Cryptographic S/W Modules with C

Design, Implementation, and Integration of Core Crypto Modules

Secure, Efficient, High-Performance Cryptographic Software Modules

Ji, Yong-hyeon

hacker3740@kookmin.ac.kr

Department of Cyber Security
Kookmin University

April 13, 2025

Contents

Chapter 1

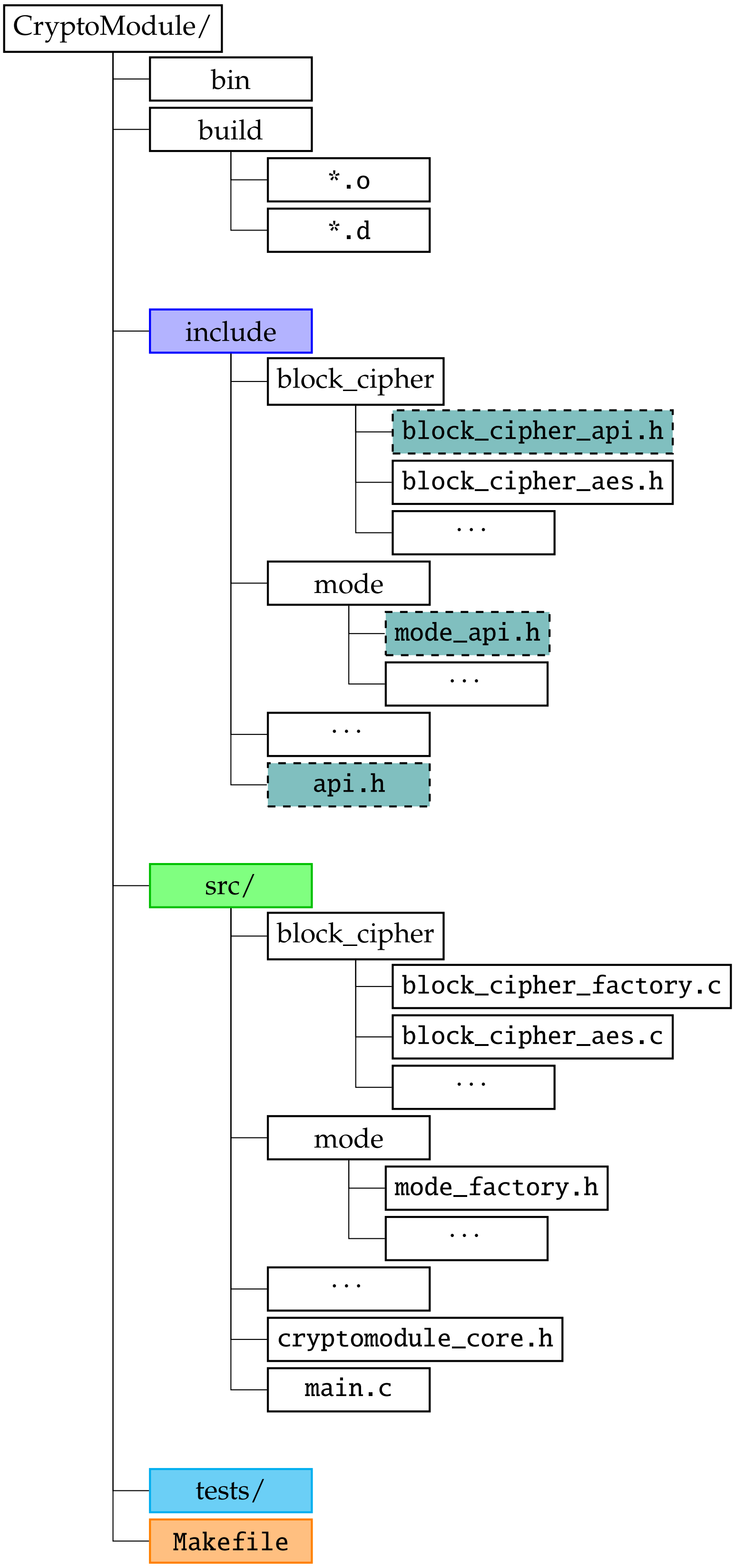
Project Overview

I have developed a cryptographic software module in the C language, with an emphasis on high performance and efficiency. This document provides a comprehensive guide to the design, implementation, and integration of cryptographic modules written in C (sometimes assembly).

Key Objectives:

- Describing the cryptographic primitives and algorithms
  (block ciphers, hash functions, MACs, signature algorithms, etc.).
- Explaining the structure of the source files and headers.
- Providing guidelines for building, testing, and integrating these modules into larger software systems.

1.1 Directory Structure



1.2 My Development Environment

- **Operating System:**

```
@>$ cat /etc/os-release
NAME="Linux Mint"
VERSION="21.3 (Virginia)"
ID=linuxmint
ID_LIKE="ubuntu debian"
PRETTY_NAME="Linux Mint 21.3"
VERSION_ID="21.3"
HOME_URL="https://www.linuxmint.com/"
SUPPORT_URL="https://forums.linuxmint.com/"
BUG_REPORT_URL="http://linuxmint-troubleshooting-guide.readthedocs.io/en/latest/"
PRIVACY_POLICY_URL="https://www.linuxmint.com/"
VERSION_CODENAME=virginia
UBUNTU_CODENAME=jammy
```

- **Compiler:**

```
@>$ gcc --version
gcc (Ubuntu 11.4.0-1ubuntu1~22.04) 11.4.0
Copyright (C) 2021 Free Software Foundation, Inc.
This is free software; see the source for copying conditions.  There is NO
warranty; not even for MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.
```

- **Hardware:**

```
@>$ lscpu
Architecture:            x86_64
  CPU op-mode(s):        32-bit, 64-bit
  Address sizes:         48 bits physical, 48 bits virtual
  Byte Order:            Little Endian
CPU(s):                  16
  On-line CPU(s) list:   0-15
Vendor ID:               AuthenticAMD
  Model name:            AMD Ryzen 7 5800X3D 8-Core Processor
    CPU family:          25
    Model:               33
    Thread(s) per core:  2
    CPU max MHz:         3400.0000
    CPU min MHz:         2200.0000
```

- **Additional Tools:**
  - valgrind for memory checks,
  - gdb for debugging,
  - and TBA

Chapter 2

Cryptographic Software Module

2.1 Block Cipher

A **block cipher** is a keyed family of permutations over a fixed-size data block.

- Let k be a fixed key size and n be a fixed block size.
- Let $\mathcal{K} = \{0, 1\}^k$ be the set of possible k-bit keys (each key is chosen from this set).
- Let $\mathcal{M} = \{0, 1\}^n$ be the set of all n-bit messages (plaintext blocks).
- Let $C = \{0, 1\}^n$ be the set of all n-bit ciphertext blocks.

A **block cipher** is have two induced functions:

$$E : \mathcal{K} \times \mathcal{M} \to C \quad \text{and} \quad D : \mathcal{K} \times C \to \mathcal{M},$$

referred to as the **encryption** and **decryption** functions, respectively. These must satisfy:

1. *Invertibility (permutation property)*: For each fixed key $k \in \mathcal{K}$, the encryption function

   $$E_k(\cdot) = E(k, \cdot) : \mathcal{M} \to C \quad \text{is a bijection (i.e., permutation) on } \{0, 1\}^n.$$

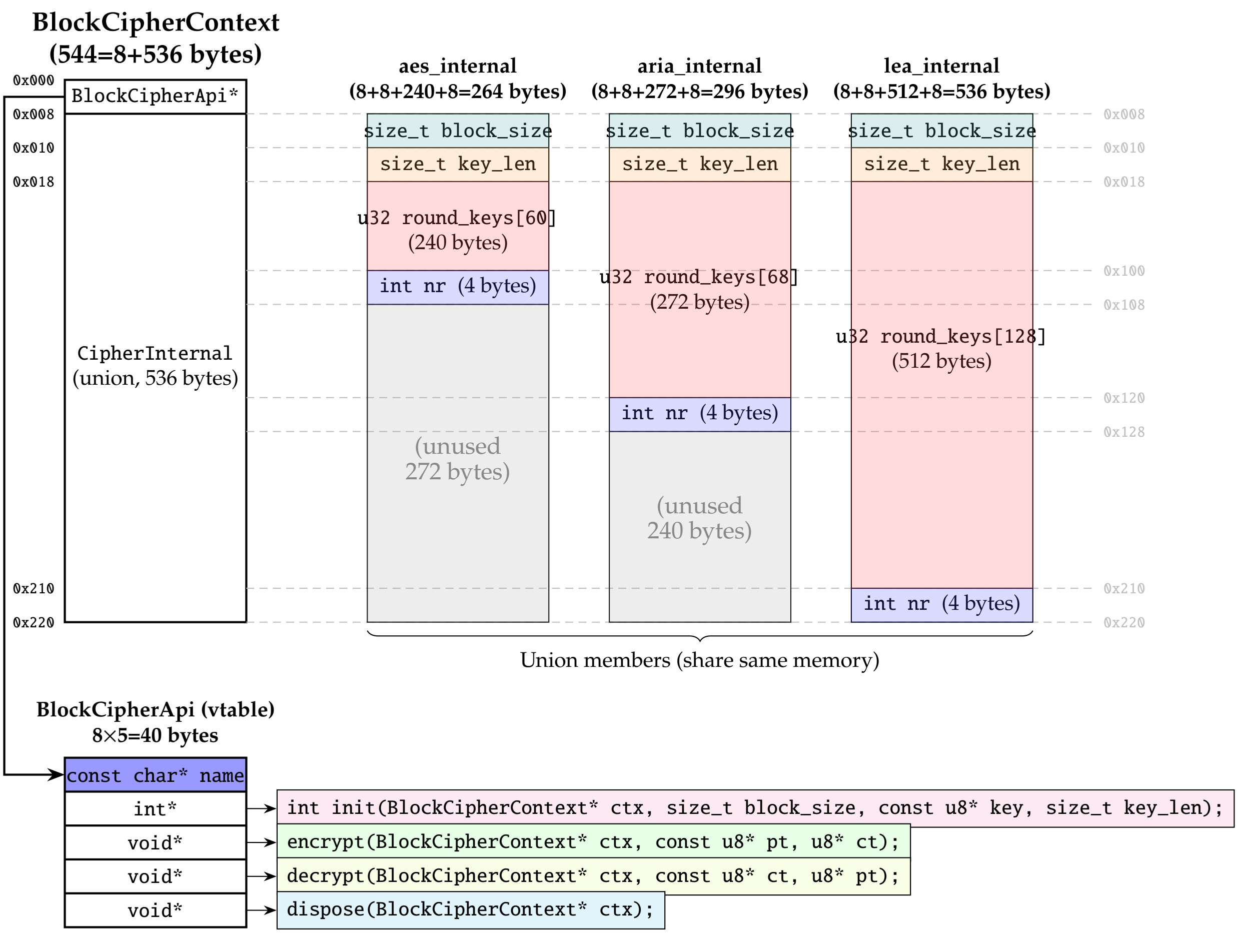
   In other words, for every key k, there is a unique inverse $D_k(\cdot) = D(k, \cdot) : C \to \mathcal{M}$ s.t.

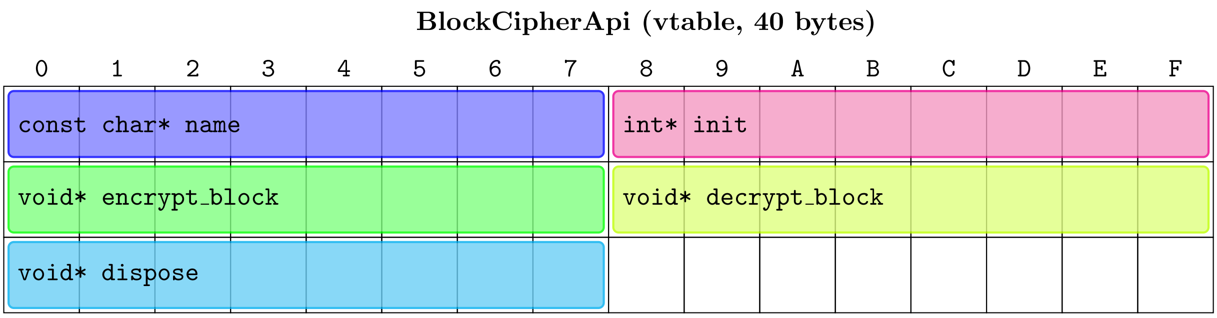
   $$D_k\big(E_k(m)\big) = m \quad \text{and} \quad E_k\big(D_k(c)\big) = c \quad \text{for every } m \in \mathcal{M} \text{ and } c \in C.$$

2. *Keyed operation*: The cipher's behavior depends on the choice of key k. Changing k results in a different permutation over the n-bit block space.

Alg.	n (bit)	k (bit)	**Rounds** (bit)	**RK Size** (bit)	**# of RKs** (bit)	**Total RK Size** (bit)
AES–128	128	128	10	128 (4-word)	11	1408 (44-word)
AES–192	128	192	12	128 (4-word)	13	1664 (52-word)
AES–256	128	256	14	128 (4-word)	15	1920 (60-word)
ARIA-128	128	128	12	128 (4-word)	13	1664 (52-word)
ARIA-192	128	192	14	128 (4-word)	15	1920 (60-word)
ARIA-256	128	256	16	128 (4-word)	17	2176 (68-word)
LEA-128	128	128	24	128 (4-word)	24	3072 (96-word)
LEA-192	128	192	28	128 (4-word)	28	3584 (112-word)
LEA-256	128	256	32	128 (4-word)	32	4096 (128-word)

Table 2.1: Comparison of AES, ARIA, and LEA parameters for 128-, 192-, and 256-bit keys.





2.1.1 AES (Advanced Encryption Standard)

Table 2.2: Parameters of the Block Cipher AES (1-word = 32-bit)

Algorithms	Block Size (N_b-word)	Key Length (N_k-word)	Number of Rounds (N_r)	Round-Key Length (word)	Number of Round-Keys ($N_r + 1$)	Total Size of Round-Keys ($N_b(N_r + 1)$)
AES-128	4	4	10	4	11	44 (176-byte)
AES-192	4	6	12	4	13	52 (208-byte)
AES-256	4	8	14	4	15	60 (240-byte)

Code 2.1: include/block_cipher/block_cipher.h

```
/* Forward declaration for the context. */
typedef struct BlockCipherContext BlockCipherContext;

/* The vtable or function pointer set describing any block cipher. */
typedef struct BlockCipherApi {
	const char *name; /* e.g. "AES" or "MyCipher" */

	/* Initialize the cipher with the chosen block size and key. */
	int (*init)(
		BlockCipherContext* ctx,
		size_t block_size,
		const u8* key,
		size_t key_len
	);
	/* Encrypt exactly one block. */
	void (*encrypt_block)(
		BlockCipherContext* ctx,
		const u8* plaintext,
		u8* ciphertext
	);
	/* Decrypt exactly one block. */
		void (*decrypt_block)(
		BlockCipherContext* ctx,
		const u8* ciphertext,
		u8* plaintext
	);
	/* Clean up resources, if needed. */
	void (*dispose)(
		BlockCipherContext* ctx
	);

} BlockCipherApi;

/* The context structure storing state. */
struct BlockCipherContext {
	const BlockCipherApi *api;
	u8 internal_data[256]; /* Example placeholder for key schedule, etc. */
};
```

Code 2.2: include/block_cipher/block_cipher_aes.h

```
const BlockCipherApi* get_aes_api(void);
```

Code 2.3: src/block_cipher/block_cipher_aes.c

```
typedef struct AesInternal {
	size_t block_size; /* Typically must be 16 for AES */
	size_t key_len; /* 16, 24, or 32 for AES-128/192/256 */
	u32 round_keys[60];
	int nr; /* e.g., 10 for AES-128, 12, or 14... */
} AesInternal;
```

2.1.2 ARIA (Academy, Research Institute, and Agency)

2.1.3 LEA (Lightweight Encryption Algorithm)

2.2 Modes of Operation

2.3 Random Number Generator

2.4 Hash Functions

2.5 Message Authentication Codes

2.6 Key Derivation Functions

2.7 Key Exchange

2.8 Signature Algorithms

Chapter 3

Build and Integration

Chapter 4

Testing

Appendices