Cryptographic Algorithms Implementation

From Scratch in C

Vraja Alexandru

May 19, 2025

Project Overview

- **Purpose**: Implement cryptographic algorithms without libraries
- **Language**: C
- **Algorithms Implemented**:
  - Symmetric Key: TEA, ChaCha20
  - Asymmetric Key: RSA

TEA (Tiny Encryption Algorithm)

- Designed by David Wheeler and Roger Needham (1994)
- Compact: Few lines of C code
- 128-bit key, 64-bit block
- 32 rounds using XOR, ADD, shifts

TEA Algorithm Structure

- Feistel Network Based
- Delta constant: `0x9E3779B9` (golden ratio)
- Simple operations: addition, XOR, shifts
- No S-boxes or permutations

TEA Encryption Code

```c
void tea_encrypt(uint32_t v[2], const uint32_t key[4])
    uint32_t sum = 0, delta = 0x9E3779B9;
    for (int i = 0; i < 32; i++) {
        sum += delta;
        v[0] += ((v[1] << 4) + key[0]) ^ (v[1] + sum)
        v[1] += ((v[0] << 4) + key[2]) ^ (v[0] + sum)
    }
}
```

ChaCha20

- Designed by Daniel J. Bernstein (2008)
- Stream cipher: keystream XOR with plaintext
- 256-bit key, 96-bit nonce, 32-bit counter
- Based on ARX (Addition, Rotation, XOR)

ChaCha20 Structure

- 4x4 state matrix of 32-bit words
- 20 rounds: 10 column + 10 diagonal rounds
- Core: quarter round function (ARX operations)
- No lookups $\rightarrow$ safe against timing attacks

ChaCha20 Quarter Round

```
static inline void chacha20_quarter_round(
    uint32_t* a, uint32_t* b, uint32_t* c, uint32_t* d
    *a += *b; *d ^= *a; *d = (*d << 16) | (*d >> 16);
    *c += *d; *b ^= *c; *b = (*b << 12) | (*b >> 20);
    *a += *b; *d ^= *a; *d = (*d << 8)  | (*d >> 24);
    *c += *d; *b ^= *c; *b = (*b << 7)  | (*b >> 25);
}
```

RSA Algorithm

- Asymmetric cryptography (public key)
- Designed by Rivest, Shamir, and Adleman (1977)
- Based on integer factorization
- Commonly 2048+ bit keys

RSA Key Generation

1. Select primes p, q
2. Compute $n = p \times q$
3. Compute $\phi(n) = (p-1)(q-1)$
4. Choose e with $1 < e < \phi(n)$, $\gcd(e, \phi(n)) = 1$
5. Compute d such that $(d \cdot e) \mod \phi(n) = 1$
6. **Public key**: (n, e), **Private key**: (n, d)

RSA Encryption and Decryption

- **Encryption**: $c = m^e \mod n$
- **Decryption**: $m = c^d \mod n$
- **m** is message, **c** is ciphertext
- Use modular exponentiation

Implementation Challenges

- C limitations: large integers
- Modular exponentiation
- Message padding/blocking
- No libraries allowed
- Manual error handling

Implementation Details

- **TEA**: Simple Feistel + padding
- **ChaCha20**: Matrix state + counters
- **RSA**: Square-and-multiply for exponentiation

Security Considerations

- TEA: Vulnerable to related-key attacks
- ChaCha20: Considered secure (used in TLS)
- RSA: Key size and implementation critical
- Note: Educational-only codebase

Thank You!

Thank You

`Vraja Alexandru`
Cryptographic Algorithms from Scratch