2. CRYPTOGRAPHIC FAILURE

📘 OWASP Top 10 — MODULE 2

A02: Cryptographic Failures — Complete Pentester Master Module

(Advanced + Beginner Friendly + Tool-Focused + Real-World Pentest Level)

---

1. Introduction

Cryptographic Failures (previously "Sensitive Data Exposure") happen when an application:

- Uses **no encryption**
- Uses **weak/old encryption**
- Misconfigures cryptographic systems
- Exposes secrets accidentally
- Transmits data in plaintext (HTTP)
- Stores secrets insecurely

This leads to:

- Account takeover
- Man-in-the-middle (MITM) attacks
- Database dumping
- Password cracking
- Session hijacking
- API key theft
- Credit card/PII leakage

---

2. Why Cryptographic Failures Occur

❌ Developers misunderstanding encryption

❌ Poor key management

❌ Using defaults

❌ Misconfigured SSL/TLS

❌ Hardcoded secrets

❌ Sending JWTs over HTTP

❌ Not enforcing encryption at REST or in transit

---

3. Real-Life Examples

- **Facebook API 2021:** User tokens stored in plaintext logs
- **Equifax 2017:** Outdated SSL certificate → attackers stayed 6 months
- **WhatsApp 2019:** Older encryption logic exploited for spyware

Cryptography failures = **persistent, silent, high-impact vulnerabilities**.

---

4. Types of Cryptographic Failures

Let's break this down into 7 categories.

4.1. Plaintext Transmission

Data sent without encryption.

Example:

```
http://login.example.com?user=atharv&pass=1234
```

Risk:

Credentials can be intercepted.

4.2. Outdated / Weak Encryption Algorithms

Weak algorithms include:

- **MD5**
- **SHA1**
- **DES / 3DES**
- **RC4**

Crackable with modern hardware.

4.3. Hardcoded Secrets and Keys

Example (found via decompiling):

```
String API_KEY = "AIzaSyD3-badkeyhere";
```

4.4. Poor Key Management

Keys stored in:

- GitHub repositories
- JS files
- Config files
- Logs
- URLs

4.5. Broken TLS Implementations

Examples:

- TLS 1.0 / 1.1
- No HSTS
- Self-signed certificates
- Weak cipher suites

4.6. Insecure Password Storage

❌ Plaintext

❌ Base64 (not encryption!)

❌ MD5

❌ SHA1

✔ Required: bcrypt, Argon2, PBKDF2

---

5. Pentesting Cryptographic Failures — Full Methodology

Below is a complete real-world pentesting workflow.

Step 1 — Check HTTP/HTTPS Enforcement

Use **curl**:

Command:

```
curl -I http://target.com
```

Explanation:

- `curl` = makes HTTP requests
- `I` = fetch only headers
- Checking if HTTP redirects to HTTPS

Vulnerable if:

Response is **200 OK** instead of:

```
301 Moved Permanently → https://target.com
```

Step 2 — Check TLS Certificate & Version

Command:

```
openssl s_client -connect target.com:443 -tls1
```

Explanation:

- `openssl s_client` = opens a TLS connection
- `connect target.com:443` = connect to server
- `tls1` = forces weak TLS 1.0

If server accepts → **TLS too old**.

Check supported protocols:

```
openssl s_client -connect target.com:443 -tls1_2
openssl s_client -connect target.com:443 -tls1_3
```

Step 3 — Enumerate Cipher Suites

Command:

```
nmap --script ssl-enum-ciphers -p 443 target.com
```

What it does:

- `nmap` = scanner
- `-script ssl-enum-ciphers` = list ciphers and grade them
- Shows weak: RC4, DES, 3DES, EXPORT ciphers

If weak ciphers detected → **Vulnerable**.

Step 4 — Check for HSTS Header

Command:

```
curl -I https://target.com | grep strict
```

What it does:

- `grep` searches for “strict-transport-security”
- If missing → MITM is possible

Step 5 — Inspect Cookies

Command:

```
curl -I https://target.com
```

Look for flags:

❌ Missing:

- Secure
- HttpOnly
- SameSite
- Max-Age

Example vulnerable cookie:

```
Set-Cookie: sessionid=abc123;
```

Should be:

```
Set-Cookie: sessionid=abc123; Secure; HttpOnly; SameSite=Lax
```

Step 6 — Detect Hardcoded API Keys

Tools:

✔️ trufflehog

✔️ gitleaks

✔️ GitHub dorks

Command:

```
gitleaks detect -v
```

What it does:

- Scans repo for leaked secrets
- Flags API keys, tokens, passwords

Step 7 — Detect Weak Hashing in Web Apps

Look at login responses, find parameters like:

```
password_hash=5f4dcc3b5aa765d61d8327deb882cf99
```

That is MD5 → predictable.

Step 8 — Password Cracking with Hashcat

Command:

```
hashcat -m 0 hashes.txt wordlist.txt
```

Explanation:

- `hashcat` = cracking tool
- `m 0` = mode 0 (MD5)
- `hashes.txt` = list of hashes
- `wordlist.txt` = dictionary

If cracked quickly → algorithm too weak.

Step 9 — Intercept Traffic with MITM

Tools:

✔ **mitmproxy**

✔ **Bettercap**

✔ **Wireshark**

Example Bettercap Command:

```
sudo bettercap -X
```

Explanation:

- `X` = sniff all HTTP traffic
- If credentials appear → encryption failure

Step 10 — Test JWT Token Strength

Decode JWT:

```
jwt_tool.py token.jwt
```

Check for "none" algorithm:

Header:

```
{"alg":"none","typ":"JWT"}
```

If allowed → CRITICAL.

Force crack weak key:

```
jwtcrack token.jwt /usr/share/wordlists/rockyou.txt
```

Explanation:

- Attempts to brute-force signing secret
- If cracked → weak JWT secret

---

6. Cryptographic Failure Attack Examples

Example 1 — MITM Credential Stealing

Traffic captured:

```
POST /login HTTP/1.1
user=atharv&password=123456
```

Reason: No HTTPS enforced.

Example 2 — Base64 Misuse

Developers think Base64 = encryption:

```
password = bXlwYXNzMTIz
```

Decoded easily:

```
echo bXlwYXNzMTIz | base64 -d
```

Example 3 — Weak JWT Secret

JWT header:

```
HS256
```

Secret found:

```
secret123
```

Modify role:

```
{
 "role": "admin"
}
```

Token resign:

```
jwt_tool.py -S -I -pc role -pv admin
```

Admin access gained.

Example 4 — MD5 Hashed Password

Hash:

```
5f4dcc3b5aa765d61d8327deb882cf99
```

Crack:

```
hashcat -m 0 hash.txt rockyou.txt
```

Output:

```
password
```

---

7. Tools Summary (Massive Toolbox)

Tool	Purpose	Key Commands
OpenSSL	TLS testing	s_client, cipher enumeration
Nmap	Cipher scanning	`ssl-enum-ciphers`
Gitleaks	Secret detection	`gitleaks detect`
Hashcat	Hash cracking	`hashcat -m 0`
Burp Suite	Weak crypto detection	Repeater/Proxy
Bettercap	MITM attacks	`bettercap -X`
Wireshark	Packet analysis	Filters: `http.auth`
jwt_tool	JWT testing	decode, sign, brute-force

---

8. Mitigation Guidelines

Developers must:

✔️ Use TLS1.2+

✔️ Enable HSTS

✔️ Use strong hashing: Argon2 or bcrypt

✔️ Do NOT store secrets in code

✔️ Disable weak cipher suites

✔️ Rotate keys regularly

✔️ Validate JWT signatures on server-side

✔️ Never use Base64 for sensitive data

---

9. Reporting Format

Title: Cryptographic Failure — Weak TLS + Sensitive Data Exposure

Severity: Critical

Endpoint: `/login`

Issue: Credentials transmitted without HTTPS

Impact: MITM attack possible

Steps to Reproduce:

1. Intercept HTTP request
2. Observe plaintext credentials

   Evidence: HTTP packet screenshot

   Mitigation: Enforce HTTPS with HSTS