# A1: Symmetric Encryption, Block and Stream Ciphers, Modes of Operation

**Goal:** Learn the fundamentals of symmetric encryption, focusing on the differences between block ciphers and stream ciphers and their respective modes of operation. By the end of this lesson, you should have a foundational understanding of symmetric encryption concepts that will help you understand encryption mechanisms and apply encryption algorithms effectively.

**Table of contents:**



## A1.T.1: Fundamentals of Symmetric Encryption

**Objective:** Understand how symmetric encryption works and familiarize yourself with common symmetric encryption algorithms.

**Tasks:**

- Learn about Symmetric Encryption:
  - Understand the concept of symmetric encryption, where the same key is used for both encryption and decryption.
    - Watch Symmetric Encryption explained by @SecPrivAca [youtube.com].
    - Watch Symmetric Encryption Ciphers by @professormesser [youtube.com].
    - Watch Symmetric Cryptosystems - Applied Cryptography by @Udacity [youtube.com].
  - Explore how symmetric encryption is used in various applications, including file encryption, secure communication, and more.
    - Read Symmetric Key Cryptography by @cipher_encoder [geeksforgeeks.org]

**Key Concepts:**

- Symmetric Encryption: Encryption where the same key is used for both encryption and decryption.

## A1.T.2: Block Ciphers

**Objective:** Understand how block ciphers function, and explore different modes of operation.

**Tasks:**

- Learn about Block Ciphers:
  - Learn about block ciphers, where data is divided into fixed-size blocks, and each block is encrypted separately.

    - Study examples of block cipher algorithms such as AES and DES.
        - Watch AES Explained (Advanced Encryption Standard) by @Computerphile [youtube.com]
        - Watch One Encryption Standard to Rule Them All! by @Computerphile [youtube.com]

**Key Concepts:**

- Block Ciphers: Symmetric encryption technique that processes data in fixed-size blocks, such as AES and DES.

## A1.T.3: Stream Ciphers

**Objective:** Understand how stream ciphers function and when they are best used.

**Tasks:**

- Stream Ciphers:
    - Understand the concept of stream ciphers, where data is encrypted one bit or byte at a time.
    - Explore examples of stream cipher algorithms such as RC4 and Chacha20.
        - Watch Basics of Cryptology – Part 5 (Modern Cryptography – Stream Ciphers – RC4) by @CryptographyForEverybody [youtube.com]
        - Watch Chacha Cipher by @Computerphile [youtube.com]
    - Discuss the advantages and disadvantages of stream ciphers compared to block ciphers, including speed and security considerations.

**Key Concepts:**

- Stream Ciphers: Symmetric encryption technique that processes data one bit or byte at a time, such as RC4 and Chacha20.
- Use Cases: Scenarios where stream ciphers are preferred, such as real-time encryption in streaming applications.

## A1.T.4: Modes of Operation

**Objective:** Understand how Modes of Operation alter the behaviour of a cipher and effectively affect the security of the encrypted message.

**Tasks:**

- Modes of Operation:
    - Understand the different modes of operation (such as ECB, CBC, CFB, OFB, CTR and GCM) and their respective use cases and vulnerabilities.
        - Watch Modes of Operation by @Computerphile [youtube.com]
        - Watch AES GCM (Advanced Encryption Standard in Galois Counter Mode) by @Computerphile [youtube.com]

**Key Concepts:**

- Modes of Operation: Variations in block cipher encryption methods, including ECB, CBC, CFB, OFB, and their effects on security.

## A1.P.1: Implementing Block Ciphers

**Objective:** Practice implementing block ciphers using a tool called `openssl` and familiarize yourself with the modes of operation.

**Prepare:**

- Create a working directory for this lesson:

```
$ mkdir A1P1/ && cd A1P1/
```

- Create a text file with following content inside

```
$ vim plaintext.txt
```

```
CONFIDENTIAL

Hello,
This message is going to be encrypted.

If you encrypt this message with AES with ECB mode of operation, you should be
able to see why ECB is a security risk, since some parts of the encrypted
message can be repeated.

Example:

THECAKEISALIE!!!THECAKEISALIE!!!THECAKEISALIE!!!THECAKEISALIE!!!

Goodbye.

CONDIFENTIAL
```

**Tasks:**

- Encrypting a File with AES-128-CBC:
  - Create a symmetric 16-byte key for encryption:

```
$ openssl rand -out A1P1.key 16
```

  - Use `openssl` to encrypt a file using the AES-256-CBC mode:

```
$ openssl enc -aes-128-cbc -nosalt -pass file:./A1P1.key -in plaintext.txt
-out encrypted.bin -v
```

  - Decrypt the file using the same key and algorithm:

```
$ openssl enc -aes-128-cbc -d -nosalt -in encrypted.bin -out decrypted.txt
-pass file:./A1P1.key -v
```

- Compare the original file with the decrypted one:

```
$ diff plaintext.txt decrypted.txt

$ sha256sum *.txt
```

- Experimenting with Different Modes:
  - Encrypt the same file using different modes like ECB and CBC, and compare the results.

```
$ openssl enc -aes-128-ecb -nosalt -pass file:./A1P1.key -in plaintext.txt
-out encrypted_aes_ecb.bin -v
$ openssl enc -aes-128-cbc -nosalt -pass file:./A1P1.key -in plaintext.txt
-out encrypted_aes_cbc.bin -v
```

  - Observe the patterns in ECB mode and understand why CBC is generally preferred for most applications.

```
$ hexdump -vC encrypted_aes_ecb.bin
$ hexdump -vC encrypted_aes_cbc.bin
```

- Decrypting and Verifying:
  - Decrypt both files and ensure that the decrypted content matches the original plaintext.

```
$ openssl enc -d -aes-128-ecb -nosalt -in encrypted_aes_ecb.bin -out
decrypted_ecb.txt -pass file:./A1P1.key -v
$ openssl enc -d -aes-128-cbc -nosalt -in encrypted_aes_cbc.bin -out
decrypted_cbc.txt -pass file:./A1P1.key -v
```

**Key Concepts:**

- Block Cipher Implementation: Using `openssl` to encrypt and decrypt files with block ciphers like AES.
- Modes of Operation: Experimenting with ECB and CBC modes to understand their differences and impacts on security.

## A1.P.2: Implementing Stream Ciphers

**Objective:** Practice implementing stream ciphers using `openssl` and understand their real-time encryption capabilities.

**Prepare:**

- Create a working directory for this lesson:

```
$ mkdir A1P2/ && cd A1P2/
```

- Create a simple text file with custom content inside:

```
$ vim plaintext.txt
```

- Generate a large file (8 GB) filled with random content:

```
$ dd if=/dev/zero of=large_file.bin bs=1G count=8
```

- Create a symmetric key:

```
$ openssl rand -out A1P2.key 128
```

**Tasks:**

- Encrypting Data with Chacha20 (Stream Cipher):
  - Use `openssl` to encrypt a message using the Chacha20 algorithm.

```
$ openssl enc -chacha20 -in plaintext.txt -out encrypted_chacha20.bin -
pass file:./A1P2.key -salt -v
```

  - Decrypt the message using the same password.

```
$ openssl enc -chacha20 -d -in encrypted_chacha20.bin -out decrypted.txt -
pass file:./A1P2.key -salt -v
```

- Comparing Stream Cipher Speeds:
  - Encrypt large files using both Chacha20 (stream cipher) and AES-256-CBC (block cipher) and compare the encryption speeds.

```
$ time openssl enc -chacha20 -in plaintext.txt -out encrypted_chacha20.bin
-pass file:./A1P2.key -salt -v # Encrypting a small file with Chacha20

$ time openssl enc -aes-256-cbc -in plaintext.txt -out
encrypted_aes_cbc.bin -pass file:./A1P2.key -salt -v # Encrypting a small
file with AES-256-CBC

$ time openssl enc -chacha20 -salt -in large_file.bin -out
encrypted_chacha20.bin -pass file:./A1P2.key -salt -v # Encrypting a large
file with Chacha20

$ time openssl enc -aes-256-cbc -salt -in large_file.bin -out
```

```
encrypted_aes_cbc.bin -pass file:./A1P2.key -salt -v # Encrypting a large file with AES-256-CBC
```

    - Note the difference in processing times and discuss the trade-offs.

**Key Concepts:**

- Stream Cipher Implementation: Using `openssl` to encrypt and decrypt data with stream ciphers like Chacha20.
- Performance Considerations: Understanding the speed differences between stream and block ciphers and when to use each.

## A1.P.3: Working with Modes of Operation

**Objective:** Compare different outputs based on selected mode of operation of the cipher.

**Prepare:**

- Create a working directory for this lesson:

```
$ mkdir A1P3/ && cd A1P3/
```

- Download Tux.png from Wikipedia using `curl`:

```
$ curl -o Tux.png https://upload.wikimedia.org/wikipedia/commons/a/af/Tux.png
```

- Download ImageMagick library to covert the Tux.png from PNG to PPM format later:

```
$ sudo apt-get install imagemagick
```

- Create a symmetric key:

```
$ openssl rand -out A1P3.key 128
```

**Tasks:**

- Convert the `Tux.png` into a `Tux.ppm` using the `convert` command from ImageMagick library

```
$ convert Tux.png Tux.ppm
```

- Encrypt the image contents using AES-128 and ECB as a Mode of Operation:

    - Save the header into a file:

```
```

```
$ head -n 3 Tux.ppm > _tux_header
```

- Save the body into a file (this part will get encrypted):

```
$ tail -n +4 Tux.ppm > _tux_body
```

- Encrypt the body using AES-128 and ECB as a Mode of Operation:

```
$ openssl enc -aes-128-ecb -nosalt -pass file:./A1P3.key -in _tux_body -
out _tux_body_ecb.bin -v
```

- Asseble the encrypted contents into a picture and convert it back into PNG format:

```
$ cat _tux_header _tux_body_ecb.bin > Tux_ecb.ppm

$ convert Tux_ecb.ppm Tux_ecb.png
```

- Open and examine the picture:

```
$ open Tux_ecb.png
```

- Repeat the steps for different Modes of Operation (such as CBC, OFB, CTR, GCM).
  - You can also create a shell script that will do everything for you:

```
#!/bin/bash

# Convert PNG to PPM
echo "Converting Tux.png to PPM format..."
convert Tux.png Tux.ppm

# Extract header and body from the PPM file
echo "Extracting header and body from PPM file..."
head -n 3 Tux.ppm > _tux_header
tail -n +4 Tux.ppm > _tux_body

# Encrypt the body using AES in ECB mode
echo "Encrypting with AES-128-ECB..."
openssl enc -aes-128-ecb -nosalt -pass file:./A1P3.key -in _tux_body -out
_tux_body_ecb.bin -v -p
cat _tux_header _tux_body_ecb.bin > Tux_ecb.ppm
convert Tux_ecb.ppm Tux_ecb.png
echo "ECB encryption complete. Tux_ecb.png created."

# Encrypt the body using AES in CBC mode
```

```
echo "Encrypting with AES-128-CBC..."
openssl enc -aes-128-cbc -nosalt -pass file:./A1P3.key -in _tux_body -out
_tux_body_cbc.bin -v -p
cat _tux_header _tux_body_cbc.bin > Tux_cbc.ppm
convert Tux_cbc.ppm Tux_cbc.png
echo "CBC encryption complete. Tux_cbc.png created."

# Encrypt the body using AES in OFB mode
echo "Encrypting with AES-128-OFB..."
openssl enc -aes-128-ofb -nosalt -pass file:./A1P3.key -in _tux_body -out
_tux_body_ofb.bin -v -p
cat _tux_header _tux_body_ofb.bin > Tux_ofb.ppm
convert Tux_ofb.ppm Tux_ofb.png
echo "OFB encryption complete. Tux_ofb.png created."

# Encrypt the body using AES in CTR mode
echo "Encrypting with AES-128-CTR..."
openssl enc -aes-128-ctr -nosalt -pass file:./A1P3.key -in _tux_body -out
_tux_body_ctr.bin -v -p
cat _tux_header _tux_body_ctr.bin > Tux_ctr.ppm
convert Tux_ctr.ppm Tux_ctr.png
echo "CTR encryption complete. Tux_ctr.png created."

# Encrypt the body using AES in GCM mode
echo "Encrypting with AES-128-GCM..."
openssl enc -aes-128-gcm -salt -pass file:./A1P3.key -in _tux_body -out
_tux_body_gcm.bin -v -p

echo "Tux has been encrypted and converted into PNGs"
```

- Compare the images that have been generated.
- Think about why GCM failed with `openssl` in this case.

**Key Concepts:**

- Modes of Operation: Experimenting with CBC, ECB, CTR, CGM modes to understand their differences and impacts on security.

## A1.R: Review and Self-Assessment

**Objective:** Consolidate your knowledge and assess your understanding.

**Tasks:**

- Review:
  - Summarize the key concepts you've learned about symmetric encryption, block ciphers, stream ciphers, and their practical implementations.
  - Revisit any areas that need clarification or additional practice.
- Self-Assessment:
  - Encrypt and decrypt a file using both AES (block cipher) and Chacha20 (stream cipher), then explain the process and differences in your own words.

    - Describe scenarios where you would prefer a block cipher over a stream cipher, and vice versa.
    - Verify your understanding by answering questions or taking a quiz on symmetric encryption, block ciphers, and stream ciphers.

## Helpful Tips

- Use the **enc** command in `openssl` to display all available commands and get to know them better:

```
$ openssl enc -help
```

- To get information about the key and encryption parameters stored in an encrypted file, you can use **-p** flag while encrypting a file:

```
$ openssl enc -d -aes-256-cbc -in <encrypted_file> -out /dev/null -p # To get information about the encryption key and IV
```

- To list all available symmetric algorithms you can use the `list` command in `openssl`:

```
$ openssl list -cipher-algorithms
```