3
Symmetric Cryptography

In this chapter, you will be introduced to the concepts, theory, and practical aspects of *symmetric cryptography*. We will focus more on the elements that are specifically relevant in the context of blockchain technology. We will provide you with concepts that are required to understand the material covered in later chapters.

You will also be introduced to applications of cryptographic algorithms so that you can gain hands-on experience in the practical implementation of cryptographic functions. For this, we'll use the OpenSSL command-line tool. Before starting with the theoretical foundations, we'll look at the installation of OpenSSL in the following section, so that you can do some practical work as you read through the conceptual material.

Working with the OpenSSL command line

On the Ubuntu Linux distribution, OpenSSL is usually already available. However, it can be installed using the following command:

```
$ sudo apt-get install openssl
```

Examples in this chapter have been developed using OpenSSL version 1.0.2g.

It is available at `https://packages.ubuntu.com/xenial/openssl`.

You are encouraged to use this specific version, as all examples in the chapter have been developed and tested with it. The OpenSSL version can be checked using the following command:

```
$ openssl version
```

You will see the following output:

```
OpenSSL 1.0.2g  1 Mar 2016
```

Now, you are all set to run the examples provided in this chapter. If you are running a version other than 1.0.2g, the examples may still work but that is not guaranteed, as older versions lack the features used in the examples and newer versions may not be backward compatible with version 1.0.2g.

In the sections that follow, the theoretical foundations of cryptography are first discussed and then a series of relevant practical experiments will be presented.

Introduction

Cryptography is the science of making information secure in the presence of adversaries. It does so under the assumption that limitless resources are available to adversaries. **Ciphers** are algorithms used to encrypt or decrypt data, so that if intercepted by an adversary, the data is meaningless to them without **decryption**, which requires a secret key.

Cryptography is primarily used to provide a confidentiality service. On its own, it cannot be considered a complete solution, rather it serves as a crucial building block within a more extensive security system to address a security problem. For example, securing a blockchain ecosystem requires many different cryptographic primitives, such as hash functions, symmetric key cryptography, digital signatures, and public key cryptography.

In addition to a confidentiality service, cryptography also provides other security services such as integrity, authentication (entity authentication and data origin authentication), and non-repudiation. Additionally, accountability is also provided, which is a requirement in many security systems.

Before discussing cryptography further, some mathematical terms and concepts need to be explained in order to build a foundation for fully understanding the material provided later in this chapter.

The next section serves as a basic introduction to these concepts. An explanation with proofs and relevant background for all of these terms would require somewhat involved mathematics, which is beyond the scope of this book. More details on these topics can be found in any standard number theory, algebra, or cryptography-specific book. For example, *A Course in Number Theory and Cryptography* by *Neal Koblitz* provides an excellent presentation of all relevant mathematical concepts.

Mathematics

As the subject of cryptography is based on mathematics, this section will introduce some basic concepts that will help you understand the concepts presented later in the chapter.

Set

A **set** is a collection of distinct objects, for example, $X = \{1, 2, 3, 4, 5\}$.

Group

A **group** is a commutative set with one operation that combines two elements of the set. The group operation is closed and associated with a defined identity element. Additionally, each element in the set has an inverse. **Closure** (closed) means that if, for example, elements *A* and *B* are in the set, then the resultant element after performing an operation on the elements is also in the set. **Associative** means that the grouping of elements does not affect the result of the operation.

Field

A **field** is a set that contains both additive and multiplicative groups. More precisely, all elements in the set form an additive and multiplicative group. It satisfies specific axioms for addition and multiplication. For all group operations, the **distributive law** is also applied. The law dictates that the same sum or product will be produced even if any of the terms or factors are reordered.

A finite field

A **finite field** is one with a finite set of elements. Also known as *Galois fields*, these structures are of particular importance in cryptography as they can be used to produce accurate and error-free results of arithmetic operations. For example, prime finite fields are used in **Elliptic Curve Cryptography** (**ECC**) to construct discrete logarithm problems.

Order

The **order** is the number of elements in a field. It is also known as the *cardinality* of the field.

An abelian group

An **abelian group** is formed when the operation on the elements of a set is commutative. The commutative law means that changing the order of the elements does not affect the result of the operation, for example, $A \ X \ B = B \ X \ A$.

Prime fields

A **prime field** is a finite one with a prime number of elements. It has specific rules for addition and multiplication, and each nonzero element in the field has an inverse. Addition and multiplication operations are performed modulo *p*, that is, prime.

Ring

If more than one operation can be defined over an abelian group, that group becomes a **ring**. There are also specific properties that need to be satisfied. A ring must have closure and associative and distributive properties.

A cyclic group

A **cyclic group** is a type of group that can be generated by a single element called the *group generator*.

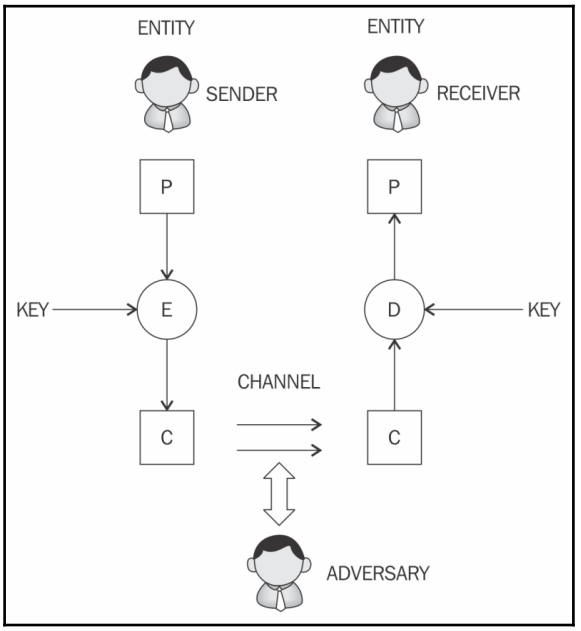
Modular arithmetic

Also known as clock arithmetic, numbers in modular arithmetic wrap around when they reach a certain fixed number. This fixed number is a positive number called **modulus**, and all operations are performed concerning this fixed number. Analogous to a clock, there are numbers from 1 to 12. When it reaches 12, the number 1 starts again. In other words, this type of arithmetic deals with the remainders after the division operation. For example, 50 mod 11 is 6 because 50 / 11 leaves a remainder of 6.

This completes a basic introduction to some mathematical concepts involved in cryptography. In the next section, you will be introduced to cryptography concepts.

Cryptography

A generic cryptography model is shown in the following diagram:



A model of the generic encryption and decryption model

In the preceding diagram, **P**, **E**, **C**, and **D** represent plaintext, encryption, ciphertext, and decryption, respectively. Also based on this model, explanations of concepts such as entity, sender, receiver, adversary, key, and channel follow:

- **Entity**: Either a person or system that sends, receives, or performs operations on data
- **Sender**: This is an entity that transmits the data
- **Receiver**: This is an entity that takes delivery of the data
- **Adversary**: This is an entity that tries to circumvent the security service
- **Key**: A key is data that is used to encrypt or decrypt other data
- **Channel**: Channel provides a medium of communication between entities

Next, we will describe the cryptography services mentioned earlier in the chapter in greater detail.

Confidentiality

Confidentiality is the assurance that information is only available to authorized entities.

Integrity

Integrity is the assurance that information is modifiable only by authorized entities.

Authentication

Authentication provides assurance about the identity of an entity or the validity of a message.

There are two types of authentication mechanisms, namely entity authentication and data origin authentication, which are discussed in the following section.

Entity authentication

Entity authentication is the assurance that an entity is currently involved and active in a communication session. Traditionally, users are issued a username and password that is used to gain access to the various platforms with which they are working. This practice is known as **single-factor authentication**, as there is only one factor involved, namely, *something you know*, that is, the password and username. This type of authentication is not very secure for a variety of reasons, for example, password leakage; therefore, additional factors are now commonly used to provide better security. The use of additional techniques for user identification is known as **multifactor authentication** (or two-factor authentication if only two methods are used).

Various authentication factors are described here:

- The first factor is *something you have*, such as a hardware token or a smart card. In this case, a user can use a hardware token in addition to login credentials to gain access to a system. This mechanism protects the user by requiring two factors of authentication. A user who has access to the hardware token and knows the login credentials will be able to access the system. Both factors should be available to gain access to the system, thus making this method a two-factor authentication mechanism. In case if the hardware token is lost, on its own it won't be of any use unless, *something you know*, the login password is also used in conjunction with the hardware token.
- The second factor is *something you are*, which uses biometric features to identify the user. With this method, a user's fingerprint, retina, iris, or hand geometry is used to provide an additional factor for authentication. This way, it can be ensured that the user was indeed present during the authentication process, as biometric features are unique to every individual. However, careful implementation is required to guarantee a high level of security, as some research has suggested that biometric systems can be circumvented under specific conditions.

Data origin authentication

Also known as *message authentication*, **data origin authentication** is an assurance that the source of the information is indeed verified. Data origin authentication guarantees data integrity because if a source is corroborated, then the data must not have been altered. Various methods, such as **Message Authentication Codes** (**MACs**) and digital signatures are most commonly used. These terms will be explained in detail later in the chapter.

Non-repudiation

Non-repudiation is the assurance that an entity cannot deny a previous commitment or action by providing incontrovertible evidence. It is a security service that offers definitive proof that a particular activity has occurred. This property is essential in debatable situations whereby an entity has denied the actions performed, for example, placement of an order on an e-commerce system. This service produces cryptographic evidence in electronic transactions so that in case of disputes, it can be used as a confirmation of an action.

Non-repudiation has been an active research area for many years. Disputes in electronic transactions are a common issue, and there is a need to address them to increase the confidence level of consumers in such services.

The non-repudiation protocol usually runs in a communication network, and it is used to provide evidence that an action has been taken by an entity (originator or recipient) on the network. In this context, there are two communications models that can be used to transfer messages from originator *A* to recipient *B*:

- A message is sent directly from originator *A* to recipient *B*.
- A message is sent to a delivery agent from originator *A*, which then delivers the message to recipient *B*.

The primary requirements of a non-repudiation protocol are fairness, effectiveness, and timeliness. In many scenarios, there are multiple participants involved in a transaction, as opposed to only two parties. For example, in electronic trading systems, there can be many entities, such as clearing agents, brokers, and traders that can be involved in a single transaction. In this case, two-party non-repudiation protocols are not appropriate. To address this problem, **Multi-Party Non-Repudiation** (**MPNR**) protocols have been developed.

Accountability

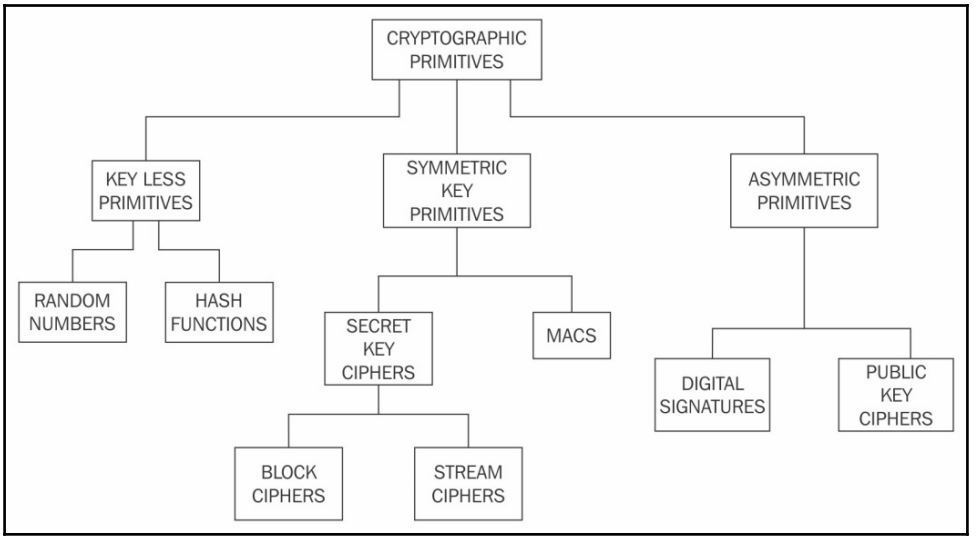
Accountability is the assurance which states that actions affecting security can be traced back to the responsible party. This is usually provided by logging and audit mechanisms in systems where a detailed audit is required due to the nature of the business, for example, in electronic trading systems. Detailed logs are vital to trace an entity's actions, such as when a trade is placed in an audit record with the date and timestamp and the entity's identity is generated and saved in the log file. This log file can optionally be encrypted and be part of the database or a standalone ASCII text log file on a system.

In order to provide all of the services discussed earlier, different cryptographic primitives are used that are presented in the next section.

Cryptographic primitives

Cryptographic primitives are the basic building blocks of a security protocol or system. In the following section, you are introduced to cryptographic algorithms that are essential for building secure protocols and systems. A **security protocol** is a set of steps taken to achieve the required security goals by utilizing appropriate security mechanisms. Various types of security protocols are in use, such as authentication protocols, non-repudiation protocols, and key management protocols.

The taxonomy of cryptographic primitives can be visualized as shown here:



Cryptographic primitives

As shown in the cryptographic primitives taxonomy diagram, cryptography is mainly divided into two categories: *symmetric cryptography* and *asymmetric cryptography*.

These primitives are discussed further in the next section.

Symmetric cryptography

Symmetric cryptography refers to a type of cryptography where the key that is used to encrypt the data is the same one that is used for decrypting the data. Thus, it is also known as **shared key cryptography**. The key must be established or agreed upon before the data exchange occurs between the communicating parties. This is the reason it is also called **secret key cryptography**.

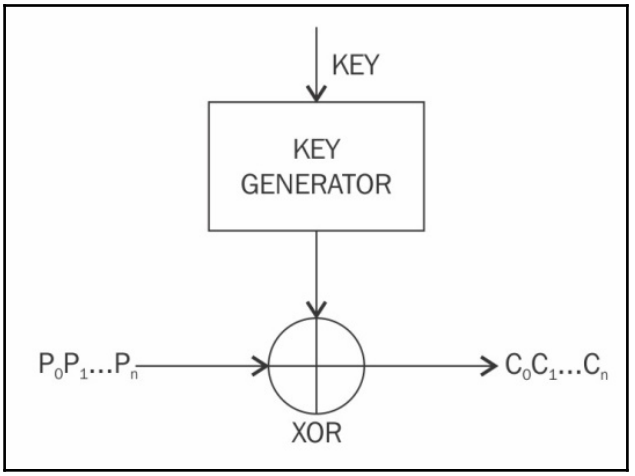
There are two types of symmetric ciphers: *stream ciphers* and *block ciphers*. **Data Encryption Standard** (**DES**) and **Advanced Encryption Standard** (**AES**) are typical examples of block ciphers, whereas RC4 and A5 are commonly used stream ciphers.

Stream ciphers

Stream ciphers are encryption algorithms that apply encryption algorithms on a bit-by-bit basis (one bit at a time) to plaintext using a keystream. There are two types of stream ciphers: *synchronous stream ciphers* and *asynchronous stream ciphers*:

- **Synchronous stream ciphers** are those where the keystream is dependent only on the key
- **Asynchronous stream ciphers** have a keystream that is also dependent on the encrypted data

In stream ciphers, encryption and decryption are the same function because they are simple modulo-2 additions or XOR operations. The fundamental requirement in stream ciphers is the security and randomness of keystreams. Various techniques ranging from pseudorandom number generators to true random number generators implemented in hardware have been developed to generate random numbers, and it is vital that all key generators be cryptographically secure:



Operation of a stream cipher

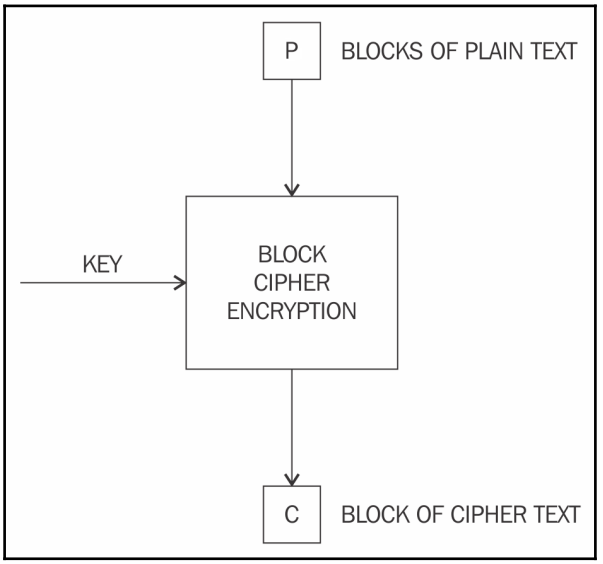
Block ciphers

Block ciphers are encryption algorithms that break up the text to be encrypted (plaintext) into blocks of a fixed length and apply the encryption block-by-block. Block ciphers are generally built using a design strategy known as a **Feistel cipher**. Recent block ciphers, such as AES (Rijndael) have been built using a combination of substitution and permutation called a **Substitution-Permutation Network** (**SPN**).

Feistel ciphers are based on the Feistel network, which is a structure developed by Horst Feistel. This structure is based on the idea of combining multiple rounds of repeated operations to achieve desirable cryptographic properties known as *confusion* and *diffusion*. Feistel networks operate by dividing data into two blocks (left and right) and processing these blocks via keyed *round functions* in iterations to provide sufficient pseudorandom permutation.

Confusion makes the relationship between the encrypted text and plaintext complex. This is achieved by substitution. In practice, *A* in plaintext is replaced by *X* in encrypted text. In modern cryptographic algorithms, substitution is performed using lookup tables called *S-boxes*. The diffusion property spreads the plaintext statistically over the encrypted data. This ensures that even if a single bit is changed in the input text, it results in changing at least half (on average) of the bits in the ciphertext. Confusion is required to make finding the encryption key very difficult, even if many encrypted and decrypted data pairs are created using the same key. In practice, this is achieved by transposition or permutation.

A key advantage of using a Feistel cipher is that encryption and decryption operations are almost identical and only require a reversal of the encryption process to achieve decryption. DES is a prime example of Feistel-based ciphers:



Simplified operation of a block cipher

Various modes of operation for block ciphers are **Electronic Code Book** (**ECB**), **Cipher Block Chaining** (**CBC**), **Output Feedback** (**OFB**) mode, and **Counter** (**CTR**) mode. These modes are used to specify the way in which an encryption function is applied to the plaintext. Some of these modes of block cipher encryption are introduced here.

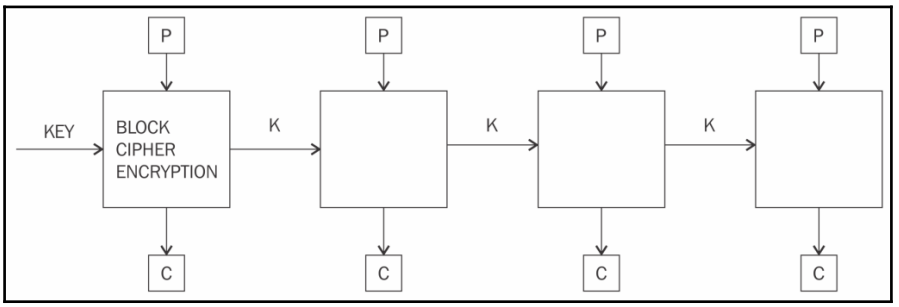
Block encryption mode

In **block encryption mode**, the plaintext is divided into blocks of fixed length depending on the type of cipher used. Then the encryption function is applied to each block.

The most common block encryption modes are briefly discussed in the following subsections.

Electronic Code Book

Electronic Code Book (**ECB**) is a basic mode of operation in which the encrypted data is produced as a result of applying the encryption algorithm one-by-one to each block of plaintext. This is the most straightforward mode, but it should not be used in practice as it is insecure and can reveal information:

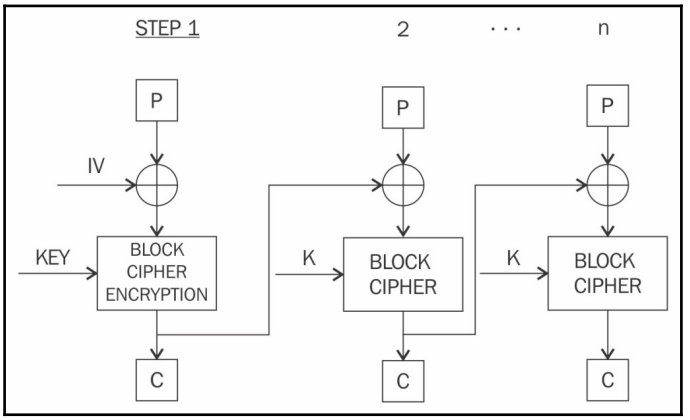


Electronic Code Book mode for block ciphers

The preceding diagram shows that we have plaintext **P** provided as an input to the block cipher encryption function, along with a key **KEY** and ciphertext **C** is produced as output.

Cipher Block Chaining

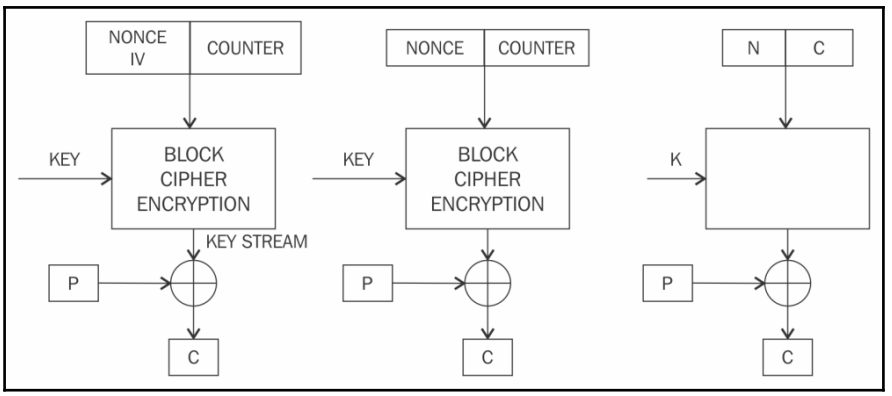
In **Cipher Block Chaining** (**CBC**) mode, each block of plaintext is XOR'd with the previously-encrypted block. CBC mode uses the **Initialization Vector** (**IV**) to encrypt the first block. It is recommended that the IV be randomly chosen:



Cipher block chaining mode

Counter mode

The **Counter** (**CTR**) mode effectively uses a block cipher as a stream cipher. In this case, a unique nonce is supplied that is concatenated with the counter value to produce a **keystream**:



Counter mode

There are other modes, such as **Cipher Feedback** (**CFB**) mode, **Galois Counte**r (**GCM**) mode, and **Output Feedback** (**OFB**) mode, which are also used in various scenarios.

Keystream generation mode

In **keystream generation mode**, the encryption function generates a keystream that is then XOR'd with the plaintext stream to achieve encryption.

Message authentication mode

In **message authentication mode**, a **Message Authentication Code** (**MAC**) results from an encryption function. The MAC is a cryptographic checksum that provides an integrity service. The most common method to generate a MAC using block ciphers is CBC-MAC, where a part of the last block of the chain is used as a MAC. For example, a MAC can be used to ensure that if a message is modified by an unauthorized entity. This can be achieved by encrypting the message with a key using the MAC function. The resultant message and MAC of the message once received by the receiver can be checked by encrypting the message received again by the key and comparing it with the MAC received from the sender. If they both match, then the message has not modified by unauthorized user thus integrity service is provided. If they both don't match, then it means that message is modified by unauthorized entity during the transmission.

Cryptographic hash mode

Hash functions are primarily used to compress a message to a fixed-length digest. In **cryptographic hash mode**, block ciphers are used as a compression function to produce a hash of plaintext.

With this, we have now concluded the introduction to block ciphers. In the following section, you will be introduced to the design and mechanism of a currently market-dominant block cipher known as AES.

Before discussing AES, however, some history is presented about the **Data Encryption Standard** (**DES**) that led to the development of the new AES standard.

Data Encryption Standard

The **Data Encryption Standard** (**DES**) was introduced by the U.S. **National Institute of Standards and Technology** (**NIST**) as a standard algorithm for encryption, and it was in widespread use during the 1980s and 1990s. However, it did not prove to be very resistant to brute force attacks, due to advances in technology and cryptography research. In July 1998, for example, the **Electronic Frontier Foundation** (**EFF**) broke DES using a special-purpose machine called EFF DES cracker (or *Deep Crack*).

DES uses a key of only 56 bits, which raised some concerns. This problem was addressed with the introduction of **Triple DES** (**3DES**), which proposed the use of a 168-bit key by means of three 56-bit keys and the same number of executions of the DES algorithm, thus making brute force attacks almost impossible. However, other limitations, such as slow performance and 64-bit block size, were not desirable.

Advanced Encryption Standard

In 2001, after an open competition, an encryption algorithm named Rijndael invented by cryptographers Joan Daemen and Vincent Rijmen was standardized as **Advanced Encryption Standard** (**AES**) with minor modifications by NIST. So far, no attack has been found against AES that is more effective than the brute-force method. The original version of Rijndael permits different key and block sizes of 128-bit, 192-bit, and 256-bits. In the AES standard, however, only a 128-bit block size is allowed. However, key sizes of 128-bit, 192-bit, and 256-bit are permissible.

How AES works

During AES algorithm processing, a 4 x 4 array of bytes known as the **state** is modified using multiple rounds. Full encryption requires 10 to 14 rounds, depending on the size of the key. The following table shows the key sizes and the required number of rounds:

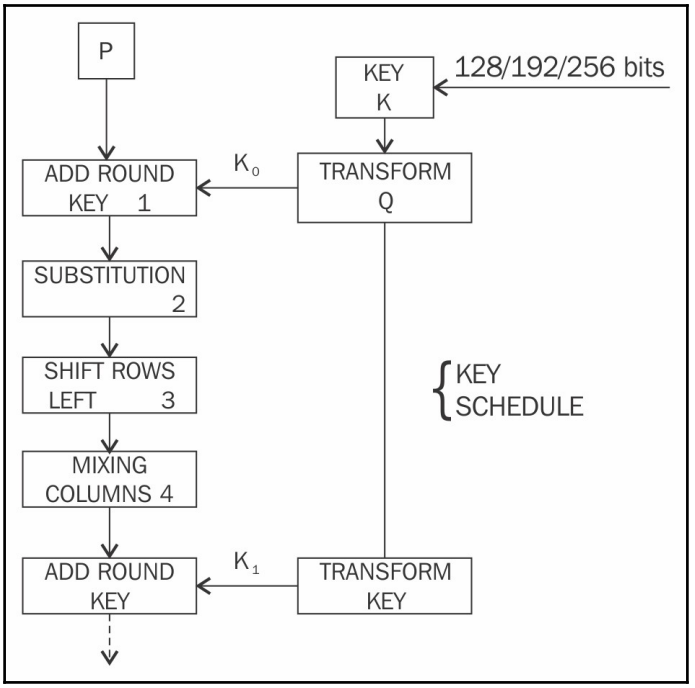
Key size	Number of rounds required
128-bit	10 rounds
192-bit	12 rounds
256-bit	14 rounds

Once the state is initialized with the input to the cipher, four operations are performed in four stages to encrypt the input. These stages are: `AddRoundKey`, `SubBytes`, `ShiftRows`, and `MixColumns`:

1. In the `AddRoundKey` step, the state array is XOR'd with a subkey, which is derived from the master key
2. `SubBytes` is the substitution step where a lookup table (S-box) is used to replace all bytes of the state array
3. The `ShiftRows` step is used to shift each row to the left, except for the first one, in the state array to the left in a cyclic and incremental manner
4. Finally, all bytes are mixed in the `MixColumns` step in a linear fashion, column-wise

The preceding steps describe one round of AES.

In the final round (either 10, 12, or 14, depending on the key size), stage 4 is replaced with `AddRoundKey` to ensure that the first three steps cannot be simply reversed:



AES block diagram, showing the first round of AES encryption. In the last round, the mixing step is not performed

Various cryptocurrency wallets use AES encryption to encrypt locally-stored data. Especially in Bitcoin wallet, AES-256 in the CBC mode is used.

Here's an OpenSSL example of how to encrypt and decrypt using AES:

```
$ openssl enc -aes-256-cbc -in message.txt -out message.bin
enter aes-256-cbc encryption password:
Verifying - enter aes-256-cbc encryption password:
$ ls -ltr
-rw-rw-r-- 1 drequinox drequinox 14 Sep 21 05:54 message.txt
-rw-rw-r-- 1 drequinox drequinox 32 Sep 21 05:57 message.bin
$ cat message.bin
```

The following are the contents of the `message.bin` file:

```
Salted__w    s_ÿ h~?                    :~/Crypt$
:~/Crypt$
```

Note that `message.bin` is a binary file. Sometimes, it is desirable to encode this binary file in a text format for compatibility/interoperability reasons. The following command can be used to do just that:

```
$ openssl enc -base64 -in message.bin -out message.b64
$ ls -ltr
-rw-rw-r-- 1 drequinox drequinox 14 Sep 21 05:54 message.txt
-rw-rw-r-- 1 drequinox drequinox 32 Sep 21 05:57 message.bin
-rw-rw-r-- 1 drequinox drequinox 45 Sep 21 06:00 message.b64
$ cat message.b64
U2FsdGVkX193uByIcwZf0Z7J1at+4L+Fj8/uzeDAtJE=
```

In order to decrypt an AES-encrypted file, the following commands can be used. An example of `message.bin` from a previous example is used:

```
$ openssl enc -d -aes-256-cbc -in message.bin -out message.dec
enter aes-256-cbc decryption password:
$ ls -ltr
-rw-rw-r-- 1 drequinox drequinox 14 Sep 21 05:54 message.txt
-rw-rw-r-- 1 drequinox drequinox 32 Sep 21 05:57 message.bin
-rw-rw-r-- 1 drequinox drequinox 45 Sep 21 06:00 message.b64
-rw-rw-r-- 1 drequinox drequinox 14 Sep 21 06:06 message.dec
$ cat message.dec
Datatoencrypt
```

Astute readers will have noticed that no IV has been provided, even though it's required in all block encryption modes of operation except ECB. The reason for this is that OpenSSL automatically derives the IV from the given password. Users can specify the IV using the following switch:

```
-K/-iv      , (Initialization Vector) should be provided in Hex.
```

In order to decode from base64, the following commands are used. Follow the `message.b64` file from the previous example:

```
$ openssl enc -d -base64 -in message.b64 -out message.ptx
$ ls -ltr
-rw-rw-r-- 1 drequinox drequinox 14 Sep 21 05:54 message.txt
-rw-rw-r-- 1 drequinox drequinox 32 Sep 21 05:57 message.bin
-rw-rw-r-- 1 drequinox drequinox 45 Sep 21 06:00 message.b64
-rw-rw-r-- 1 drequinox drequinox 14 Sep 21 06:06 message.dec
-rw-rw-r-- 1 drequinox drequinox 32 Sep 21 06:16 message.ptx
$ cat message.ptx
```

The following are the contents of the `message.ptx` file:

```
:~/Crypt$ cat message.ptx
Salted__w    s_ў  h~?                    :~/Crypt$
```

There are many types of ciphers that are supported in OpenSSL. You can explore these options based on the preceding examples. A list of supported cipher types is shown in the following screenshot:

```
Cipher Types
-aes-128-cbc               -aes-128-ccm               -aes-128-cfb
-aes-128-cfb1              -aes-128-cfb8              -aes-128-ctr
-aes-128-ecb               -aes-128-ofb               -aes-192-cbc
-aes-192-ccm               -aes-192-cfb               -aes-192-cfb1
-aes-192-cfb8              -aes-192-ctr               -aes-192-ecb
-aes-192-ofb               -aes-256-cbc               -aes-256-ccm
-aes-256-cfb               -aes-256-cfb1              -aes-256-cfb8
-aes-256-ctr               -aes-256-ecb               -aes-256-ofb
-aes128                    -aes192                    -aes256
-bf                        -bf-cbc                    -bf-cfb
-bf-ecb                    -bf-ofb                    -blowfish
-camellia-128-cbc          -camellia-128-cfb          -camellia-128-cfb1
-camellia-128-cfb8         -camellia-128-ecb          -camellia-128-ofb
-camellia-192-cbc          -camellia-192-cfb          -camellia-192-cfb1
-camellia-192-cfb8         -camellia-192-ecb          -camellia-192-ofb
-camellia-256-cbc          -camellia-256-cfb          -camellia-256-cfb1
-camellia-256-cfb8         -camellia-256-ecb          -camellia-256-ofb
-camellia128               -camellia192               -camellia256
-cast                      -cast-cbc                  -cast5-cbc
-cast5-cfb                 -cast5-ecb                 -cast5-ofb
-des                       -des-cbc                   -des-cfb
-des-cfb1                  -des-cfb8                  -des-ecb
-des-ede                   -des-ede-cbc               -des-ede-cfb
-des-ede-ofb               -des-ede3                  -des-ede3-cbc
-des-ede3-cfb              -des-ede3-cfb1             -des-ede3-cfb8
-des-ede3-ofb              -des-ofb                   -des3
-desx                      -desx-cbc                  -id-aes128-CCM
-id-aes128-wrap            -id-aes192-CCM             -id-aes192-wrap
-id-aes256-CCM             -id-aes256-wrap            -id-smime-alg-CMS3DESwrap
-idea                      -idea-cbc                  -idea-cfb
-idea-ecb                  -idea-ofb                  -rc2
-rc2-40-cbc                -rc2-64-cbc                -rc2-cbc
-rc2-cfb                   -rc2-ecb                   -rc2-ofb
-rc4                       -rc4-40                    -seed
-seed-cbc                  -seed-cfb                  -seed-ecb
-seed-ofb
```

Screenshot displaying rich library options available in OpenSSL

OpenSSL tool can be used to experiment with all the ciphers shown in the screenshot.

Summary

In this chapter, we introduced you to symmetric key cryptography. We started with basic mathematical definitions and cryptographic primitives. After this, we introduced you to the concepts of stream and block ciphers along with working modes of block ciphers. Moreover, we introduced you to the practical exercises using OpenSSL to complement the theoretical concepts.

In the next chapter, we will present public key cryptography, which is used extensively in blockchain technology and has very interesting properties.