Cryptographic-Module
Source-Code Development Manual

Design, Implementation, and Integration of Cryptography Modules

Secure, Efficient, High-Performance Cryptographic Software Modules

Ji, Yong-hyeon

`hacker3740@kookmin.ac.kr`

Department of Cyber Security
Kookmin University

May 29, 2025

Contents

Chapter 1

Project Overview

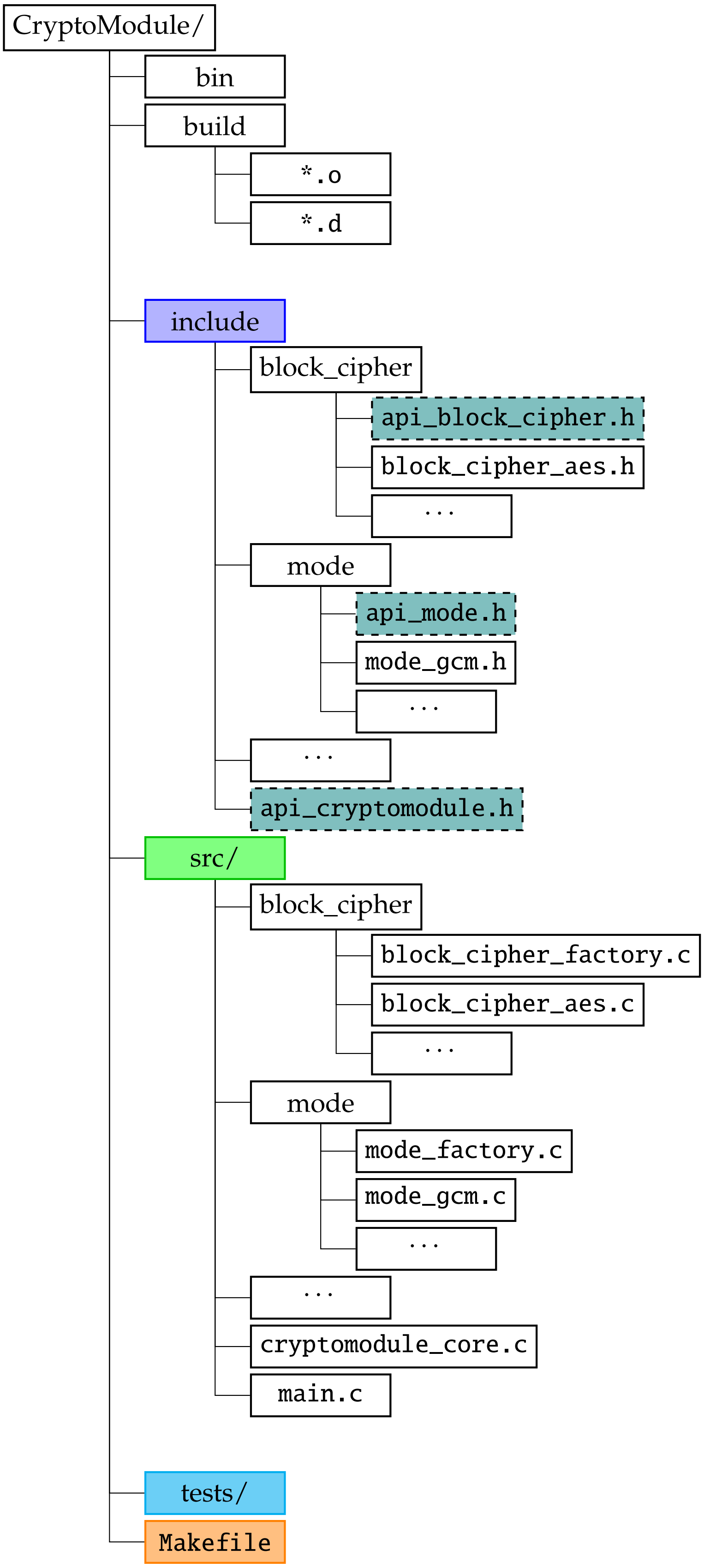
I have developed a cryptographic software module in the **C** language. This document provides a comprehensive guide to the design, implementation, and integration of cryptographic modules written in C (sometimes assembly).

Key Objectives:

- Describing the cryptographic primitives and algorithms
  (block ciphers, hash functions, 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.

Section	Description	Status
1.1	Directory layout & development environment	Drafted
1.2	Development Environment	Drafted
1.3	TBA	TBA

1.1 Directory Structure



1.2 Development Environment

- **Operating System: Linux Mint** (based on Debian and Ubuntu)

```
@>$ 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: GNU Compiler Collection** 11.4.0

```
@>$ 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: AMD Ryzen 7 5800X3D 8-Core Processor**

```
@>$ 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,

```
@>$ \valgrind --version
valgrind-3.18.1
```

  'gdb' for debugging,

```
@>$ gdb --version
GNU gdb (Ubuntu 12.1-0ubuntu1~22.04.2) 12.1
Copyright (C) 2022 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.
```

  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 efficient 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.

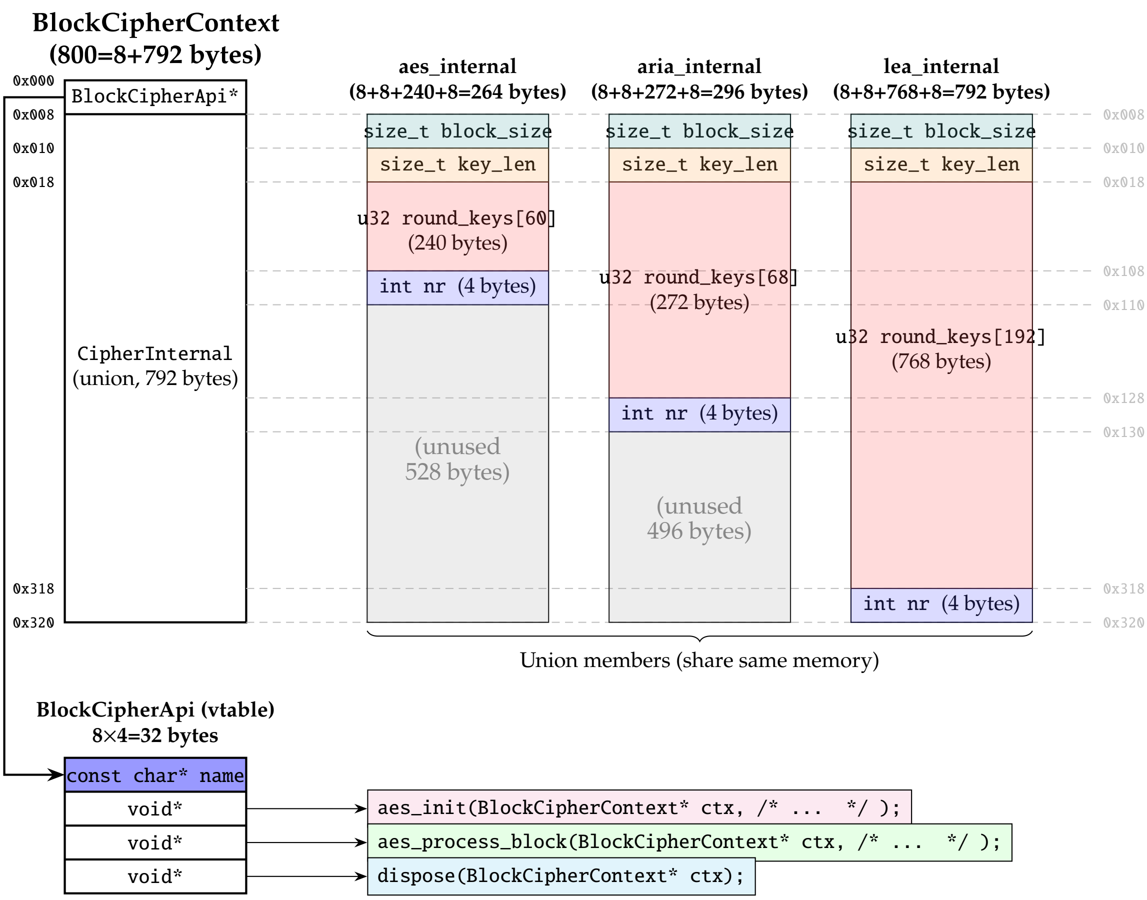
Alg.	$\boldsymbol{n}$ (bit)	$\boldsymbol{k}$ (bit)	**# of Rounds**	**RK Size** (bit)	**# of RKs**	**Total RK Size** (bit)
AES–128	128	128	10	128 (4-word)	11	1408 (44-word)
AES–192	128	192	12	128 (4-word)	13	1664 (52-word)
AES–256	128	256	14	128 (4-word)	15	1920 (60-word)
ARIA-128	128	128	12	128 (4-word)	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	192 (6-word)	24	4608 (144-word)
LEA-192	128	192	28	192 (6-word)	28	5376 (168-word)
LEA-256	128	256	32	192 (6-word)	32	6144 (192-word)

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

```c
typedef struct __BlockCipherApi__ {
   const char *name;
   void (*init)(BlockCipherContext* ctx, /* ... */);
   void (*process_block)(BlockCipherContext* ctx, /* ... */);
   void (*dispose)(BlockCipherContext* ctx);
} BlockCipherApi;

typedef union __CipherInternal__ {
   struct __aes_internal__ {
      /* ... */
   } aes_internal;
   struct __aria_internal__ {
      /* ... */
   } aria_internal;
   struct __lea_internal__ {
      /* ... */
   } lea_internal;
} CipherInternal;

typedef struct __BlockCipherContext__ {
   const BlockCipherApi *api;
   CipherInternal internal_data; /* Generic internal state for any cipher */
} BlockCipherContext;
```



Code 2.1: include/block_cipher/api_block_cipher.h

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

typedef struct __BlockCipherApi__ {
   const char *cipher_name; /* e.g. "AES" or "MyCipher" */

   block_cipher_status_t (*cipher_init)(
      BlockCipherContext* cipher_ctx,
      const u8* key,
      size_t key_len,
      size_t block_len,
      BlockCipherDirection dir);
   block_cipher_status_t (*cipher_process)(
      BlockCipherContext* cipher_ctx,
      const u8* in,
      u8* out,
      BlockCipherDirection dir);
   void (*cipher_dispose)(BlockCipherContext* cipher_ctx);
} BlockCipherApi;

typedef union __CipherInternal__ {
   struct __aes_internal__ {
      size_t block_size; /* Typically must be 16 for AES */
      size_t key_len; /* 16, 24, or 32 for AES-128/192/256 */
      /* max 60 for AES-256 */
      u32 round_keys[4 * (AES256_NUM_ROUNDS + 1)];
      int nr; /* e.g., 10 for AES-128, 12, or 14... */
   } aes_internal;
   struct __aria_internal__ {
      size_t block_size; /* Typically must be 16 for ARIA */
      size_t key_len; /* 16, 24, or 32 for ARIA-128/192/256 */
      /* max 68 for ARIA-256 */
      u32 round_keys[4 * (ARIA256_NUM_ROUNDS + 1)];
      int nr; /* e.g., 12 for ARIA-128, 14, or 16... */
   } aria_internal;
   struct __lea_internal__ {
      size_t block_size; /* Typically must be 16 for LEA */
      size_t key_len; /* 16, 24, or 32 for LEA-128/192/256 */
      /* max 192 for LEA-256 */
      u32 round_keys[6 * LEA256_NUM_ROUNDS];
      int nr; /* e.g., 24 for LEA-128, 28, or 32... */
   } lea_internal;
} CipherInternal;

struct __BlockCipherContext__ {
   const BlockCipherApi *cipher_api;
   CipherInternal cipher_state; /* Generic internal state for any cipher */
};
```

Subsection	Description	Status
2.1.1	AES (Advanced Encryption Standard)	Drafted
2.1.2	ARIA (Academy, Research Institute, and Agency)	Drafted
2.1.3	LEA (Lightweight Encryption Algorithm)	Drafted

2.1.1 AES (Advanced Encryption Standard)

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

Alg.	n (bit)	k (bit)	**# of Rounds**	**RK Size** (bit)	**# of RKs**	**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)

Code 2.2: include/block_cipher/block_cipher_aes.h

```c
/* Get the AES block cipher vtable. */
const BlockCipherApi* get_aes_api(void);

void aes_set_encrypt_key(const u8 *key, size_t bytes, u32 *rk);
void aes_set_decrypt_key(const u8 *key, size_t bytes, u32 *rk);
void aes_encrypt(const u8 *in, u8 *out, const u32 *rk, int r);
void aes_decrypt(const u8 *in, u8 *out, const u32 *rk, int r);
```

Code 2.3: src/block_cipher/block_cipher_aes.c

```c
/* Forward declarations of static functions. */
static block_cipher_status_t aes_init(
   BlockCipherContext *ctx,
   const u8 *key,
   size_t key_len,
   size_t block_len,
   BlockCipherDirection dir);
static block_cipher_status_t aes_process(
   BlockCipherContext *ctx,
   const u8 *in,
   u8 *out,
   BlockCipherDirection dir);
static void aes_dispose(BlockCipherContext *ctx);

/* The AES block cipher API. */
static const BlockCipherApi AES_API = {
   .cipher_name = "AES",
   .cipher_init = aes_init,
   .cipher_process = aes_process,
   .cipher_dispose = aes_dispose
};

/* Get the AES block cipher API. */
const BlockCipherApi *get_aes_api(void) { return &AES_API; }
```

2.1.2 ARIA (Academy, Research Institute, and Agency)

Table 2.3: Parameters of the Block Cipher ARIA (1-word = 32-bit)

Alg.	n (bit)	k (bit)	# of Rounds	RK Size (bit)	# of RKs	Total RK Size (bit)
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)

Code 2.4: include/block_cipher/block_cipher_aria.h

```
/* Get the ARIA block cipher vtable. */
const BlockCipherApi* get_aria_api(void);

void aria_set_encrypt_key(const u8 *key, size_t bytes, u32 *rk);
void aria_set_decrypt_key(const u8 *key, size_t bytes, u32 *rk);
void aria_encrypt(const u8 *in, u8 *out, const u32 *rk, int r);
void aria_decrypt(const u8 *in, u8 *out, const u32 *rk, int r);
```

Code 2.5: src/block_cipher/block_cipher_aria.c

```
/* Forward declarations of static functions. */
static block_cipher_status_t aria_init(
   BlockCipherContext *ctx,
   const u8 *key,
   size_t key_len,
   size_t block_len,
BlockCipherDirection dir);
static block_cipher_status_t aria_process(
   BlockCipherContext *ctx,
   const u8 *in,
   u8 *out,
   BlockCipherDirection dir);
static void aria_dispose(BlockCipherContext *ctx);

/* The ARIA block cipher API. */
static const BlockCipherApi ARIA_API = {
   .cipher_name = "ARIA",
   .cipher_init = aria_init,
   .cipher_process = aria_process,
   .cipher_dispose = aria_dispose
};
/* Get the ARIA block cipher API. */
const BlockCipherApi* get_aria_api(void) { return &ARIA_API; }
```

2.1.3 LEA (Lightweight Encryption Algorithm)

Table 2.4: Parameters of the Block Cipher LEA (1-word = 32-bit)

Alg.	n (bit)	k (bit)	**# of Rounds**	**RK Size** (bit)	**# of RKs**	**Total RK Size** (bit)
LEA–128	128	128	24	192 (6-word)	24	4608 (144-word)
LEA–192	128	192	28	192 (6-word)	28	5376 (168-word)
LEA–256	128	256	32	192 (6-word)	32	6144 (192-word)

Code 2.6: include/block_cipher/block_cipher_aria.h

```c
/* Get the LEA block cipher vtable. */
const BlockCipherApi* get_lea_api(void);

void lea_set_encrypt_key(const u8 *key, size_t bytes, u32 *rk);
void lea_set_decrypt_key(const u8 *key, size_t bytes, u32 *rk);
void lea_encrypt(const u8 *in, u8 *out, const u32 *rk, int r);
void lea_decrypt(const u8 *in, u8 *out, const u32 *rk, int r);
```

Code 2.7: src/block_cipher/block_cipher_aria.c

```c
/* Forward declarations of static functions. */
static block_cipher_status_t lea_init(
   BlockCipherContext *ctx,
   const u8 *key,
   size_t key_len,
   size_t block_len,
   BlockCipherDirection dir);
static block_cipher_status_t lea_process(
   BlockCipherContext *ctx,
   const u8 *in,
   u8 *out,
   BlockCipherDirection dir);
static void lea_dispose(BlockCipherContext *ctx);

/* The LEA block cipher API. */
static const BlockCipherApi LEA_API = {
   .cipher_name = "LEA",
   .cipher_init = lea_init,
   .cipher_process = lea_process,
   .cipher_dispose = lea_dispose
};
/* Get the LEA block cipher API. */
const BlockCipherApi *get_lea_api(void) { return &LEA_API; }
```

2.2 Modes of Operation

```c
typedef struct __ModeOfOperationApi__ {
   const char *name;
   void (*init)( /* ... */ );
   void (*process)( /* ... */ );
   void (*dispose)( /* ... */ );
} ModeOfOperationApi;

typedef union __ModeInternal__ {
   struct __cbc_internal__ {
      /* ... */
   } cbc_internal;
   struct __ctr_internal__ {
      /* ... */
   } ctr_internal;
   struct __gcm_internal__ {
      /* ... */
   } gcm_internal;
   struct __ecb_internal__ {
      /* ... */
   } ecb_internal;

} ModeInternal;

typedef struct __ModeOfOperationContext__ {
   const ModeOfOperationApi *api; // Pointer to the mode API
   BlockCipherContext cipher_ctx; // Block cipher context
   ModeInternal internal_data; // Internal state for the mode
} ModeOfOperationContext;
```

2.2.1 Electronic Codebook (ECB)

TBA

2.2.2 Cipher Block Chaining (CBC)

TBA

2.2.3 Counter (CTR)

TBA

2.3 Galois / Counter Mode (GCM)

- **Specification:** [NIST SP 800-38D]
  `https://csrc.nist.gov/pubs/sp/800/38/d/final`
- **Implementation:**
  `https://github.com/openssl/openssl/blob/master/crypto/modes/gcm128.c`

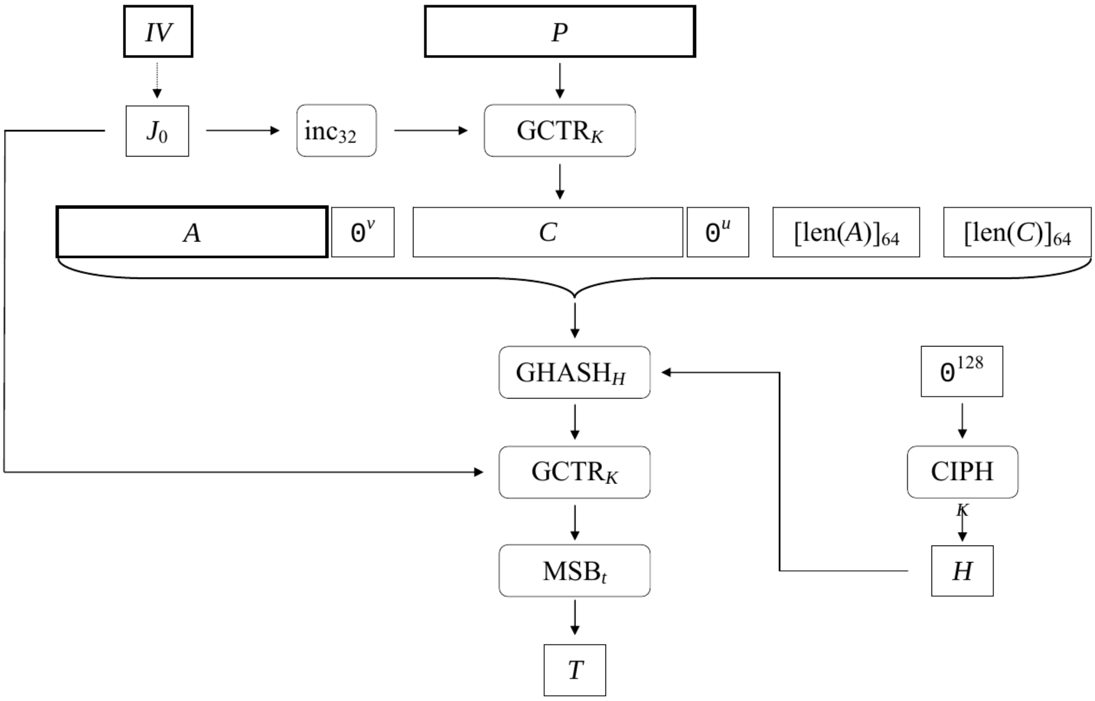


Figure 2.1: Illustration of GCM-AE$_K(IV, P, A) = (C, T)$ in NIST SP 800-38D (p16).

Algorithm 1: GCM Authenticated Encryption GCM-AE$_K(IV, P, A)$

Input:

-

Key K, IV, Plaintext P, AAD A, tag-length t

Output: Ciphertext C, Authentication Tag T

$H \leftarrow \mathrm{CIPH}_K(0^{128})$;
if $|\,\mathrm{IV}\,| = 96$ **then**
  $J_0 \leftarrow \mathrm{IV} \,\|\, 0^{31} \,\|\, 1$;
else
  $s \leftarrow 128 \cdot \lceil |\,\mathrm{IV}\,|/128 \rceil - |\,\mathrm{IV}\,|$;
  $J_0 \leftarrow \mathrm{GHASH}_H\big(\mathrm{IV} \,\|\, 0^s \,\|\, [\![\,\mathrm{IV}\,]\!]_{64}\big)$;
end
$C \leftarrow \mathrm{GCTR}_K\big(\mathrm{inc}_{32}(J_0),\, P\big)$;
$v \leftarrow 128 \cdot \lceil |A|/128 \rceil - |A|$;
$u \leftarrow 128 \cdot \lceil |C|/128 \rceil - |C|$;
$S \leftarrow \mathrm{GHASH}_H\big(A \,\|\, 0^v \,\|\, C \,\|\, 0^u \,\|\, [\![A]\!]_{64} \,\|\, [\![C]\!]_{64}\big)$;
$T \leftarrow \mathrm{MSB}_t\big(\mathrm{GCTR}_K(J_0,\, S)\big)$;
return (C, T);

Algorithm 2: GCM Authenticated Decryption (GCM-AD_K)

Input: Key K, IV, Ciphertext C, AAD A, Tag T, tag-length t
Output: Plaintext P or FAIL

if $|T| \neq t$ ***or*** *lengths not supported* **then**
  return *FAIL*
end
$H \leftarrow \mathrm{CIPH}_K(0^{128})$;
if $|\,\mathrm{IV}\,| = 96$ **then**
  $J_0 \leftarrow \mathrm{IV} \,\|\, 0^{31} \,\|\, 1$;
else
  $s \leftarrow 128 \cdot \lceil |\,\mathrm{IV}\,|/128 \rceil - |\,\mathrm{IV}\,|$;
  $J_0 \leftarrow \mathrm{GHASH}_H\big(\mathrm{IV} \,\|\, 0^s \,\|\, [\![\,\mathrm{IV}\,]\!]_{64}\big)$;
end
$P \leftarrow \mathrm{GCTR}_K\big(\mathrm{inc}_{32}(J_0),\, C\big)$;
$v \leftarrow 128 \cdot \lceil |A|/128 \rceil - |A|$;
  $u \leftarrow 128 \cdot \lceil |C|/128 \rceil - |C|$;
$S \leftarrow \mathrm{GHASH}_H\big(A \,\|\, 0^v \,\|\, C \,\|\, 0^u \,\|\, [\![A]\!]_{64} \,\|\, [\![C]\!]_{64}\big)$;
$T' \leftarrow \mathrm{MSB}_t\big(\mathrm{GCTR}_K(J_0,\, S)\big)$;
if $T' = T$ **then**
  return P;
end
else
  return *FAIL*;
end

2.3.1 Data Structure

```c
#if defined(__x86_64) || defined(__x86_64__)
# define BSWAP8(x) ({ u64 ret_=(x); \
                asm ("bswapq %0"\
                : "+r"(ret_)); ret_; })
# define BSWAP4(x) ({ u32 ret_=(x); \
                asm ("bswapl %0" \
                : "+r"(ret_)); ret_; })
#endif

#if defined(BSWAP4) && !defined(STRICT_ALIGNMENT)
# define GETU32(p) BSWAP4(*(const u32 *)(p))
# define PUTU32(p,v) *(u32 *)(p) = BSWAP4(v)
#else
# define GETU32(p) ((u32)(p)[0]<<24|(u32)(p)[1]<<16|(u32)(p)[2]<<8|(u32)(p)[3])
# define PUTU32(p,v)
    ↪ ((p)[0]=(u8)((v)>>24),(p)[1]=(u8)((v)>>16),(p)[2]=(u8)((v)>>8),(p)[3]=(u8)(v))
#endif

/*- GCM definitions */ typedef struct {
   u64 hi, lo;
} u128;


typedef void (*gcm_init_fn)(u128 Htable[16], const u64 H[2]);
typedef void (*gcm_ghash_fn)(u64 Xi[2], const u128 Htable[16], const u8 *inp, size_t
    ↪ len);
typedef void (*gcm_gmult_fn)(u64 Xi[2], const u128 Htable[16]);
struct gcm_funcs_st {
   gcm_init_fn ginit;
   gcm_ghash_fn ghash;
   gcm_gmult_fn gmult;
};

struct gcm128_context {
   /* Following 6 names follow names in GCM specification */
   union {
      u64 u[2];
      u32 d[4];
      u8 c[16];
      size_t t[16 / sizeof(size_t)];
   } Yi, EKi, EK0, len, Xi, H;
   /*
   * Relative position of Yi, EKi, EK0, len, Xi, H and pre-computed Htable is
   * used in some assembler modules, i.e. don't change the order!
   */
   u128 Htable[16];
   struct gcm_funcs_st funcs;
   unsigned int mres, ares;
   block128_f block;
   void *key;
};
```

2.3.2 title

2.4 Random Number Generator

TBA

2.5 Hash Functions

2.5.1 SHA-2 Algorithms

TBA

2.5.2 SHA-3 Algorithms

TBA

2.5.3 Lightweight Secure Hash (LSH)

TBA

2.6 Message Authentication Codes

TBA

2.7 Key Derivation Functions

TBA

2.8 Diffie-Hellman Key Exchange

TBA

2.9 Signature Algorithms

TBA

Chapter 3

Build and Integration

3.1 Makefile Configuration and Overview

This section describes the build system for the CryptoModule demo, driven by a single GNU Makefile. It covers compiler settings, directory layout, source discovery, and all available targets.

3.1.1 Compiler, Flags, and Directories

```
# Compiler and flags
CC := gcc
CFLAGS := -std=c99 -g -O2 -Wall -Wextra -I. -Iinclude -Isrc

# Executable name
TARGET := cryptomodule-demo

# Output directories
OBJ_DIR := build
BIN_DIR := bin
```

- `gcc` in C99 mode, with debug symbols (`-g`) and optimization (`-O2`).
- Warnings enabled (`-Wall -Wextra`), include paths set for project headers.
- Object files placed under `build/`, preserving the `src/` subdirectory structure; final binary in `bin/`.

3.1.2 Automatic Source and Object Discovery

```
# Find all .c files in src/ recursively
SRCS := $(shell find src -name '*.c')

# Map src/foo.c -> build/foo.o
OBJS := $(patsubst src/%.c,$(OBJ_DIR)/%.o,$(SRCS))
```

3.1.3 Usage Examples

```
################################################################################
# 1) build : compile + link
################################################################################
build: $(BIN_DIR)/$(TARGET)

# Link step: gather all objects into a single executable
$(BIN_DIR)/$(TARGET): $(OBJS)
	@echo "[LINK] Linking objects to create $@"
	@mkdir -p $(BIN_DIR)
	$(CC) $(CFLAGS) $^ -o $@
# Compile step: For each .c -> .o
$(OBJ_DIR)/%.o: src/%.c
	@echo "[CC] Compiling $< into $@"
	@mkdir -p $(dir $@)
	$(CC) $(CFLAGS) -c $< -o $@
################################################################################
# 2) run : run the resulting binary
################################################################################
run: build
	@echo "[RUN] Running $(BIN_DIR)/$(TARGET)"
	@./$(BIN_DIR)/$(TARGET)

################################################################################
# 3) clean : remove build artifacts
################################################################################
clean:
@echo "[CLEAN] Removing build artifacts..."
	rm -rf $(OBJ_DIR) $(BIN_DIR)
	@echo "[CLEAN] Removing *.req and *.rsp files in testvectors folder..."
	find testvectors -type f \( -name '*.req' -o -name '*.rsp' \) -delete

################################################################################
# 4) rebuild : clean + build
################################################################################
rebuild: clean build

################################################################################
# 5) valgrind : run the binary under Valgrind for memory checking
################################################################################
valgrind: build
	@echo "[VALGRIND] Running Valgrind..."
	valgrind --leak-check=full ./$(BIN_DIR)/$(TARGET)
```

make build Compile (`.c → .o`) and link (`.o → executable`).

make run Build if necessary, then execute `bin/cryptomodule-demo`.

make clean Remove `build/`, `bin/`, and any `*.req`/`*.rsp` in `testvectors/`.

make rebuild Alias for `clean` followed by `build`.

make valgrind Build, then run under Valgrind for memory-leak checks.

3.2 Example: Main Function for Block-Cipher KATs

Code 3.1: Invoke known-answer tests for AES block ciphers

```
int main(void) {
	KAT_TEST_BLOCKCIPHER(BLOCK_CIPHER_AES128);
	KAT_TEST_BLOCKCIPHER(BLOCK_CIPHER_AES192);
	KAT_TEST_BLOCKCIPHER(BLOCK_CIPHER_AES256);
	return 0;
}
```

```
@>$ make rebuild
@>$ make run
```

```
~/Desktop/2025/CryptoModule  main !48 ?6  make run
[RUN] Running bin/cryptomodule-demo
--------------------------------- KAT TEST for AES-128 ---------------------------------
[REQ] ? Creating request file : ./testvectors/block_cipher_tv/nist_aes/ECB_AES128_KAT.req
[REQ] ! Created request file  : ./testvectors/block_cipher_tv/nist_aes/ECB_AES128_KAT.req
[RSP] ? Creating response file: ./testvectors/block_cipher_tv/nist_aes/ECB_AES128_KAT.rsp
[RSP] ! Created response file : ./testvectors/block_cipher_tv/nist_aes/ECB_AES128_KAT.rsp

[PATH] Test vector file : ./testvectors/block_cipher_tv/nist_aes/ECB_AES128_KAT.fax
[PATH] Request file     : ./testvectors/block_cipher_tv/nist_aes/ECB_AES128_KAT.req
[PATH] Response file    : ./testvectors/block_cipher_tv/nist_aes/ECB_AES128_KAT.rsp


[==================================================] 100% (512/512)

[*] Test Results:
- Total vectors : 512
- Passed vectors: 512
[O] Result: PASSED

----------------------------------------- END -----------------------------------------


--------------------------------- KAT TEST for AES-192 ---------------------------------
[REQ] ? Creating request file : ./testvectors/block_cipher_tv/nist_aes/ECB_AES192_KAT.req
[REQ] ! Created request file  : ./testvectors/block_cipher_tv/nist_aes/ECB_AES192_KAT.req
[RSP] ? Creating response file: ./testvectors/block_cipher_tv/nist_aes/ECB_AES192_KAT.rsp
[RSP] ! Created response file : ./testvectors/block_cipher_tv/nist_aes/ECB_AES192_KAT.rsp

[PATH] Test vector file : ./testvectors/block_cipher_tv/nist_aes/ECB_AES192_KAT.fax
[PATH] Request file     : ./testvectors/block_cipher_tv/nist_aes/ECB_AES192_KAT.req
[PATH] Response file    : ./testvectors/block_cipher_tv/nist_aes/ECB_AES192_KAT.rsp


[==================================================] 100% (640/640)

[*] Test Results:
- Total vectors : 640
- Passed vectors: 640
[O] Result: PASSED

----------------------------------------- END -----------------------------------------


--------------------------------- KAT TEST for AES-256 ---------------------------------
[REQ] ? Creating request file : ./testvectors/block_cipher_tv/nist_aes/ECB_AES256_KAT.req
[REQ] ! Created request file  : ./testvectors/block_cipher_tv/nist_aes/ECB_AES256_KAT.req
[RSP] ? Creating response file: ./testvectors/block_cipher_tv/nist_aes/ECB_AES256_KAT.rsp
[RSP] ! Created response file : ./testvectors/block_cipher_tv/nist_aes/ECB_AES256_KAT.rsp

[PATH] Test vector file : ./testvectors/block_cipher_tv/nist_aes/ECB_AES256_KAT.fax
[PATH] Request file     : ./testvectors/block_cipher_tv/nist_aes/ECB_AES256_KAT.req
[PATH] Response file    : ./testvectors/block_cipher_tv/nist_aes/ECB_AES256_KAT.rsp


[==================================================] 100% (768/768)

[*] Test Results:
- Total vectors : 768
- Passed vectors: 768
[O] Result: PASSED

----------------------------------------- END -----------------------------------------
```

Bibliography

Appendices

TBA