
Cryptographic-Module Source-Code Development Manual

Design, Implementation, and Integration of Cryptography Modules

Secure, Efficient, High-Performance Cryptographic Software Modules

KMU

Ji, Yong-hyeon

hacker3740@kookmin.ac.kr

Department of Cyber Security
Kookmin University

May 28, 2025

Contents

1	Project Overview	2
1.1	Directory Structure	3
1.2	Development Environment	4
2	Cryptographic Software Module	5
2.1	Block Cipher	5
2.1.1	AES (Advanced Encryption Standard)	8
2.1.2	ARIA (Academy, Research Institute, and Agency)	9
2.1.3	LEA (Lightweight Encryption Algorithm)	10
2.2	Modes of Operation	11
2.2.1	Electronic Codebook (ECB)	11
2.2.2	Cipher Block Chaining (CBC)	11
2.2.3	Counter (CTR)	11
2.3	Galois / Counter Mode (GCM)	12
2.3.1	Data Structure	14
2.3.2	title	15
2.4	Random Number Generator	16
2.5	Hash Functions	16
2.5.1	SHA-2 Algorithms	16
2.5.2	SHA-3 Algorithms	16
2.5.3	Lightweight Secure Hash (LSH)	16
2.6	Message Authentication Codes	16
2.7	Key Derivation Functions	16
2.8	Diffie-Hellman Key Exchange	16
2.9	Signature Algorithms	16
3	Build and Integration	17
3.1	Makefile Configuration and Overview	17
3.1.1	Compiler, Flags, and Directories	17
3.1.2	Automatic Source and Object Discovery	17
3.1.3	Usage Examples	18
3.2	Example: Main Function for Block-Cipher KATs	19

Chapter 1

Project Overview

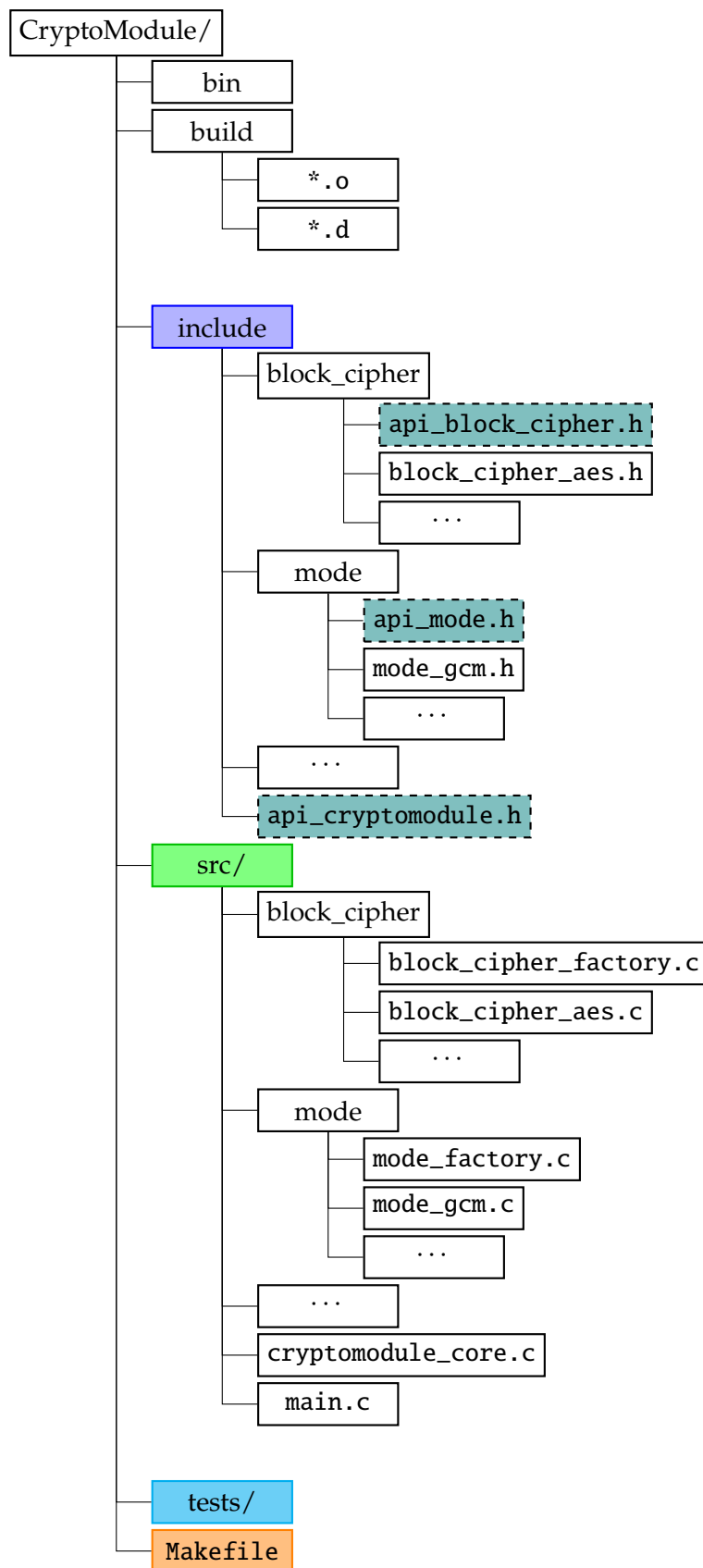
I have developed a cryptographic software module in the C language. This document provides a comprehensive guide to the design, implementation, and integration of cryptographic modules written in C (sometimes assembly).

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 $\mathcal{C} = \{0, 1\}^n$ be the set of all n -bit ciphertext blocks.

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

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

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

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

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

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

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.

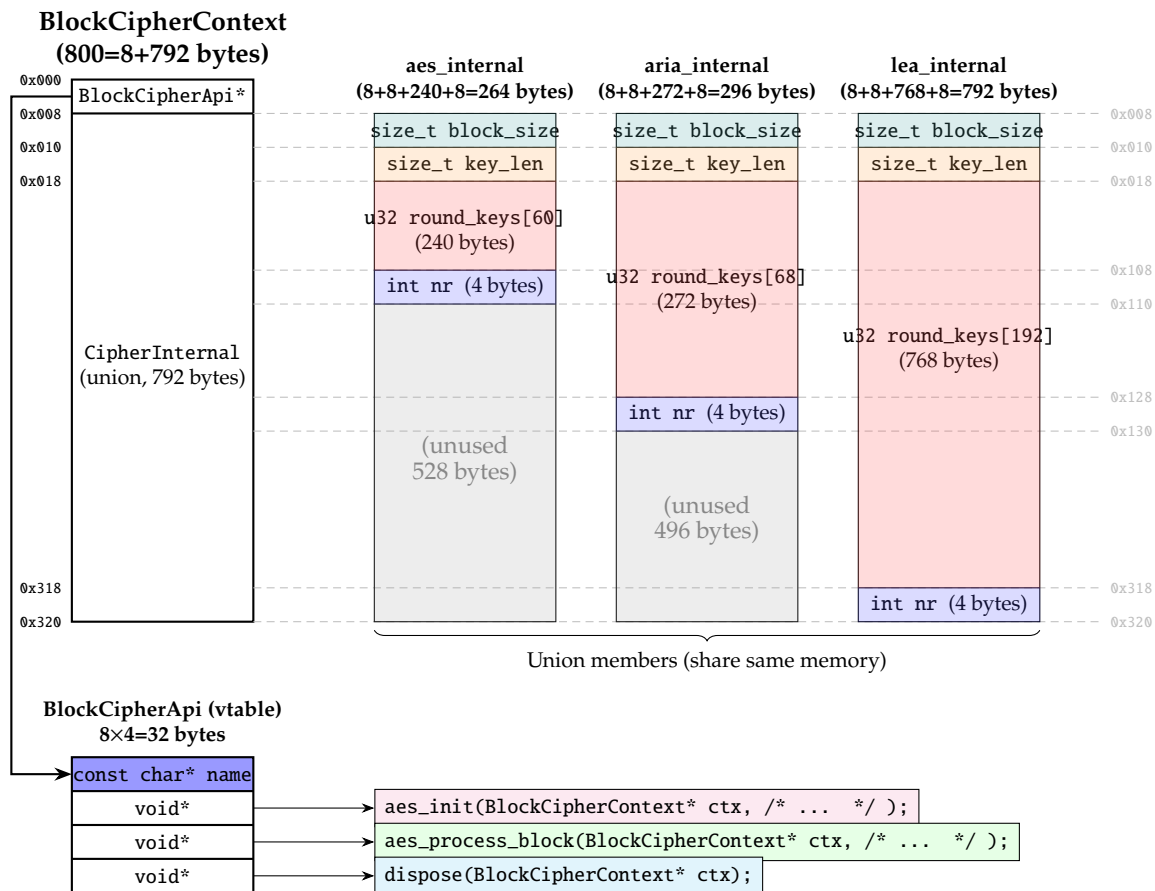
Alg.	n (bit)	k (bit)	# of Rounds	RK Size (bit)	# of RKs	Total RK Size (bit)
AES-128	128	128	10	128 (4-word)	11	1408 (44-word)
AES-192	128	192	12	128 (4-word)	13	1664 (52-word)
AES-256	128	256	14	128 (4-word)	15	1920 (60-word)
ARIA-128	128	128	12	128 (4-word)	13	1664 (52-word)
ARIA-192	128	192	14	128 (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.

```

1 typedef struct __BlockCipherApi__ {
2     const char *name;
3     void (*init)(BlockCipherContext* ctx, /* ... */);
4     void (*process_block)(BlockCipherContext* ctx, /* ... */);
5     void (*dispose)(BlockCipherContext* ctx);
6 } BlockCipherApi;
7
8 typedef union __CipherInternal__ {
9     struct __aes_internal__ {
10         /* ... */
11     } aes_internal;
12     struct __aria_internal__ {
13         /* ... */
14     } aria_internal;
15     struct __lea_internal__ {
16         /* ... */
17     } lea_internal;
18 } CipherInternal;
19
20 typedef struct __BlockCipherContext__ {
21     const BlockCipherApi *api;
22     CipherInternal internal_data; /* Generic internal state for any cipher */
23 } BlockCipherContext;

```



Code 2.1: include/block_cipher/api_block_cipher.h

```

1  /* Forward declaration for the context. */
2  typedef struct __BlockCipherContext__ BlockCipherContext;
3
4  typedef struct __BlockCipherApi__ {
5      const char *cipher_name; /* e.g. "AES" or "MyCipher" */
6
7      block_cipher_status_t (*cipher_init)(
8          BlockCipherContext* cipher_ctx,
9          const u8* key,
10         size_t key_len,
11         size_t block_len,
12         BlockCipherDirection dir);
13     block_cipher_status_t (*cipher_process)(
14         BlockCipherContext* cipher_ctx,
15         const u8* in,
16         u8* out,
17         BlockCipherDirection dir);
18     void (*cipher_dispose)(BlockCipherContext* cipher_ctx);
19 } BlockCipherApi;
20
21 typedef union __CipherInternal__ {
22     struct __aes_internal__ {
23         size_t block_size; /* Typically must be 16 for AES */
24         size_t key_len; /* 16, 24, or 32 for AES-128/192/256 */
25         /* max 60 for AES-256 */
26         u32 round_keys[4 * (AES256_NUM_ROUNDS + 1)];
27         int nr; /* e.g., 10 for AES-128, 12, or 14... */
28     } aes_internal;
29     struct __aria_internal__ {
30         size_t block_size; /* Typically must be 16 for ARIA */
31         size_t key_len; /* 16, 24, or 32 for ARIA-128/192/256 */
32         /* max 68 for ARIA-256 */
33         u32 round_keys[4 * (ARIA256_NUM_ROUNDS + 1)];
34         int nr; /* e.g., 12 for ARIA-128, 14, or 16... */
35     } aria_internal;
36     struct __lea_internal__ {
37         size_t block_size; /* Typically must be 16 for LEA */
38         size_t key_len; /* 16, 24, or 32 for LEA-128/192/256 */
39         /* max 192 for LEA-256 */
40         u32 round_keys[6 * LEA256_NUM_ROUNDS];
41         int nr; /* e.g., 24 for LEA-128, 28, or 32... */
42     } lea_internal;
43 } CipherInternal;
44
45 struct __BlockCipherContext__ {
46     const BlockCipherApi *cipher_api;
47     CipherInternal cipher_state; /* Generic internal state for any cipher */
48 };

```

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

```

1  /* Get the AES block cipher vtable. */
2  const BlockCipherApi* get_aes_api(void);
3
4  void aes_set_encrypt_key(const u8 *key, size_t bytes, u32 *rk);
5  void aes_set_decrypt_key(const u8 *key, size_t bytes, u32 *rk);
6  void aes_encrypt(const u8 *in, u8 *out, const u32 *rk, int r);
7  void aes_decrypt(const u8 *in, u8 *out, const u32 *rk, int r);

```

Code 2.3: src/block_cipher/block_cipher_aes.c

```

1  /* Forward declarations of static functions. */
2  static block_cipher_status_t aes_init(
3      BlockCipherContext *ctx,
4      const u8 *key,
5      size_t key_len,
6      size_t block_len,
7      BlockCipherDirection dir);
8  static block_cipher_status_t aes_process(
9      BlockCipherContext *ctx,
10     const u8 *in,
11     u8 *out,
12     BlockCipherDirection dir);
13 static void aes_dispose(BlockCipherContext *ctx);
14
15 /* The AES block cipher API. */
16 static const BlockCipherApi AES_API = {
17     .cipher_name = "AES",
18     .cipher_init = aes_init,
19     .cipher_process = aes_process,
20     .cipher_dispose = aes_dispose
21 };
22
23 /* Get the AES block cipher API. */
24 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

```

1 /* Get the ARIA block cipher vtable. */
2 const BlockCipherApi* get_aria_api(void);
3
4 void aria_set_encrypt_key(const u8 *key, size_t bytes, u32 *rk);
5 void aria_set_decrypt_key(const u8 *key, size_t bytes, u32 *rk);
6 void aria_encrypt(const u8 *in, u8 *out, const u32 *rk, int r);
7 void aria_decrypt(const u8 *in, u8 *out, const u32 *rk, int r);

```

Code 2.5: src/block_cipher/block_cipher_aria.c

```

1 /* Forward declarations of static functions. */
2 static block_cipher_status_t aria_init(
3     BlockCipherContext *ctx,
4     const u8 *key,
5     size_t key_len,
6     size_t block_len,
7     BlockCipherDirection dir);
8 static block_cipher_status_t aria_process(
9     BlockCipherContext *ctx,
10    const u8 *in,
11    u8 *out,
12    BlockCipherDirection dir);
13 static void aria_dispose(BlockCipherContext *ctx);
14
15 /* The ARIA block cipher API. */
16 static const BlockCipherApi ARIA_API = {
17     .cipher_name = "ARIA",
18     .cipher_init = aria_init,
19     .cipher_process = aria_process,
20     .cipher_dispose = aria_dispose
21 };
22 /* Get the ARIA block cipher API. */
23 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

```

1  /* Get the LEA block cipher vtable. */
2  const BlockCipherApi* get_lea_api(void);
3
4  void lea_set_encrypt_key(const u8 *key, size_t bytes, u32 *rk);
5  void lea_set_decrypt_key(const u8 *key, size_t bytes, u32 *rk);
6  void lea_encrypt(const u8 *in, u8 *out, const u32 *rk, int r);
7  void lea_decrypt(const u8 *in, u8 *out, const u32 *rk, int r);

```

Code 2.7: src/block_cipher/block_cipher_aria.c

```

1  /* Forward declarations of static functions. */
2  static block_cipher_status_t lea_init(
3      BlockCipherContext *ctx,
4      const u8 *key,
5      size_t key_len,
6      size_t block_len,
7      BlockCipherDirection dir);
8  static block_cipher_status_t lea_process(
9      BlockCipherContext *ctx,
10     const u8 *in,
11     u8 *out,
12     BlockCipherDirection dir);
13 static void lea_dispose(BlockCipherContext *ctx);
14
15 /* The LEA block cipher API. */
16 static const BlockCipherApi LEA_API = {
17     .cipher_name = "LEA",
18     .cipher_init = lea_init,
19     .cipher_process = lea_process,
20     .cipher_dispose = lea_dispose
21 };
22 /* Get the LEA block cipher API. */
23 const BlockCipherApi *get_lea_api(void) { return &LEA_API; }

```

2.2 Modes of Operation

```

1 typedef struct __ModeOfOperationApi__ {
2     const char *name;
3     void (*init)( /* ... */ );
4     void (*process)( /* ... */ );
5     void (*dispose)( /* ... */ );
6 } ModeOfOperationApi;
7
8 typedef union __ModeInternal__ {
9     struct __cbc_internal__ {
10         /* ... */
11     } cbc_internal;
12     struct __ctr_internal__ {
13         /* ... */
14     } ctr_internal;
15     struct __gcm_internal__ {
16         /* ... */
17     } gcm_internal;
18     struct __ecb_internal__ {
19         /* ... */
20     } ecb_internal;
21 } ModeInternal;
22
23
24 typedef struct __ModeOfOperationContext__ {
25     const ModeOfOperationApi *api; // Pointer to the mode API
26     BlockCipherContext cipher_ctx; // Block cipher context
27     ModeInternal internal_data; // Internal state for the mode
28 } 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>

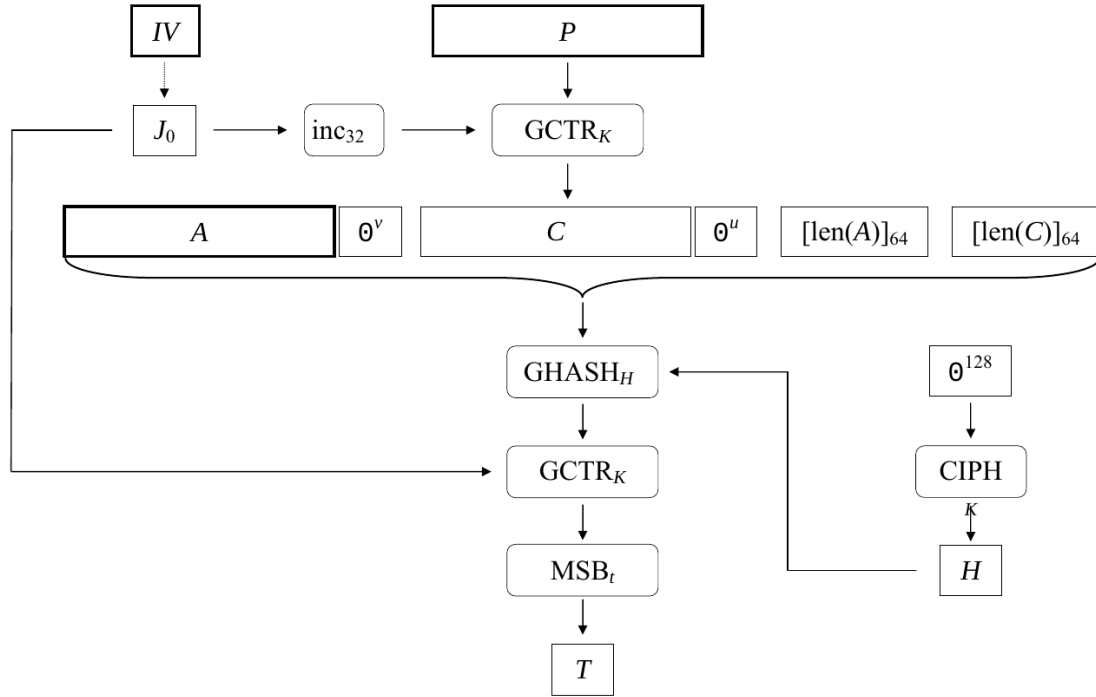


Figure 2.1: Illustration of $\text{GCM-AE}_K(IV, P, A) = (C, T)$ in NIST SP 800-38D (p16).

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

Input:

•

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

Output: Ciphertext C , Authentication Tag T

```

1  $H \leftarrow \text{CIPH}_K(\mathbf{0}^{128});$ 
2 if  $|IV| = 96$  then
3    $J_0 \leftarrow IV \parallel \mathbf{0}^{31} \parallel 1;$ 
4 else
5    $s \leftarrow 128 \cdot \lceil |IV|/128 \rceil - |IV|;$ 
6    $J_0 \leftarrow \text{GHASH}_H(IV \parallel \mathbf{0}^s \parallel IV_{64});$ 
7 end
8  $C \leftarrow \text{GCTR}_K(\text{inc}_{32}(J_0), P);$ 
9  $v \leftarrow 128 \cdot \lceil |A|/128 \rceil - |A|;$ 
10  $u \leftarrow 128 \cdot \lceil |C|/128 \rceil - |C|;$ 
11  $S \leftarrow \text{GHASH}_H(A \parallel \mathbf{0}^v \parallel C \parallel \mathbf{0}^u \parallel A_{64} \parallel C_{64});$ 
12  $T \leftarrow \text{MSB}_t(\text{GCTR}_K(J_0, S));$ 
13 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

```

1 if  $|T| \neq t$  or lengths not supported then
2   | return FAIL
3 end
4  $H \leftarrow \text{CIPH}_K(0^{128})$ ;
5 if  $|IV| = 96$  then
6   |  $J_0 \leftarrow IV \parallel 0^{31} \parallel 1$ ;
7 else
8   |  $s \leftarrow 128 \cdot \lceil |IV|/128 \rceil - |IV|$ ;
9   |  $J_0 \leftarrow \text{GHASH}_H(IV \parallel 0^s \parallel \llbracket IV \rrbracket_{64})$ ;
10 end
11  $P \leftarrow \text{GCTR}_K(\text{inc}_{32}(J_0), C)$ ;
12  $v \leftarrow 128 \cdot \lceil |A|/128 \rceil - |A|$ ;
13  $u \leftarrow 128 \cdot \lceil |C|/128 \rceil - |C|$ ;
14  $S \leftarrow \text{GHASH}_H(A \parallel 0^v \parallel C \parallel 0^u \parallel \llbracket A \rrbracket_{64} \parallel \llbracket C \rrbracket_{64})$ ;
15  $T' \leftarrow \text{MSB}_t(\text{GCTR}_K(J_0, S))$ ;
16 if  $T' = T$  then
17   | return  $P$ ;
18 end
19 else
20   | return FAIL;
21 end

```

2.3.1 Data Structure

```

1 #if defined(__x86_64) || defined(__x86_64__)
2 # define BSWAP8(x) ({ u64 ret_=(x); \
3     asm ("bswapq %0" \
4         : "+r"(ret_)); ret_; })
5 # define BSWAP4(x) ({ u32 ret_=(x); \
6     asm ("bswapl %0" \
7         : "+r"(ret_)); ret_; })
8 #endif
9
10 #if defined(BSWAP4) && !defined(STRICT_ALIGNMENT)
11 # define GETU32(p) BSWAP4(*(const u32 *) (p))
12 # define PUTU32(p,v) *(u32 *) (p) = BSWAP4(v)
13 #else
14 # define GETU32(p) ((u32) (p)[0]<<24|(u32) (p)[1]<<16|(u32) (p)[2]<<8|(u32) (p)[3])
15 # define PUTU32(p,v)
16     ↪ ((p)[0]=(u8)((v)>>24),(p)[1]=(u8)((v)>>16),(p)[2]=(u8)((v)>>8),(p)[3]=(u8)(v))
17 #endif
18
19 /*- GCM definitions */ typedef struct {
20     u64 hi, lo;
21 } u128;
22
23 typedef void (*gcm_init_fn)(u128 Htable[16], const u64 H[2]);
24 typedef void (*gcm_ghash_fn)(u64 Xi[2], const u128 Htable[16], const u8 *inp, size_t
25     ↪ len);
26 typedef void (*gcm_gmult_fn)(u64 Xi[2], const u128 Htable[16]);
27 struct gcm_funcs_st {
28     gcm_init_fn ginit;
29     gcm_ghash_fn ghash;
30     gcm_gmult_fn gmult;
31 };
32
33 struct gcm128_context {
34     /* Following 6 names follow names in GCM specification */
35     union {
36         u64 u[2];
37         u32 d[4];
38         u8 c[16];
39         size_t t[16 / sizeof(size_t)];
40     } Yi, EK_i, EK_0, len, Xi, H;
41     /*
42      * Relative position of Yi, EK_i, EK_0, len, Xi, H and pre-computed Htable is
43      * used in some assembler modules, i.e. don't change the order!
44      */
45     u128 Htable[16];
46     struct gcm_funcs_st funcs;
47     unsigned int mres, ares;
48     block128_f block;
49     void *key;
50 };

```

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

```
1 int main(void) {  
2     KAT_TEST_BLOCKCIPHER(BLOCK_CIPHER_AES128);  
3     KAT_TEST_BLOCKCIPHER(BLOCK_CIPHER_AES192);  
4     KAT_TEST_BLOCKCIPHER(BLOCK_CIPHER_AES256);  
5     return 0;  
6 }
```

```
1 @>$ make rebuild  
2 @>$ make run
```

```
● ~/Desktop/2025/CryptoModule main 148 ?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
[0] 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
[0] 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
[0] Result: PASSED

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

Bibliography

Appendices

TBA