# Cryptographic Failures

**Failures** refer to any situation where **data is not adequately protected in transit or at rest** due to missing, weak, or misconfigured cryptography. These failures expose sensitive data to unauthorized access or tampering — even without an actual exploit.

# Example:

| Scenario | Description |
| --- | --- |
| No HTTPS | App transmits credentials via HTTP instead of HTTPS |
| Weak Encryption | App uses outdated ciphers like RC4, DES, or 3DES |
| Hardcoded Keys | App contains hardcoded secrets or private keys |
| Insecure Password Storage | Plaintext passwords or unsalted hashes |
| Missing TLS Validation | App accepts invalid/self-signed TLS certs |
| JWT Without Signature | Uses alg: none, no signature validation |

Sensitive data like passwords or credit card information is transmitted over HTTP without encryption.

* URL: http://example.com/login

# Impact:

| Type | Description |
| --- | --- |
| Data Theft | PII, passwords, financial data, session tokens exposed |
| Man-in-the-Middle Attacks (MitM) | Eavesdropping or modifying requests/responses |
| Authentication Bypass | Predictable or brute-forceable tokens |
| Regulatory Violations | GDPR, HIPAA, PCI-DSS fines and penalties |
| Reputation Damage | User trust lost after data leak |

Attackers can intercept sensitive data using man-in-the-middle attacks, leading to identity theft or financial loss.

# Types of Cryptographic Failures

## 1. Data in Transit Not Encrypted

* **Example**: Login form sends POST request over http://
* **Fix**: Enforce HTTPS with strong TLS (≥1.2), HSTS headers

## 2. Insecure Password Storage

* **Example**: Stored in plaintext or weak hashes like MD5/SHA1
* **Fix**: Use bcrypt, scrypt, or Argon2 with salting + iterations

## 3. Weak or Deprecated Algorithms

* **Example**: RC4, MD5, SHA1, DES, Base64
* **Fix**: Use AES-256, SHA-256/512, RSA ≥ 2048-bit

## 4. Hardcoded Secrets

* **Example**: Secrets or API keys in source code/GitHub
* **Fix**: Use environment variables or vault services (e.g., HashiCorp Vault)

## 5. Improper Certificate Validation

* **Example**: Accepts invalid/self-signed certificates
* **Fix**: Enforce strict TLS cert verification in client/app

## 6. Missing Integrity or Signature

* **Example**: JWTs with alg: none or unsigned config files
* **Fix**: Use HMAC-SHA256 or RSA-signed tokens with strict verification

## 7. Predictable Keys / IVs

* **Example**: Reusing initialization vectors (IVs) in AES-CBC mode
* **Fix**: Generate cryptographically secure random IVs and keys per session

# Tools Detect Cryptographic Failures

| **Tool** | **Use** |
| --- | --- |
| **Burp Suite** | Check TLS usage, HTTP endpoints, JWT structure |
| **testssl.sh / SSL Labs** | Analyze SSL/TLS configurations |
| **GitLeaks / TruffleHog** | Scan for secrets in code repos |
| **jwt.io** | Decode and validate JWTs |
| **Wireshark** | Inspect plaintext traffic over HTTP or weak ciphers |
| **Static Code Analysis** | Look for hardcoded keys, use of weak crypto libraries (Crypto, OpenSSL, etc.) |

# Mitigation

## Data in Transit

| Fix | Description |
| --- | --- |
| Enforce HTTPS | Use TLS 1.2+ and HSTS |
| Strong Cipher Suites | Disable RC4, 3DES, export-grade ciphers |
| Validate Certs | Don’t accept invalid/self-signed certs in code |

## Data at Rest

| Fix | Description |
| --- | --- |
| Use AES-256 | Industry standard block cipher |
| Encrypt DB and backups | Especially with regulated data |
| Encrypt API tokens | Do not store them in plaintext |

## Key Management

| Area | Action |
| --- | --- |
| No Hardcoded Secrets | Use .env, secret managers, never hardcode keys |
| Algorithm Choices | Avoid: MD5, SHA1, DES, RC4. Prefer: SHA-256+, AES-GCM |
| Key Rotation | Regularly rotate encryption/signing keys |
| Token Expiry | Set short-lived JWTs, invalidate on logout/change |

* Use HTTPS to encrypt data in transit.
* Store sensitive data using strong, modern encryption algorithms.
* Regularly update cryptographic libraries and ensure proper key management.

# Points

“Cryptographic Failures often stem from developers **using encryption incorrectly**, not from the algorithms themselves being flawed.”

“Test for this via **TLS inspection, secrets auditing, JWT validation**, and **manual code reviews**.”

“Secure coding policy mandates **bcrypt for passwords**, **TLS 1.2+**, and **managed KMS for key handling**.”

# Cryptography Cheat Sheet

| Task | Secure Practice |
| --- | --- |
| Password hashing | bcrypt, scrypt, Argon2 |
| Symmetric encryption | AES-256 in GCM mode |
| TLS | Version ≥ 1.2, valid certs |
| JWT | Sign with RS256 or HS256, never none |
| Keys | Store in AWS KMS, Azure Key Vault, or HashiCorp Vault |
| RNG | Use crypto.randomBytes() or CSPRNGs only |