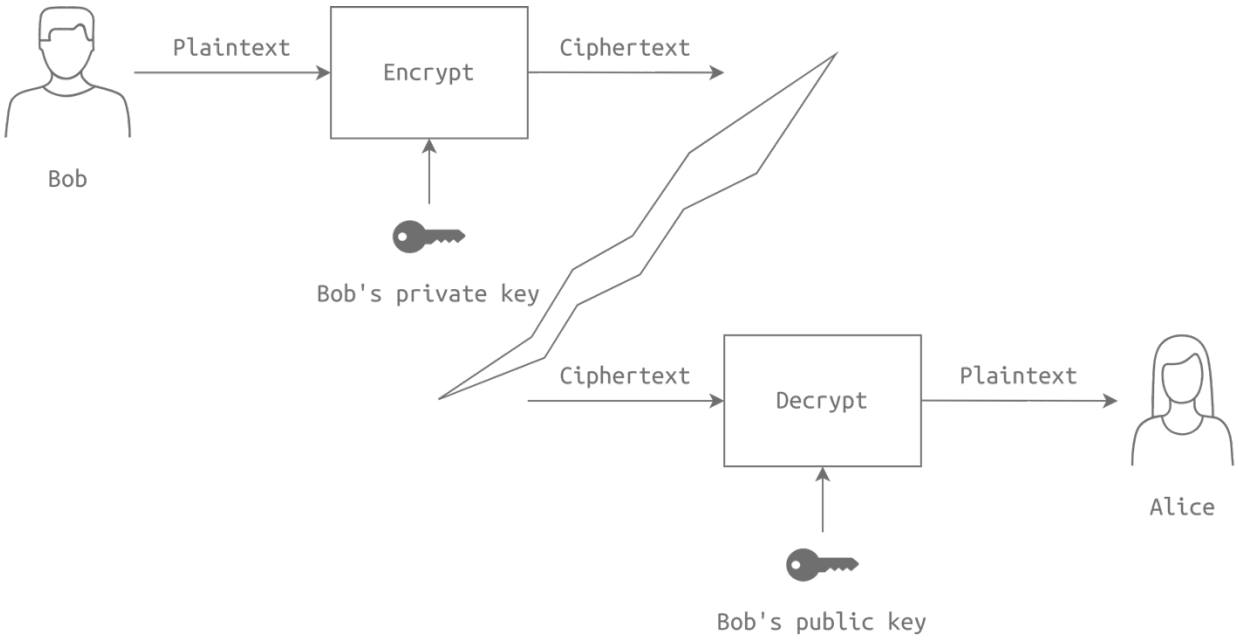
Note: In practice, symmetric encryption algorithms allow faster operations than asymmetric encryption; therefore, we will cover later how we can use the best of both worlds.

Integrity, Authenticity, and Nonrepudiation

Beyond confidentiality, asymmetric encryption can solve integrity, authenticity and nonrepudiation. Let's say that Bob wants to make a statement and wants everyone to be able to confirm that this statement indeed came from him. Bob needs to encrypt the message using his private key; the recipients can decrypt it using Bob's public key. If the message decrypts successfully with Bob's public key, it means that the message was encrypted using Bob's private key. (In practice, he would encrypt a hash of the original message. We will elaborate on this later.)

Being decrypted successfully using Bob's public key leads to a few interesting conclusions.

- First, the message was not altered across the way (communication channel); this proves the message *integrity*.
- Second, knowing that no one has access to Bob's private key, we can be sure that this message did indeed come from Bob; this proves the message *authenticity*.
- Finally, because no one other than Bob has access to Bob's private key, Bob cannot deny sending this message; this establishes *nonrepudiation*.



We have seen how asymmetric encryption can help establish confidentiality, integrity, authenticity, and nonrepudiation. In real-life scenarios, asymmetric encryption can be relatively slow to encrypt large files and vast amounts of data.

PKI and SSL/TLS

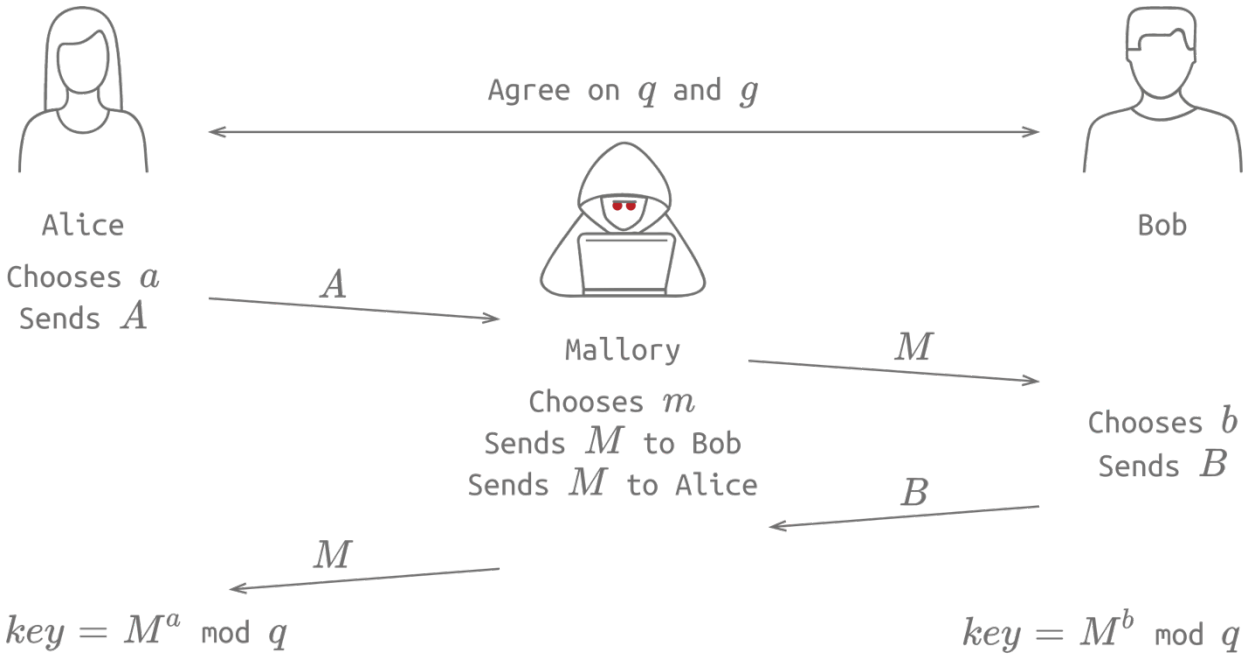
Using a key exchange such as the Diffie-Hellman key exchange allows us to agree on a secret key under the eyes and ears of eavesdroppers. This key can be used with a symmetric encryption algorithm to ensure confidential communication. However, the key exchange we described earlier is not immune to Man-in-the-Middle (MITM) attack. The reason is that Alice has no way of ensuring that she is

communicating with Bob, and Bob has no way of ensuring that he is communicating with Alice when exchanging the secret key.

Consider the figure below. It is an attack against the key exchange explained in the Diffie-Hellman Key Exchange task. The steps are as follows:

1. Alice and Bob agree on *q* and *g*. Anyone listening on the communication channel can read these two values, including the attacker, Mallory.
2. As she would normally do, Alice chooses a random variable *a*, calculates *A* ($A = (g^a) \bmod q$) and sends *A* to Bob. Mallory has been waiting for this step, and she has selected a random variable *m* and calculated the respective *M*. As soon as Mallory receives *A*, she sends *M* to Bob, pretending she is Alice.
3. Bob receives *M* thinking that Alice sent it. Bob has already picked a random variable *b* and calculated the respective *B*; he sends *B* to Alice. Similarly, Mallory intercepts the message, reads *B* and sends *M* to Alice instead.
4. Alice receives *M* and calculates $key = M^a \bmod q$.
5. Bob receives *M* and calculates $key = M^b \bmod q$.

Alice and Bob continue to communicate, thinking that they are communicating directly, unaware that they are communicating with Mallory, who can read and modify the messages before sending them to the intended recipient.

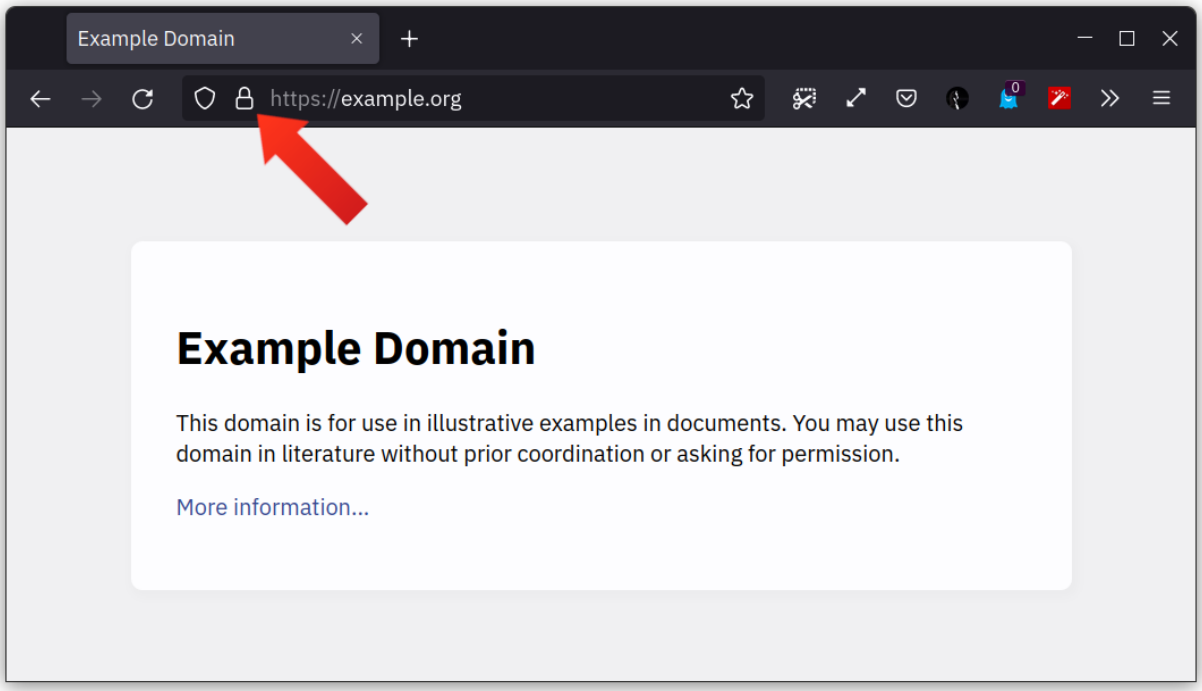


This susceptibility necessitates some mechanism that would allow us to confirm the other party's identity. This brings us to Public Key Infrastructure (PKI).

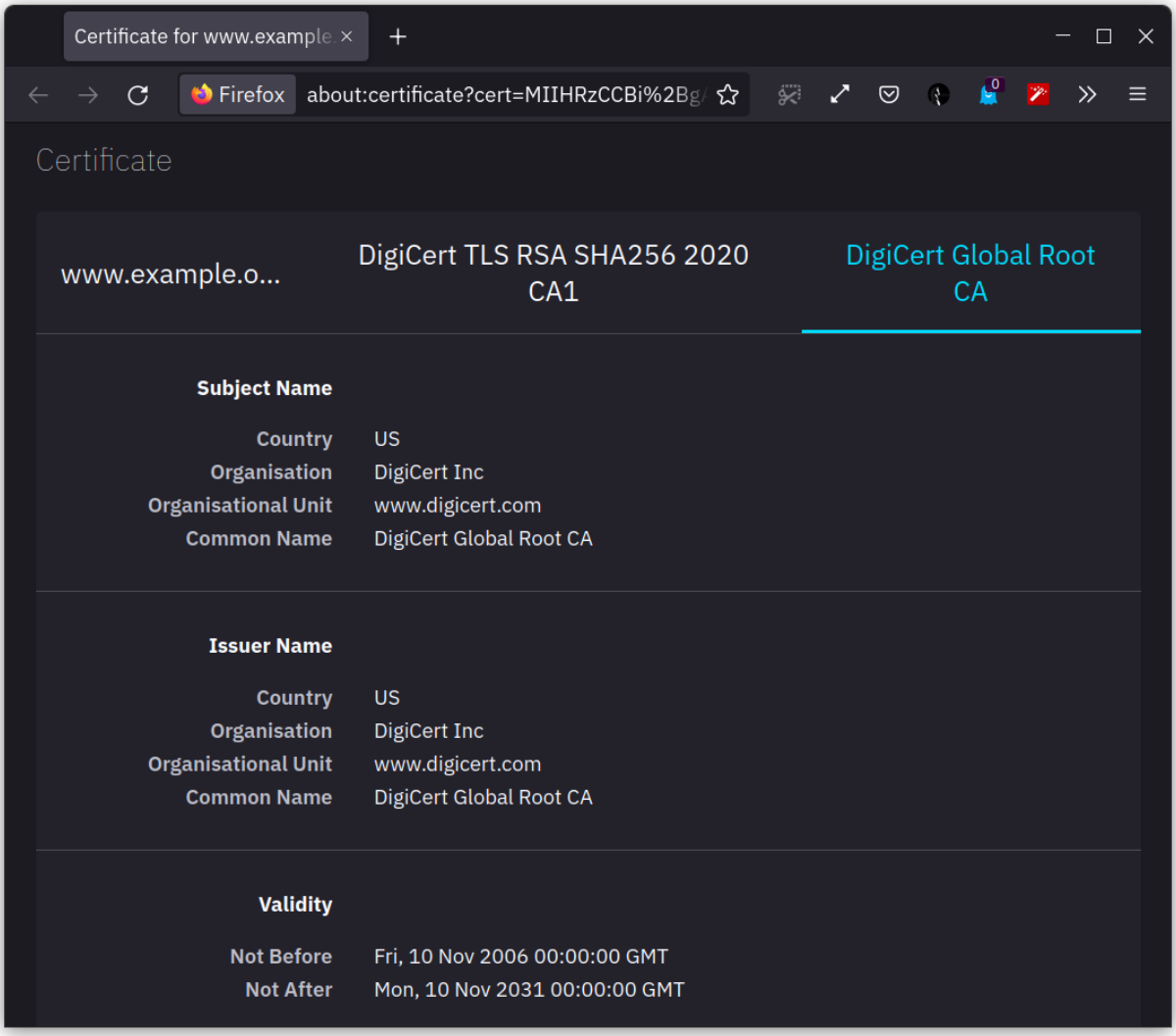
Consider the case where you are browsing the website example.org over HTTPS. How can you be confident that you are indeed communicating with the example.org server(s)? In other words, how can

you be sure that no man-in-the-middle intercepted the packets and altered them before they reached you? The answer lies in the website certificate.

The figure below shows the page we get when browsing example.org. Most browsers represent the encrypted connection with some kind of a lock icon. This lock icon indicates that the connection is secured over HTTPS with a valid certificate.



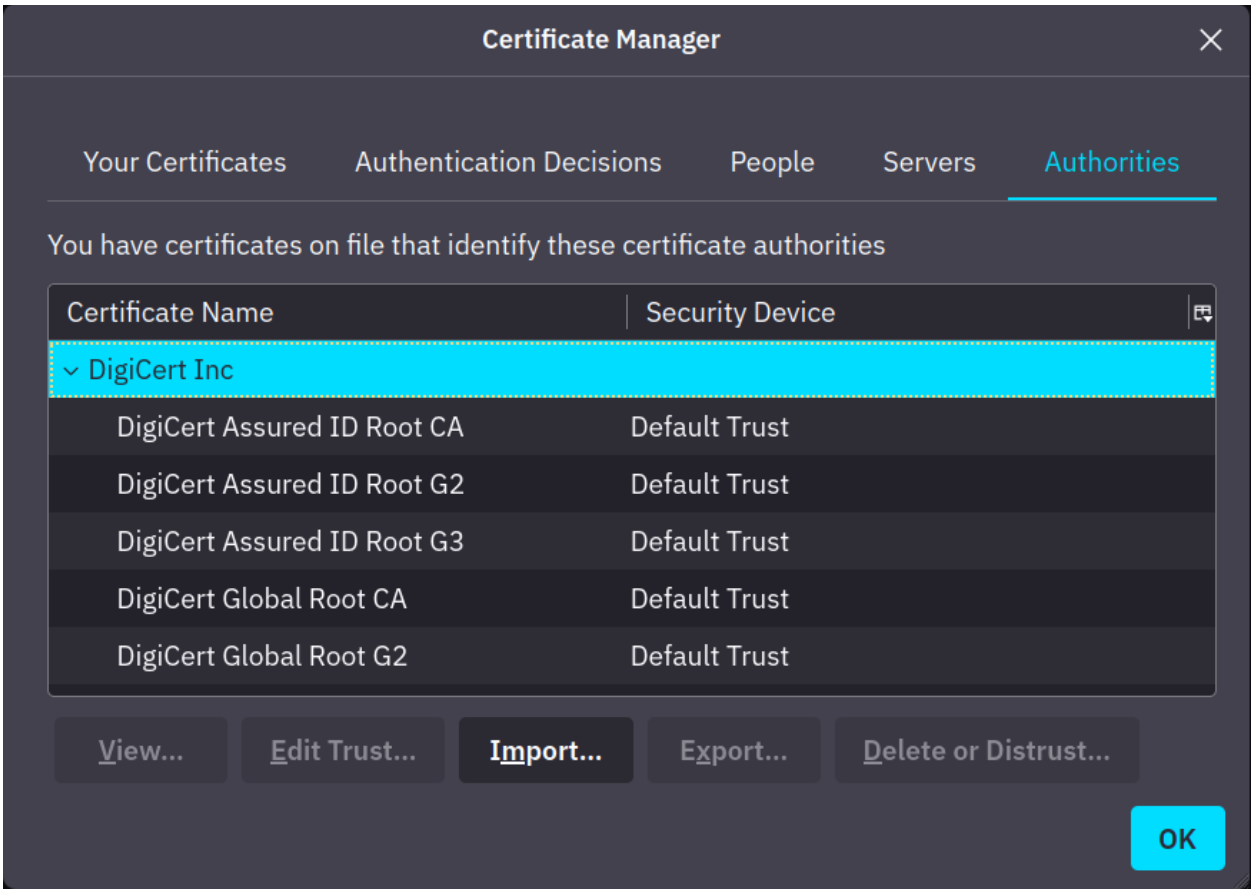
At the time of writing, example.org uses a certificate signed by DigiCert Inc., as shown in the figure below. In other words, DigiCert confirms that this certificate is valid (till a certain date).



For a certificate to get signed by a certificate authority, we need to:

1. Generate Certificate Signing Request (CSR): You create a certificate and send your public key to be signed by a third party.
2. Send your CSR to a Certificate Authority (CA): The purpose is for the CA to sign your certificate. The alternative and usually insecure solution would be to self-sign your certificate.

For this to work, the recipient should recognize and trust the CA that signed the certificate. And as we would expect, our browser trusts DigiCert Inc as a signing authority; otherwise, it would have issued a security warning instead of proceeding to the requested website.



Hashing

Hashing is an off-shoot of cryptography with very similar workings, but quite different requirements. In this article, we will look at how hashing helps us to securely store passwords, prove who wrote what, and even allow two people to talk openly about a secret without people listening in to know what it is!

Hash Functions

A hash function is an algorithm that maps data *the message* to an array of a fixed size (the *hash value* or *digest*).

The cool thing about hash functions are they are **one way**, that means that they are "impossible" to invert. This means that the only way to discover the message that was hashed would be a brute force search of all possible inputs.

Cryptographic Impossibility

In truth nothing is "impossible" in cryptography, given enough time and resources.

However, we have the concept of "Computationally Infeasible", IE it takes that much processing time and power to reverse the algorithm, that it may as well be impossible.

Characteristics of Hash functions

1. A Hash function should be deterministic, meaning that the input will always result in the same hash value.
2. Small changes to the input value will result in large changes to the Hashed value. This means it becomes impossible to correlate values between similar inputs and gain some insight into the message.
3. Collisions should be avoided. In hashing a collision is where two (or more) inputs give the same hashed value. In simple hash functions it is easy to get collisions, (although in some cases, such as storing data in a hash table this may not matter). While in modern cryptographic hashes it is extremely hard, or impossible :p, to find collisions.

Common Hashing Algorithms

Hash functions have evolved over time. Many have been proposed by developers, and some have been found to be insecure on modern hardware (like MD5 and SHA1).