Lesson 8
Cryptographic vulnerabilities

A. Sorokin (Input | Output)

Self paced course

July 31, 2025

Cryptographic security and failures

Cryptography ensures **confidentiality, integrity, and authenticity** of data.

However, when cryptographic systems fail to provide the intended protection, opportunities arise for gaining unauthorized access to data such as credentials, credit card numbers, and personal information.

Classification of cryptographic vulnerabilities

Four pillars of cryptographic failure are vulnerabilities in

(1) algorithm itself (*the cipher*);

(2) implementation (*the code*);

(3) protocol and usage (*the configuration*);

(4) system and environment (*the context*).

1. The Cipher

Vulnerabilities in the algorithm itself are flaws in the underlying mathematical design of the cryptographic primitives (ciphers, hash functions).

These are often the most severe but, thankfully, also the rarest in modern algorithms. This is why community-recommended standard algorithms like AES and SHA-256 are used instead of rolling our own.

1. The Cipher

Types of algorithm vulnerabilities:

(i) **theoretical breaks**: an attack that is better than brute force but is still impractical (e.g., reducing the effective key strength from 128-bit to 126-bit);

(ii) **practical breaks**: a fundamental flaw that makes the algorithm completely insecure (e.g., the WEP encryption algorithm for WiFi was completely broken);

(iii) **weak constructions**: algorithms that have inherent weaknesses, like a small key space (e.g., DES's 56-bit key) or undesirable properties (e.g., non-random output in hash functions).

2. The Code

Vulnerabilities in implementation mean that the algorithm is sound, but the way it was programmed introduces flaws. This is one of the most common sources of real-world vulnerabilities.

A perfect design can be destroyed by a bad implementation, so writing secure cryptographic code is extremely important yet being difficult.

2. The Code

Types of implementation vulnerabilities:

(i) **side-channel attacks**: leaking information through indirect channels other than the algorithm's input/output;

- **timing attacks**: measuring how long an operation takes provides information about the key or data (e.g., a string comparison that exits early on a wrong character);
- **power analysis**: monitoring power consumption of a device (like a smart card) to infer cryptographic operations;
- **fault injection**: itentionally causing a hardware error (e.g., with a laser or voltage glitch) to produce an erroneous output that reveals key material;

2. The Code

Types of implementation vulnerabilities:

(ii) **memory handling issues**:

- **heartbleed**: failing to properly validate input leads to leaking the contents of a server's memory, which could include private keys;
- **failure to securely wipe** sensitive data (e.g. keys) from memory after use;
- **integer overflows/underflows**: incorrect math leading to buffer overflows or logic errors during cryptographic operations.

3. The Configuration

When the algorithm and implementation are correct, but they are used incorrectly or within a flawed larger protocol then **vulnerabilities in protocol and usage** arise.

Cryptography is a toolkit, so using the right tool in the right way is just as important as having the right tool.

3. The Configuration

Types of protocol and usage vulnerabilities:

(i) **weak cryptographic primitive**: using a deprecated algorithm (e.g., MD5, RC4, SHA-1) for a sensitive task;

(ii) **insufficient key size**: using a key length that is too short to resist modern brute-force attacks;

(iii) **poor key management**:

- **hardcoded keys** (e.g. shipping keys within software);
- **poor key generation** (e.g. using a non-cryptographically secure random number generator to generate keys);
- **lack of key rotation** (e.g. never changing keys).

3. The Configuration

Types of protocol and usage vulnerabilities:

(iv) **incorrect encryption mode**: using the Electronic Codebook mode (encrypting every block individually) for encryption, which leaks patterns in the plaintext (e.g. famous ECB penguin image);

(v) **missing integrity protection**: encrypting data without authenticating it which allows attackers to malleate ciphertexts (e.g. using AES-CBC without HMAC);

(vi) **reusing nonces/IVs**: using the same nonce (number used once) with the same key in certain modes (e.g. AES-GCM) catastrophically compromises security.

4. The Context

Vulnerabilities in system and environment appear when the cryptography is perfect, but the surrounding system undermines its security.

Security is a chain, and it's only as strong as its weakest link, so every cryptographic component must be supported by a secure environment.

4. The Context

Types of system and environment vulnerabilities:

(i) **poor randomness**: a system-wide lack of entropy, leading to predictable "random" numbers used for keys and nonces;

(ii) **physical access**: failure to use hardware security modules or secure enclaves, allowing an attacker a physical access to extract keys;

(iii) **compromised root of trust**: if the Certificate Authority (CA) system is compromised, or hardware trust anchors are faulty, the entire chain of trust breaks;

(iv) **implementation-adjacent flaws**: an application that performs encryption *after* compression can be vulnerable to attacks like CRIME.

Best practices

Best practices to avoid crypto vulnerabilities and enhance privacy are:

(a) *use strong algorithms* for encryption (e.g., AES-256 for data, ChaCha20 for mobile) and hashing (e.g., SHA-256 or SHA-3), avoid deprecated algorithms (e.g., MD5, SHA-1, DES);

(b) *secure key management*: use hardware security modules, practice key rotation, avoid hardcoded keys and shared keys across apps;

(c) *enforce HTTPS everywhere* via using TLS 1.2/1.3, disable SSLv3, TLS 1.0;

Best practices

(d) *validate inputs and use constant-time code*, prevent buffer overflows;

(e) *encrypt data at rest* (e.g., BitLocker for Windows, FileVault for Mac) and *in transit* (e.g., TLS (HTTPS), VPNs for sensitive data);

(f) *audit and update* via scannig for vulnerabilities (e.g, nmap, OpenVAS) and patching crypto libraries (e.g., OpenSSL).