Implementing Crypto in C with OpenSSL

CATALDO BASILE

< CATALDO.BASILE@ POLITO.IT >

POLITECNICO DI TORINO

What is OpenSSL?

opensource initiative composed of two libraries

- a general-purpose cryptography library (libcrypto)
- a full implementation of the SSL/TLS protocols (libssl)

originally developed and known as SSLeay

- in 1995 by Eric A. Young and Tim J. Hudson
- renamed in 1998 as OpenSSL (0.9.1c)
- (latest) stable versions:
  - 1.1.1t (07 Feb 2023, reached its End of Life)
  - *3.4.1 (Feb* 2025*)*

- https://www.openssl.org/source/
- https://github.com/openssl/openssl/blob/master/CHANGES.md
- https://www.openssl.org/docs/man3.0/man7/migration_guide.html

OpenSSL for the Cryptography course…

open-source implementation of cryptographic functions

- play with cryptographic functions
  - a library of crypto APIs
    - to develop cryptographic applications

OpenSSL follows the object-oriented principle

- available for both C and C++
- when used in C…
  - each cryptographic algorithm is built upon a Context
    - a Context is an object that holds the necessary parameters and keep the states

Installing OpenSSL

```
$ apt install openssl
```

- ... or download the latest stable version from http://www.openssl.org
- compile and install (the latest version of) OpenSSL
  - https://github.com/openssl/openssl/blob/master/INSTALL.md#installation-steps-in-detail

```
$ gunzip  openssl-3.4.1.tar.gz
$ tar xvf openssl-3.4.1.tar
$ cd openssl-3.2.1
$ ./config
$ make
$ make test   # this command is optional
$ make install
```

  - typically
    - binaries installed in /usr/local/bin/openssl
    - libraries are installed in */usr/local/lib*
    - header files are in */usr/local/include/openssl*

Overview of crypto library (I)

to see the symmetric crypto algorithms supported, type:

```
$ openssl help
$ openssl list --cipher-algorithms
$ openssl list --cipher-commands
```

- symmetric block algorithms: AES, DES, 3DES, Camellia, Aria, SM4, (CAST, RC2, RC5 IDEA, Blowfish, SEED), …
  - in CBC, CFB, ECB, and OFB modes; (+) CTR and XTS for AES; for each cipher the default mode is CBC
- symmetric stream algorithms: RC4, ChaCha20, …

Overview of crypto library (II)

to see the supported digest algorithms, type:

```
$ openssl list  --digest-algorithms
$ openssl list  --digest-commands
```

- digest algorithms: BLAKE2b512, BLAKE2s256, MD4, MD5, RIPEMD160, SHA1, SHA224, SHA384, SHA512, SHA3, whirlpool, …
- asymmetric algorithms: RSA, DSA, DH, ECC
- authentication: HMAC
- authenticated encryption: AES-128-CBC-HMAC-SHA1, AES-128-CBC-HMAC-SHA256, AES-256-CBC-HMAC-SHA1, AES-256-CBC-HMAC-SHA256, ChaCha20-Poly1305, id-aes128-CCM, id-aes128GCM

Ready for programming? Not yet…

the standard installation (e.g., in Kali) does not include the necessary files for compiling programs

need to install *libssl-dev*

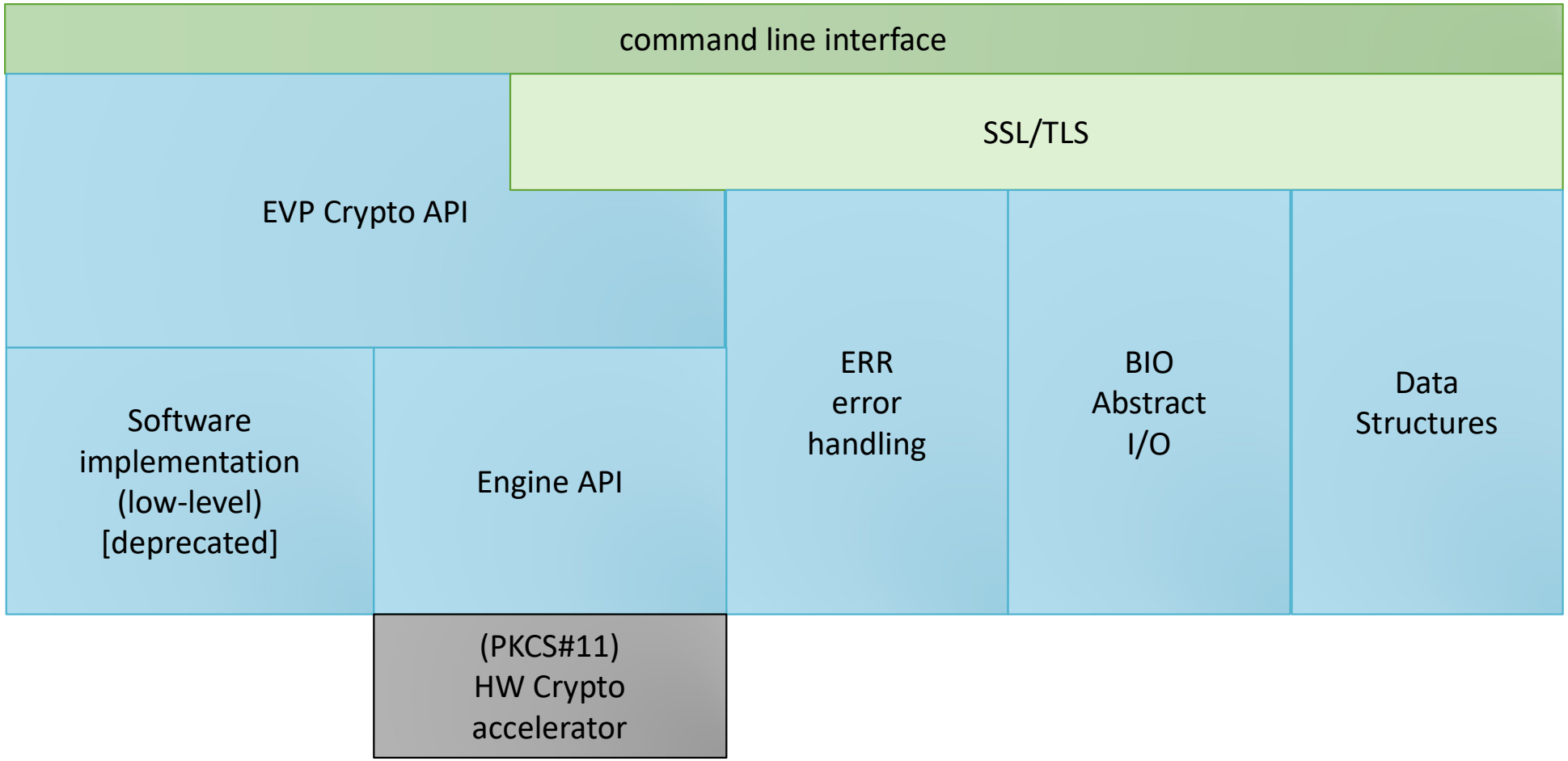
- in Debian-like distributions

```
$ sudo apt install libssl-dev
```

- compiling and linking the crypto libraries with *gcc*

```
gcc yourprogram.c -lcrypto
```

OpenSSL architecture



Overview of crypto library (IV)

the library itself is divided into logical modules, among which:

- *openssl/evp.h*: Envelope API (high-level API)
  - wraps the low-level crypto.h operations in a higher-level interface
  - after years of use, finally, it is considered mature enough
- *openssl/ssl.h*: SSL, TLS, DTLS protocols
  - exposes protocols' operations
- *openssl/rand.h*: pseudo-random number generator
- openssl/err.h: errors, standard messages
  - helps you easily recognize errors and print proper messages
- *openssl/crypto.h*: basic cryptographic algorithms (low-level API)
  - symmetric, asymmetric, hash, elliptic crypto curves
  - deprecated in v3+

Command-line interface of OpenSSL

- provides a command-line tool
  - exposes the features of the library, e.g., to calculate a hash, to encrypt/decrypt data
  - “batch” mode
- may also be used in “interactive” mode
  - by running the OpenSSL binary (with no options)
  - enters in “interactive” mode
  - a prompt indicates that the tool is ready to execute OpenSSL commands

```
$ openssl
OpenSSL> type openssl commands here
```

Command-line interface of OpenSSL

Batch mode

- by running openssl (+ openssl commands with parameters)

```
$ openssl command  [command_opts]  [command_args]
```

Help

```
$ man openssl
```

A typical OpenSSL program structure

Typical use of OpenSSL

include OpenSSL libraries

```
#include <openssl/evp.h>
#include <openssl/rand.h>
#include <openssl/err.h>
```

load OpenSSL facilities

when you need to use a crypto function

- create the context
- initialize the context
- operate on the context
- finalize on the context
- destroy/free the context

free OpenSSL facilities data

Typical use of OpenSSL

include OpenSSL libraries

load OpenSSL facilities

```
ERR_load_crypto_strings();
OpenSSL_add_all_algorithms();
```

when you need to use a crypto function

- create the context
- initialize the context
- operate on the context
- finalize on the context
- destroy/free the context

free OpenSSL facilities data

Typical use of OpenSSL

include OpenSSL libraries

load OpenSSL facilities

when you need to use a crypto function

- create the context
- initialize the context
- operate on the context
- finalize on the context
- destroy/free the context

select the crypto tool
- sym/asym/hash/...
prepare the memory and the data structures
- equivalent to a *new*

EXAMPLE: create an object for performing symmetric encryption

free OpenSSL facilities data

Typical use of OpenSSL

include OpenSSL libraries

load OpenSSL facilities

when you need to use a crypto function

- create the context
- initialize the context
- operate on the context
- finalize on the context
- destroy/free the context

add more details
- select specific modes
- assign keys, IV, nonces
- ... and more configuration options

select the algorithm and mode
- AES128 in CBC mode
- pass the key and the IV

free OpenSSL facilities data

Typical use of OpenSSL

include OpenSSL libraries

load OpenSSL facilities

when you need to use a crypto function

- create the context
- initialize the context
- operate on the context
- finalize on the context
- destroy/free the context

passes the data on which the algorithm operates
- call it all the times you need it!
- you may receive some intermediate output that need to be stored

free OpenSSL facilities data

feed the context with the data you want to encrypt

Typical use of OpenSSL

include OpenSSL libraries

load OpenSSL facilities

when you need to use a crypto function

- create the context
- initialize the context
- operate on the context
- finalize on the context
- destroy/free the context

- perform the concluding operations and extract the last output

pass the last data, add the padding, and cipher the last block

free OpenSSL facilities data

Typical use of OpenSSL applications

include OpenSSL libraries

load OpenSSL facilities

when you need to use a crypto function

- create the context
- initialize the context
- operate on the context
- finalize on the context
- destroy/free the context

free OpenSSL facilities data

just free the memory

after being used, objects are no longer needed
- free memory
- they are stateful, you cannot reuse them

ERR_load_crypto_strings()

Typical use of OpenSSL

include OpenSSL libraries

load OpenSSL facilities

when you need to use a crypto function

- create the context
- initialize the context
- operate on the context
- finalize on the context
- destroy/free the context

free OpenSSL facilities data → ERR_free_strings()