Security Analysis of ISO 9796-2 Digital Signature and RSA Signature Implementation

RSA with SHA-256

Vikash Kumar Ojha

CS22B013

[9th Nov, 2025]

Executive Summary

Scope of Analysis

- ISO 9796-2 digital signature with message recovery capability
- RSA-2048 minimum key size
- SHA-256 cryptographic hashing
- OAEP padding for RSA encryption/decryption
- ISO 9796-2 scheme 1 for signatures with message recovery
- NIST-compliant RSA key generation

Overall Assessment

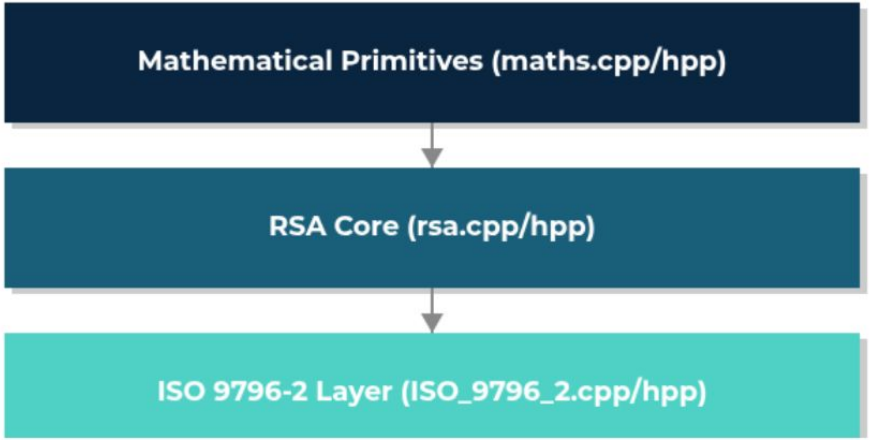
- Core Cryptographic Primitives: **SOUND**
- Side-Channel Resistance: **NEEDS IMPROVEMENT**
- Suitable for educational and low-risk applications
- Production deployment requires addressing side-channel vulnerabilities

Key Findings Summary

Issue	Severity	Impact
Information Leakage (Padding Oracle)	HIGH	Side-channel attacks
Non-Constant-Time Operations	MEDIUM-HIGH	Timing attacks
Debug Output in Verification	MEDIUM	Information disclosure
No Secure Memory Wiping	MEDIUM	Memory forensics risk
128-bit Salt	LOW-MEDIUM	Long-term collision risk

Implementation Architecture

Three-Layer Design



1. Mathematical Primitives

maths.cpp/hpp

- GMP-based modular arithmetic
- Miller-Rabin primality testing
- Cryptographically secure RNG

2. RSA Core

rsa.cpp/hpp

- NIST-compliant key generation
- OAEP padding (PKCS#1 v2.1)
- Standard RSA operations

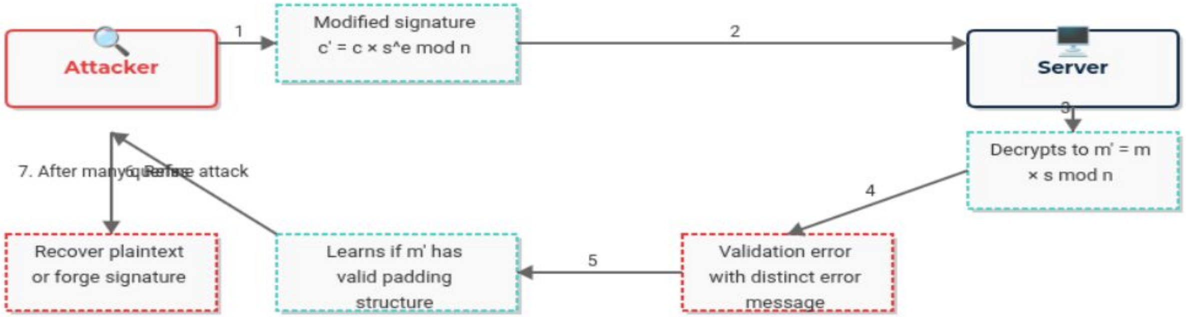
3. ISO 9796-2 Layer

ISO_9796_2.cpp/hpp

- Message encoding with salt/trailer
- Signature generation/verification
- Message recovery from signature

Critical Vulnerability #1 - Padding Oracle

Severity: HIGH



Technical Description

The verification function leaks information about signature structure through distinguishable error messages.

- Different error paths exist for different validation failures
- Error responses reveal which validation step failed
- Timing differences also leak verification details

Oracle Classification

This is a format oracle that reveals structural information:

- Which validation step failed
- Structural properties of decrypted signature
- Information about padding correctness

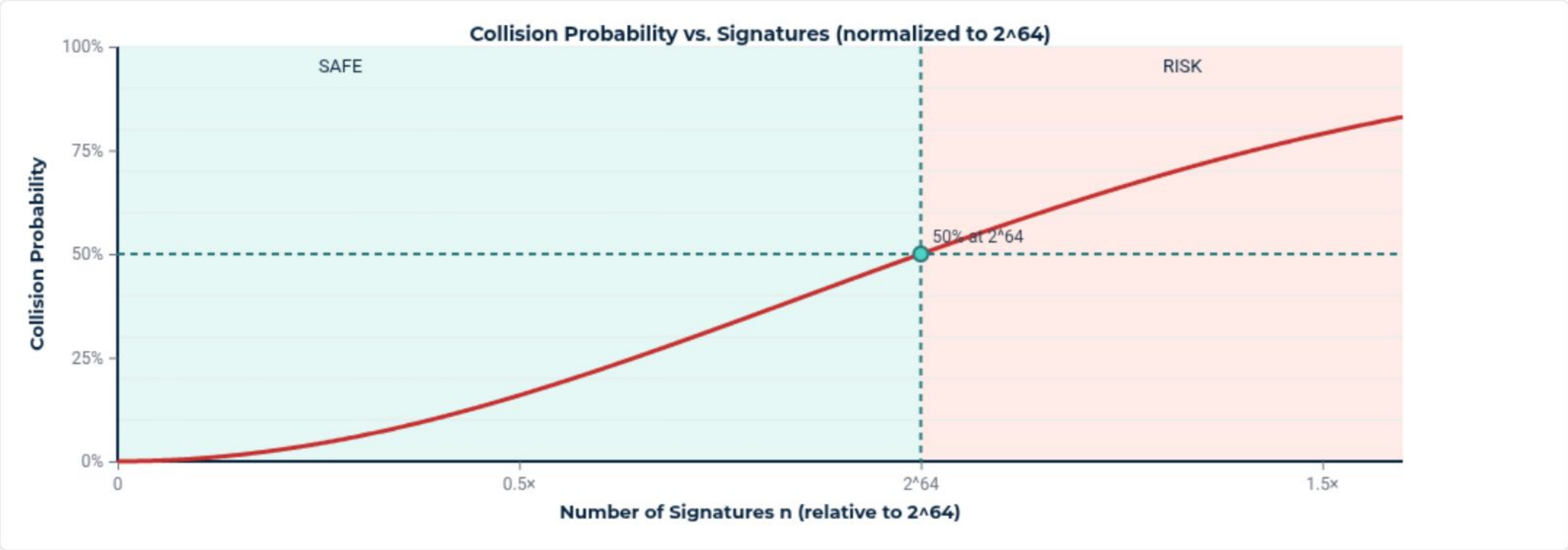
Attack Vectors

Bleichenbacher-style Attack:

- Submit many encoded text variations
- Build tree of valid/invalid padding responses
- Narrow down valid message space
- Eventually recover or forge messages

Critical Vulnerability #2 - Salt Entropy

Severity: MEDIUM-HIGH



Technical Description

- Implementation uses only 128-bit random salt
- Insufficient for long-term security requirements

Birthday Considerations

- Birthday bound: 2^64 signatures
- Collision probability: P ≈ n² / (2 × 2^128)
- For n = 2^40: P ≈ 2^-48 (non-negligible)

Impact

- Randomization weakened on collision
- Potential related-message risks
- Reduced long-term security margin

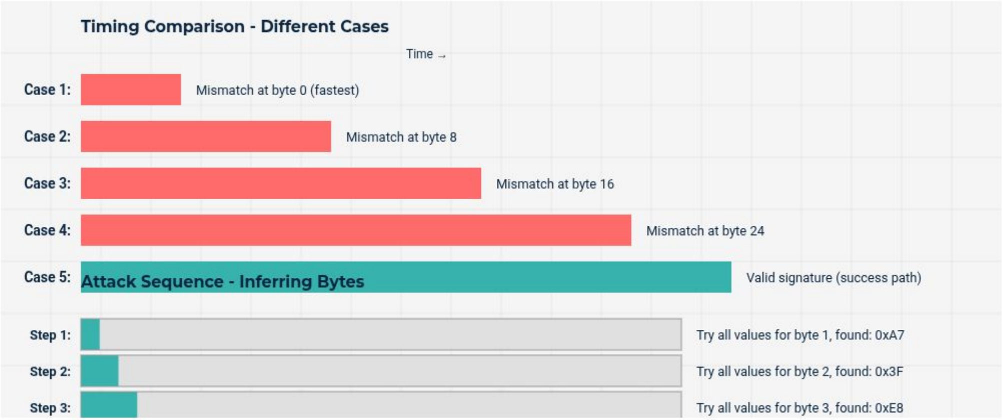
Recommendation

- Increase to 256-bit salt minimum
- Birthday bound becomes 2^128 signatures
- Future-proof against high-volume usage

Critical Vulnerability #3 - Timing Attacks

Severity: MEDIUM

Timing Attack Surface: Non-constant-time operations create measurable timing differences that can be exploited.



Vulnerable Code Pattern

Non-constant-time digest comparison:

```
if (digest != digest_check) {
    return {false, {}};
}
```

The vector comparison operator stops at first differing byte:

- Match at byte 0: Fastest failure
- Match at byte 31: Slowest failure
- Full match: Different code path (success)

Exploitation Steps

1. Submit signatures with varying hash values
2. Measure verification time
3. Infer correct hash bytes one at a time
4. Eventually forge valid signatures

Other Variable-Time Operations

- Loop iterations depend on message length
- Different error paths have different execution times
- Memory operations vary with data size

Vulnerability #4 – Prime Generation

Severity: MEDIUM

Technical Description

- Reliance on probabilistic primality testing (Miller-Rabin) only

Missing Security Checks

- **Strong primes:** p-1 should have a large prime factor
- **Smooth number prevention:** Check that p-1 is not B-smooth for small B
- **Safe prime option:** Consider $p = 2q + 1$ where q is prime
- **Distance between primes:** Ensure $|p - q|$ is sufficiently large (present in code)

Attack Implications

- **Pollard's p-1 Algorithm:**
  - If p-1 has only small prime factors, factorization is efficient
  - Complexity: $O(B \log B \log^2 n)$ where B is largest factor of p-1
  - Risk: Higher if primes are not carefully selected
- **Williams's p+1 Algorithm:**
  - Similar vulnerability if p+1 is smooth
  - Uses Lucas sequences instead of modular exponentiation

Vulnerability #5 – Memory Security

Severity: MEDIUM

Issues

- No explicit secure memory wiping
- Sensitive buffers persist after use
- Code Example:

```
std::vector<unsigned char> salt(16);
RAND_bytes(salt.data(), salt.size());
// ... use salt ...
// salt vector destructor called, but memory not wiped
// Private keys, intermediate values remain in memory
```

Risk Exposure

- Memory forensics attacks
- Cold boot attacks (RAM remanence)
- Swap file exposure
- Memory dumps during crashes

Required Solutions

- Explicit memory wiping before deallocation
- Use of secure memory allocation functions
- Ensure compiler doesn't optimize away wiping

Positive Aspects

Strong Foundation

- RSA implementation aligns with **NIST guidance**
- Prime size checks, GCD constraints, and sufficient separation of p and q
- Private exponent validation for **Wiener's attack protection**
- Comprehensive validation checks: `if (d < (1 << bits/4)) return false;`

Libraries

- **OpenSSL** for SHA-256 and secure random number generation
- **GMP** for efficient arbitrary-precision arithmetic
- Modern C++ practices and standard library features
- Clean architecture with appropriate encapsulation

Padding Schemes

- **OAEP** padding implemented according to PKCS#1 v2.1 specifications
- **ISO 9796-2** scheme 1 implemented correctly for signatures with message recovery
- Proper use of salt and trailer fields in signature format
- Correct implementation of cryptographic primitives

RSA Key Size Recommendations (2025)

Key Size	Security Level	Symmetric Equivalent	Status
512 bits	0 bits	—	Broken (1999)
768 bits	~60 bits	—	Broken (2009)
1024 bits	~80 bits	2TDEA	Deprecated
2048 bits	~112 bits	3TDEA	Current minimum
3072 bits	~128 bits	AES-128	Recommended
4096 bits	~140 bits	—	High security
7680 bits	~192 bits	AES-192	PQ hedge
15360 bits	~256 bits	AES-256	Classical max

NIST Guidance:

Minimum for current use: 2048 bits
Protection beyond 2030: 3072 bits
Long-term security: 4096+ bits

Major Threat Categories

Six Main Attack Vectors

1 Mathematical Attacks

- Factorization (GNFS, special number methods)
- Direct RSA attacks (common modulus, low exponent)

2 Padding Oracle Attacks

- Bleichenbacher's attack on PKCS#1 v1.5
- Manger's attack on OAEP implementation flaws

3 Side-Channel Attacks

- Timing, power analysis (SPA/DPA)
- Fault attacks, cache-timing attacks

4 Implementation Bugs

- Weak random number generation
- Prime generation flaws, memory vulnerabilities

5 Protocol-Level Attacks

- Chosen ciphertext attacks
- Signature forgery via multiplicative property, key reuse

6 Quantum Computing

- Shor's algorithm (polynomial-time factoring)
- Harvest-now-decrypt-later strategy

Mathematical Attacks – Factorization

General Number Field Sieve (GNFS)

- Asymptotic leader for large moduli factorization
- Complexity: exp((1.923 + o(1))(ln n)^(1/3)(ln ln n)^(2/3))
- Current record: **829-bit number** factored (2020)
- Practical limit: ~1024 bits with current resources
- Works on any modulus but requires enormous computational resources

Special Number Factorization

- **Fermat's Method**: If |p - q| is small, factors can be found by searching near √n
- **Pollard's p-1**: If p-1 is smooth (has only small prime factors), modulus can be factored efficiently
- **Williams's p+1**: Similar to Pollard's but exploits when p+1 has only small prime factors
- These methods exploit poor prime selection and can break RSA keys much faster than GNFS

Mathematical Attacks – Direct RSA

Common Modulus Attack

- Same `n` shared between users with different exponents
- Attacker intercepts same message encrypted to both users
- Extended Euclidean Algorithm recovers plaintext without factoring

Low Private Exponent

- Wiener's attack works when `d < n^{0.25}`
- Uses continued fraction methods to recover small private keys
- Boneh-Durfee extends this to `d < n^{0.292}` using lattice techniques

Low Public Exponent

- Hastad's Broadcast Attack exploits small `e` (like `e=3`)
- When same message is sent to `e` different recipients without randomized padding
- Chinese Remainder Theorem allows computing `m^e` over integers
- Message recovered by taking `e`-th root: `m = sqrt[e]{m^e}`

Franklin-Reiter Attack

- Targets linearly related messages encrypted with small exponent
- If `m_2 = a*m_1 + b` for known `a,b`
- Both messages recovered by computing polynomial GCD
- Shows that even slight message relationships can be exploited

Padding Oracle Attacks

Bleichenbacher's Attack (PKCS#1 v1.5)

- **Mechanism:** Server reveals whether decrypted ciphertext has valid padding format (0x00 0x02)
- **Attack Process:** Submit modified ciphertexts $c' = c \times s^e \bmod n$
  - When decrypted: $m' = m \times s \bmod n$
  - Oracle responses narrow possible values of m
- **Complexity:** 2^{20} to 2^{24} oracle queries (feasible)

Manger's Attack (OAEP)

- Exploits implementations revealing if decrypted value is numerically less than the modulus
- Uses binary search techniques similar to Bleichenbacher but targets different oracle condition

General Format Oracles

- **Key Principle:** If an attacker can tell WHY validation failed, they gain information about decrypted content
- **Common Information Leaks:**
  - Different error codes for "bad padding" vs "bad hash"
  - Timing variations indicating which check failed
  - Exception types or HTTP status codes
- **Defense Strategy:**
  - Uniform error messages for all failures
  - Constant-time validation regardless of error
  - Single error path for all validation failures

Side-Channel Attacks

Timing Attacks (Kocher)

- Square-and-multiply exponentiation leaks bit patterns
- Operation time differs for '0' vs '1' key bits
- Statistical correlation reveals private key
- **Defense:** Constant-time algorithms, blinding, Montgomery ladder

Power Analysis

- **SPA:** Direct power trace reveals operations
- **DPA:** Statistical analysis of many traces reveals secrets
- **CPA:** Correlation between hypothesized and actual power

Fault Attacks (Bellcore)

- Targets RSA-CRT optimization
- Induces fault during computation (voltage spike, clock glitch)
- Combines correct m_p with faulty m_q'
- gcd(m'^e - c, n) reveals factor p, breaking the system

Cache-Timing Attacks

- Exploit CPU cache behavior
- **Techniques:** Flush+Reload, Prime+Probe
- Memory access patterns leak secret-dependent operations
- Requires co-location (shared CPU, cloud environment)

Implementation Vulnerabilities

Weak Random Number Generation

- **Debian OpenSSL Bug (2008):**
  - Code change removed entropy sources
  - Only 2^15 (32,768) possible keys
  - Keys generated 2006-2008 vulnerable
- **Batch GCD Attack:**
  - Research found 12,000+ internet keys sharing factors
  - Computing $\gcd(n_1, n_2)$ revealed shared primes

Prime Generation Flaws

- **Insufficient primality testing:**
  - Too few Miller-Rabin rounds
  - 25+ rounds needed for cryptographic security

Memory Vulnerabilities

- **Key exposure vectors:**
  - Private keys remaining in RAM
  - System swap files containing key material
  - Buffer overflows in padding operations
- **Cold Boot Attacks:**
  - RAM retains data after power loss (remanence)
  - Keys recoverable even minutes after shutdown
- **Prevention strategies:**
  - Explicit memory wiping (zeroization)
  - Non-swappable memory allocation
  - Compiler protections against optimization removal

Protocol-Level Attacks

Chosen Ciphertext Attacks

- **RSA Malleability:** Given $c = m^e$, attacker creates $c' = c \times r^e$
- When decrypted: $m' = (c')^d = m \times r$
- Multiple queries with carefully chosen r values reveal information
- **Defense:** Use IND-CCA2 secure padding (OAEP)

Signature Forgery & Key Reuse

- **Multiplicative Property:** $\sigma_1 \times \sigma_2 = (m_1 \times m_2)^d \bmod n$
- **Defense:** Hash messages with proper padding (PSS, ISO 9796-2)
- **Key Reuse Attack:** Using same key for encryption and signing
- Attacker tricks victim into "signing" a ciphertext to decrypt it
- **Defense:** Use separate keys for different purposes

Quantum Computing Threat

Shor's Algorithm

- Polynomial-time factoring: **$O((\log n)^3)$** quantum operations
- Factors any RSA modulus in polynomial time
- Resources Required: Thousands of logical qubits
- Current Status: Largest quantum computers have **~1000 qubits**

Harvest-Now-Decrypt-Later

- Adversaries store encrypted data today
- Decrypt when quantum computers become available
- Threat to long-term confidentiality of current RSA-encrypted data

Mitigation Strategy

- Monitor NIST post-quantum cryptography standards
- Plan migration timeline
- Consider hybrid classical/post-quantum schemes
- Evaluate data sensitivity and required protection period

Timeline Concern:

Sensitive data encrypted today may be at risk in 10-20 years when practical quantum computers become available. Organizations must consider whether their data needs protection beyond the quantum timeline.

Recommended Padding Schemes

Critical: Always use RSA with proper padding

Use Case	Padding Scheme	Standard
Encryption	OAEP	PKCS#1 v2.1, RFC 8017
Signatures	PSS or ISO 9796-2	PKCS#1 v2.1, ISO/IEC 9796-2
Legacy (avoid)	*PKCS#1 v1.5*	*Deprecated*

OAEP Properties

- Provably IND-CCA2 secure in random oracle model
- Prevents chosen-ciphertext attacks
- Randomized encryption (different ciphertexts for same message)

PSS Properties

- Provably secure signatures in random oracle model
- Tight security reduction to RSA problem
- Randomized signatures enhance security

Implementation Best Practices

Security Requirements for Safe RSA Usage

1 Constant-Time Operations

Protect against timing side-channels:

- No data-dependent branches in cryptographic code
- Constant-time modular exponentiation
- Constant-time padding verification
- Constant-time comparison functions

2 Blinding

Defeat power analysis and timing attacks:

- Multiply input by r^e before private key operation
- Multiply result by r^-1 after operation
- Randomizes intermediate values
- Defeats timing and power analysis

3 Error Handling

Prevent information leakage:

- Identical error messages for all decryption/verification failures
- No padding oracle information leakage
- Single error return path
- Constant-time error paths

4 Memory Security

Protect sensitive data in memory:

- Wipe sensitive data immediately after use
- Use secure memory allocation (mlock, non-swappable)
- Minimize lifetime of keys in memory
- Ensure wiping not optimized away by compiler

Thank You

Thank You

Questions?