CryptoModule Development Manual

Cryptographic Algorithm / Ji, Yong-hyeon

April 5, 2025

Contents

1 Introduction

This manual describes how to develop, build, and maintain the `CryptoModule`—a C-based cryptographic library that provides various cryptographic algorithms, modes of operation, and utility functions. This manual is intended for developers who plan to modify, extend, or integrate the library.

1.1 Project Overview

`CryptoModule` is designed to be modular and easily extendable. It includes:

- Block Ciphers (AES, ARIA, LEA)
- Modes of Operation (ECB, CBC, CTR, GCM)
- Random Number Generators (CTR-DRBG)
- Hash Functions (SHA2, SHA3, LSH)
- Message Authentication Codes (HMAC)
- Key Derivation Functions (PBKDF)
- Key Setup (EC / DH)
- Signatures (RSAPSS, ECDSA, EC-KCDSA)

2 Directory Structure

To keep the repository consistent and intuitive, the code is subdivided by cryptographic category. The typical structure:

2.1 Key Folders

`include/cryptomodule/` Public-facing headers, grouped by cryptographic function (block, mode, rng, etc.). Clients of the library typically include these header files.

`src/` Implementation (.c) files for each algorithm or mode.

`tests/` Minimal test suite or unit test code. Each `test_*.c` can be compiled and run to validate correctness.

`Makefile` A simple top-level build system that compiles objects and creates the static library `libcryptomodule.a`, plus test executables.

3 Build and Usage

3.1 Building the Library

A simple `Makefile` is provided. Type:

```
1 make
```

This will build all objects in `build/obj/` (or a similar location), archive them into `libcryptomodule.a`, and place test executables in `build/bin/`.

3.2 Running Tests

After running `make`, you can run:

```
make test
make run-tests
```

to compile test files and optionally execute them. The `run-tests` target (if implemented) loops over each test binary, outputting pass/fail status.

3.3 Linking and Using the Library

In your own C program:

```c
#include <stdio.h>
#include <cryptomodule/block/aes.h>
#include <cryptomodule/mode/gcm.h>

int main(void) {
    // Example usage
    // e.g. set up AES key, GCM mode, etc.
    return 0;
}
```

Then compile and link:

```
gcc -I./include -L. -lcryptomodule my_app.c -o my_app
```

Adjust as needed for your directory paths. The library name may be placed under `-L./build/lib` if you store artifacts there.

4 Development Guidelines

4.1 Style and Naming Conventions

- **Functions and Variables**: Use lowercase with underscores for internal helper functions. Exported (public) functions should have a prefix, e.g. `aes_encrypt()`, `gcm_init()`, etc.
- **Headers**: Each cryptographic feature has a matching `.h` / `.c` pair named consistently in the appropriate subdirectory.
- **Indentation**: Typically 4 spaces, no hard tabs.

4.2 Contributing a New Algorithm

If you want to add a new block cipher, for example:

1. Create `mycipher.h` in `include/cryptomodule/block/`.

2. Place the implementation in `mycipher.c` under `src/block/`.

3. Add references to it in the main `Makefile` (or rely on wildcard if used).

4. Write minimal test code in `tests/test_mycipher.c` for coverage.

4.3 Testing and Validation

Test each block cipher, mode, or function individually. Some recommended steps:

- **Unit tests**: Confirm core functionality. E.g., check known test vectors for AES, HMAC, etc.
- **Integration tests**: For example, test AES + GCM end-to-end encryption/decryption with known vectors.
- **Continuous Integration (CI)**: If you host on a platform that supports CI, set up automatic builds and tests for each commit / pull request.

5 Example Makefile Snippet

A simplified snippet is shown below, grouping the relevant source files:

Listing 1: Sample Makefile Snippet

```
CC          = gcc
AR          = ar
RANLIB      = ranlib
CFLAGS      = -O2 -Wall -I./include
LIB_NAME    = libcryptomodule.a

BUILD_DIR   = build
OBJ_DIR     = $(BUILD_DIR)/obj
BIN_DIR     = $(BUILD_DIR)/bin

BLOCK_SRCS   = $(wildcard src/block/*.c)
MODE_SRCS    = $(wildcard src/mode/*.c)
RNG_SRCS     = $(wildcard src/rng/*.c)
HASH_SRCS    = $(wildcard src/hash/*.c)
MAC_SRCS     = $(wildcard src/mac/*.c)
KDF_SRCS     = $(wildcard src/kdf/*.c)
KEYSETUP_SRCS= $(wildcard src/keysetup/*.c)
SIGN_SRCS    = $(wildcard src/sign/*.c)

SRCS_ALL     = $(BLOCK_SRCS) $(MODE_SRCS) $(RNG_SRCS) \
                $(HASH_SRCS)  $(MAC_SRCS)  $(KDF_SRCS) \
                $(KEYSETUP_SRCS) $(SIGN_SRCS)
OBJS_ALL     = $(patsubst src/%.c,$(OBJ_DIR)/%.o,$(SRCS_ALL))

TEST_SRCS    = $(wildcard tests/*.c)
TEST_OBJS    = $(patsubst tests/%.c,$(OBJ_DIR)/%.o,$(TEST_SRCS))
TEST_BINS    = $(patsubst tests/%.c,$(BIN_DIR)/%,$(TEST_SRCS))
```

```
28
29 .PHONY: all clean test run-tests
30
31 all: $(LIB_NAME) test
32
33 $(LIB_NAME): $(OBJS_ALL)
34     $(AR) rcs $@ $^
35     $(RANLIB) $@
36
37 $(OBJ_DIR)/%.o: src/%.c
38     @mkdir -p $(dir $@)
39     $(CC) $(CFLAGS) -c $< -o $@
40
41 test: $(TEST_BINS)
42
43 $(OBJ_DIR)/%.o: tests/%.c
44     @mkdir -p $(dir $@)
45     $(CC) $(CFLAGS) -c $< -o $@
46
47 $(BIN_DIR)/%: $(OBJ_DIR)/%.o $(LIB_NAME)
48     @mkdir -p $(BIN_DIR)
49     $(CC) $(CFLAGS) $< -o $@ -L. -lcryptomodule
50
51 run-tests: test
52     @for t in $(TEST_BINS); do echo "Running $$t..."; $$t || exit 1; done
53
54 clean:
55     rm -rf $(BUILD_DIR) $(LIB_NAME)
```

6 Security Considerations

- This library is intended as a reference or a building block. For production use, ensure the code is reviewed, tested, and validated for your environment.
- Keep in mind side-channel leaks, secure memory wiping, and other cryptographic best practices.

7 Acknowledgments & Contact

We appreciate any contributions, bug reports, or improvements. If you find issues, please open an issue or contact the maintainers.

End of Document