Lab 2 (Crypto) Symmetric Cryptography

Learning Objectives

- Understand symmetric encryption and its applications.
- Compare different algorithms (DES, 3DES, AES).
- Understand the impact of key size, cipher mode (ECB vs CBC), and padding.
- Use SSL/TLS tools to verify whether a system's encryption is **configured securely** and meets current **security and compliance standards**.

Tools Required: Kali Linux with *OpenSSL, sslscan, sslyze,* text editor, sample text file, sample image.

Overview

Symmetric encryption is a core cryptographic technique where the **same key** is used for both encryption and decryption. It ensures **confidentiality**, allowing only parties who possess the secret key to recover the original information.

Key Concepts

1. **Shared Secret Key**
   Both sender and receiver must securely share the same key. The security of the system depends entirely on keeping this key secret. If an attacker obtains the key, all encrypted communication is compromised.
2. **Block vs Stream Ciphers**
   - **Block ciphers** divide plaintext into fixed-size blocks (e.g., 128 bits) and encrypt each block, often using chaining or feedback modes to enhance security.
   - **Stream ciphers** generate a continuous keystream, combining it with plaintext one unit at a time, making them suitable for real-time or streaming data.
3. **Cipher Modes of Operation**
   Block ciphers use various **modes of operation** (ECB, CBC, CTR, GCM) to encrypt multiple blocks securely. Modes handle block chaining, initialization vectors (IVs), and randomness to prevent pattern leakage. Some modes, like GCM, also provide **integrity protection**.
4. **Deterministic vs Probabilistic Encryption**
   - **Deterministic encryption**: Same plaintext + key → same ciphertext. Vulnerable to pattern analysis.

- **Probabilistic encryption**: Introduces randomness (IVs, nonces, salts), producing different ciphertexts for repeated encryption, enhancing security.

5. **Security Goals**
   Symmetric encryption ensures **confidentiality**. Combined with MACs or AEAD modes, it can also provide **integrity** and **authenticity**.
6. **Applications**
   Symmetric encryption is widely used in TLS/SSL, disk encryption, database protection, and secure messaging due to its efficiency and performance for large datasets.

Common Symmetric Encryption Algorithms

Algorithm	Type	Key/Block Size	Features	Typical Use Case
AES	Block	128-bit block, 128/192/256-bit key	Fast, secure, widely adopted; supports multiple modes (CBC, CTR, GCM)	TLS/SSL, disk encryption, secure messaging
DES	Block	64-bit block, 56-bit key	Legacy standard; short key length makes it insecure today	Historical reference, academic purposes
3DES	Block	64-bit block, 112/168-bit key	Encrypts each block three times; more secure than DES but slower	Legacy systems, backward compatibility
Blowfish	Block	64-bit block, 32–448-bit key	Flexible key size; fast in software	General-purpose encryption, older software
Twofish	Block	128-bit block, up to 256-bit key	AES finalist; strong security; efficient in software	Disk encryption, file encryption
RC4	Stream	40–2048-bit key	Simple and fast; now considered insecure due to biases	Historical network protocols (SSL v3, WEP)
ChaCha20	Stream	256-bit key	Modern, fast, secure; resistant to timing attacks	Secure messaging, VPNs, TLS 1.3 (alternative to AES-GCM)

> **Note:** Block ciphers encrypt fixed-size blocks, while stream ciphers operate on continuous streams of data.

Common Cipher Modes for Block Ciphers

Mode	Type	Description	Pros	Cons	Use Case
ECB	Block	Encrypts each block independently using same encryption key.	Simple and fast	Deterministic; leaks patterns; insecure for multi-block messages	Rare; single-block encryption/te
CBC	Block	Each block XORed with previous ciphertext; IV used for first block	Hides patterns; widely supported	Sequential encryption; error propagation; requires random IV	File/messag encryption (legacy)
CFB	Block → Stream	Encrypts previous ciphertext to generate keystream; XOR with plaintext	Stream encryption; arbitrary length	Sequential; error propagation	Streaming d encryption
OFB	Block → Stream	Generates keystream from previous cipher output; XOR with plaintext	Precomputable keystream; errors do not propagate	Sequential; IV must be unique	Secure streaming; legacy syste
CTR	Block → Stream	Encrypts incrementing counter; XOR with plaintext	Parallelizable; supports random access	Nonce reuse breaks security	TLS, disk encryption, h performance applications
GCM	AEAD (CTR + authentication)	CTR encryption with Galois-field authentication; outputs ciphertext + tag	Provides confidentiality + integrity; parallelizable	Nonce reuse breaks security	TLS 1.2/1.3, VPNs, secur messaging
XTS	Block	Applies “tweak” per block for storage encryption	Prevents block repetition leaks; supports random access	No authentication; only for storage	Disk encrypt (BitLocker, LUKS)

Mode	Type	Description	Pros	Cons	Use Case
SIV	Deterministic AEAD	Deterministic encryption safely with authentication	Safe deterministic encryption; integrity included	Slower; less widely implemented	Encrypted databases, searchable encryption

> **Note:** Always use secure modes (e.g., GCM, CTR with unique nonce) and avoid ECB in multi-block messages.

Prerequisites

---

1. Install openssl and check version:

```
# Run to update and upgrade the system packages
sudo apt update
sudo apt upgrade -y

# Install openssl  using apt package manager
sudo apt install openssl

# Check openssl version
openssl version
```

2. Create a test file:

```
echo "This is a test message." > message.txt
```

3. Have a sample image (e.g., sample.jpg) ready for encryption experiments.

Activities

Activity 1: Encrypt/Decrypt files using DES, 3DES, AES

1. **Encrypt using DES**

```
openssl enc -des -in message.txt -out message_des.enc -k secret123 -pbkdf2 -
provider legacy
```

2. **Decrypt using DES**

```
openssl enc -d -des -in message_des.enc -out message_des_dec.txt -k secret123 -
pbkdf2 -provider legacy
```

3. **Validate decrypted file using `diff`**

```
diff message_des.enc message_aes_dec.txt
```

4. **Encrypt using AES-256-CBC**

```
openssl enc -aes-256-cbc -in message.txt -out message_aes.enc -k secret123 -
pbkdf2
```

5. **Decrypt using AES-256-CBC**

```
openssl enc -d -aes-256-cbc -in message_aes.enc -out message_aes_dec.txt -k
secret123 -pbkdf2
```

Activity 2: Measure encryption times

1. **Measure AES-256-CBC encryption time with OpenSSL**

```
time openssl enc -aes-256-cbc -in message.txt -out ciphertext.aes -k secret123 -
pbkdf2
```

> **Note:** What you get after the command finishes. For most use cases, **real** time is what you compare.

```
real    0m0.012s
user    0m0.008s
sys     0m0.004s
```

- **real** = elapsed “wall-clock” time
- **user** = CPU time spent in user mode
- **sys** = CPU time spent in kernel mode

2. **Compare encryption times for different ciphers**

```
time openssl enc -des-cbc -in message.txt -out ciphertext_des.aes -k mypassword -
pbkdf2 -provider legacy
time openssl enc -aes-128-cbc -in message.txt -out ciphertext_aes128.aes -k
mypassword -pbkdf2
```

```
time openssl enc -aes-256-cbc -in message.txt -out ciphertext_aes256.aes -k
mypassword -pbkdf2
```

Activity 3: Compare the sizes of ciphertext files

1. **Using the `ls -l` command**

```
ls -l ciphertext_des.aes ciphertext_aes128.aes ciphertext_aes256.aes
```

2. **Using the `stat` command (more detailed)**

```
stat ciphertext_des.aes
stat ciphertext_aes128.aes
stat ciphertext_aes256.aes
```

Activity 4: Compare ECB vs CBC Mode

Encrypting an image file (`sample.jpg`) using **AES-256** in two different modes:

- **ECB (Electronic Codebook)** — known to be insecure
- **CBC (Cipher Block Chaining)** — considered secure (when used properly)

1. **Generate a random key**

```
# For AES-256, use 32 bytes key (=256 bits key)
openssl rand -hex 32 > aes256.key
```

2. **ECB Mode Command**

```
openssl enc -aes-256-ecb -in sample.jpg -out sample_ecb.enc -K $(cat aes256.key)
-nosalt
```

- `-aes-256-ecb` → Uses AES with 256-bit key in ECB mode
- `-K` → Provides a 256-bit key in hex (32 characters = 16 bytes = 128 bits — this is only half of what's needed; should be 64 hex digits for AES-256)
- `-nosalt` → Disables OpenSSL's default salting mechanism (so encryption is deterministic)

> **Note:** The image is encrypted block by block **independently**

3. **Generate a random IV (Initialization Vector):**

```
openssl rand -hex 16 > aes_iv.hex
```

> **Note:** AES block size = 16 bytes (128 bits), so IV is 16 bytes.

4. **CBC Mode Command**

```
openssl enc -aes-256-cbc -in sample.jpg -out sample_cbc.enc -K $(cat aes256.key)
-iv $(cat aes_iv.hex) -nosalt
```

- `-aes-256-cbc` → Uses AES in CBC mode
- `-iv` → Initialization vector (must be random in production, here it's static for demo)

> **Note**: Each block is XORed with the previous ciphertext block before being encrypted. First block is XORed with the IV.

5. **Compare ECB and CBC outputs** by opening the files in an image viewer or examining binary patterns:

```
# ECB(Electronic Codebook)
xxd -p sample_ecb.enc | tr -d '\n' | fold -w32 | sort | uniq -c | sort -nr | head

# CBC (Cipher Block Chaining)
xxd -p sample_cbc.enc | tr -d '\n' | fold -w32 | sort | uniq -c | sort -nr | head
```

Activity 5: `sslscan` tools to perform detailed SSL/TLS analysis

A powerful CLI tool named `sslscan` to perform detailed SSL/TLS vulnerability scans against any target (e.g., `example.com`), uncovering deprecated protocols, weak cipher suites, and misconfigured certificates.

1. **Basic discovery, detailed cipher & protocol enumeration**

```
# Basic scan
sslscan badssl.com

# Scan specific port
sslscan badssl.com:443

# Multiple target scan
echo "badssl.com:443" > target.txt
echo "example.com:443" >> target.txt
sslscan --targets=target.txt
```

2. **STARTTLS / Mail service checks (implicit + explicit TLS)**

```
# Check for IMPLICIT TLS support
openssl s_client -connect pop.gmail.com:995 -crlf
openssl s_client -connect imap.gmail.com:993 -crlf
```

```
openssl s_client -connect smtp.gmail.com:465 -crlf

# Implicit TLS (direct TLS). Encrypted from the very beginning
sslscan smtp.gmail.com:465         # SMTPS
sslscan imap.gmail.com:993         # IMAPS
sslscan pop.gmail.com:995          # POP3S

# Check for STARTTLS support
openssl s_client -starttls pop3 -connect pop3.gmail.com:110 -crlf
openssl s_client -starttls imap -connect imap.gmail.com:143 -crlf
openssl s_client -starttls smtp -connect smtp.gmail.com:587 -crlf

# STARTTLS (upgrade). Connection starts as plain text, then client sends a
STARTTLS command to upgrade to encrypted mode.
sslscan --starttls=smtp smtp.gmail.com:25
sslscan --starttls=smtp smtp.gmail.com:587
sslscan --starttls=imap imap.gmail.com:143
sslscan --starttls=pop3 pop3.gmail.com:110
```

3. **Vulnerability indicator extraction (quick greps on sslscan output)**

```
sslscan badssl.com:443 | grep -iE "TLSv1\.0|TLSv1\.1|SSLv3|SSLv2"
sslscan badssl.com:443 | grep -iE "RC4|3DES|DES|EXPORT|NULL|LOW|WEAK|MD5"
sslscan badssl.com:443 | grep -iE
"Heartbleed|POODLE|FREAK|DROWN|Logjam|BEAST|insecure renegotiation"
```

4. **Certificate & Validity**

```
sslscan badssl.com:443 | grep -E "Subject:|Issuer:|Not After|Not Before|Serial
Number"
sslscan badssl.com:465 | grep -E "Subject:|Issuer:|Not After"
```

5. **Compliance-focused quick checks**
   Commands oriented to items auditors will expect—protocols, PFS, weak ciphers, cert expiry.

```
# Protocols: ensure no TLS1.0/1.1
sslscan badssl.com:443 | grep -E "TLSv1\.0|TLSv1\.1"

# PFS present
sslscan badssl.com:443 | grep -E "ECDHE|DHE"

# No weak ciphers present
sslscan badssl.com:443 | grep -E "RC4|3DES|DES|EXPORT|NULL|LOW|WEAK|MD5"
```

```
# Certificate expiration (Not After)
sslscan badssl.com:443 | grep -E "Not After"
```

Activity 6: `sslyze tools to perform detailed SSL/TLS analysis

> **Note:** For common ssl attack understanding follow this link: https://hackertarget.com/ssl-check/

```
# Quick triage
sslyze badssl.com:443

# Certificate deep-dive
sslyze --certinfo badssl.com:443

# STARTTLS (mail) example — SMTP
sslyze --starttls=smtp smtp.google.com:25

# Check compression, renegotiation, HTTP/2
sslyze --compression --reneg --http2 badssl.com:443

# Save machine-readable output
sslyze  badssl.com:443 --json_out=badssl.json
```

Resources

1. Kali OS Download URL