Exercises on Cryptography: intro

CATALDO BASILE

< CATALDO.BASILE@ POLITO.IT >

POLITECNICO DI TORINO

Objectives of the Cryptography exercises

- improving programming skills, mapping theory into programming languages
  - in a very specific field: cryptography

1. implementing crypto with two languages
   - C language: standard for high efficiency custom solutions
   - python: the most used language in the offensive security field
2. understanding attacks against crypto
   - learn how to mount these attacks helps understanding crypto (and the attack itself)
   - being ready if you have to mount them
   - …not exactly penetration testing but…

the first commandment for computer scientists:
don’t invent cryptosystems

The role of Python

implementing complex attacks in C is simply…

- …crazy…
- …or just very time consuming

python is the de facto standard for implementing attacks

- for attackers the only rule is "the faster the better"
- performance is reasonable in most cases
  - some python libraries run faster than (*our-)not-so-optimized* C code
  - plenty of libraries for attacking purposes
    - don't reinvent the wheel
  - a different approach to programming …more google dependent
- python was not presented in any course in your career
  - but we are computer engineers!

Planning

	C (Openssl, basic)	Python (basic)	Python (advanced)
1	intro, basics, symmetric crypto		
2	symmetric crypto, hash, MAC		
3		intro, basics, attack tools, crypto	
4		crypto, hash, MAC	
5			attacks against ECB mode
6			attacks against stream algorithms and modes
7			padding oracle
8			other attacks against CBC mode
9	asymmetric crypto		
10		asymmetric crypto	
11			RSA attacks
12			RSA attacks
13			attacks against Hash functions
14		(SRP + SAE + ECC)	

Key takeaways

- competences in computer system security
  - …in (hope) a less boring way
- alternative approach to problem solving
  - “normal” engineers
    - from requirements + design + implementation = constructive approach →
  - “attackers”
    - from implementation +(maybe requirements) →
      - misuse a system for your purposes
  - helpful to complete cybersecurity skills

- first step towards approaching the world of the CTFs
  - solve introductory challenges in the crypto area
    - funny stuff for nerds (?)

Environment for the exercises

- reference architecture: **Kali Linux 2021.1**
- VM available for most hypervisors
    - https://www.offensive-security.com/kali-linux-vm-vmware-virtualbox-image-download/
  - or install on multi-boot (do you really want to do this in 2021?)
    - https://www.kali.org/downloads/
  - or live (discouraged unless you really want to use persistence)
    - https://www.kali.org/downloads/
- the Pyhton 3 interpreter
  - additional packages will be proposed and added using *pip install*
- openssl and openssl for developers
  - install from sources or from Linux repositories
- WARNING: you may also want to use Windows, MACs, etc. but exercises will not be tested on these platforms
  - everything “should” work but if it does not you have to solve issues yourself…

Support tools

- github or dropbox
  - for the source code

- slack
  - Q/A and share questions
  - you can answer your colleagues' questions
    - will be validated

- will be created in the next days

Implementing Crypto in C with OpenSSL

CATALDO BASILE

< CATALDO.BASILE@ POLITO.IT >

POLITECNICO DI TORINO

What is OpenSSL?

opensource initiative composed of two libraries

- general-purpose cryptography library (libcrypto)
- sull 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 version: 1.1.1j (16 February 2021)
  - https://www.openssl.org/source/

OpenSSL for this course…

open-source implementation of cryptographic functions

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

follows the object-oriented principle

- each cryptographic algorithm built upon a context
  - the context is an object that holds the necessary parameters

Installing OpenSSL

```
$ apt install openssl
```

- ... or download the latest stable version from http://www.openssl.org
- compile and install (the latest version of) OpenSSL

```
$ gunzip openssl-1.1.1j.tar.gz
$ tar xvf openssl-1.1.1j.tar
$ cd openssl-1.1.1j
$ ./config
$ make
$ make test   # this command is optional
$ make install
```

  - 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, DESX 3DES, CAST, RC2, RC5 IDEA, Blowfish, SEED, Camellia, …
  - 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 (in Kali) does not include the necessary files for compiling programs

install libssl-dev

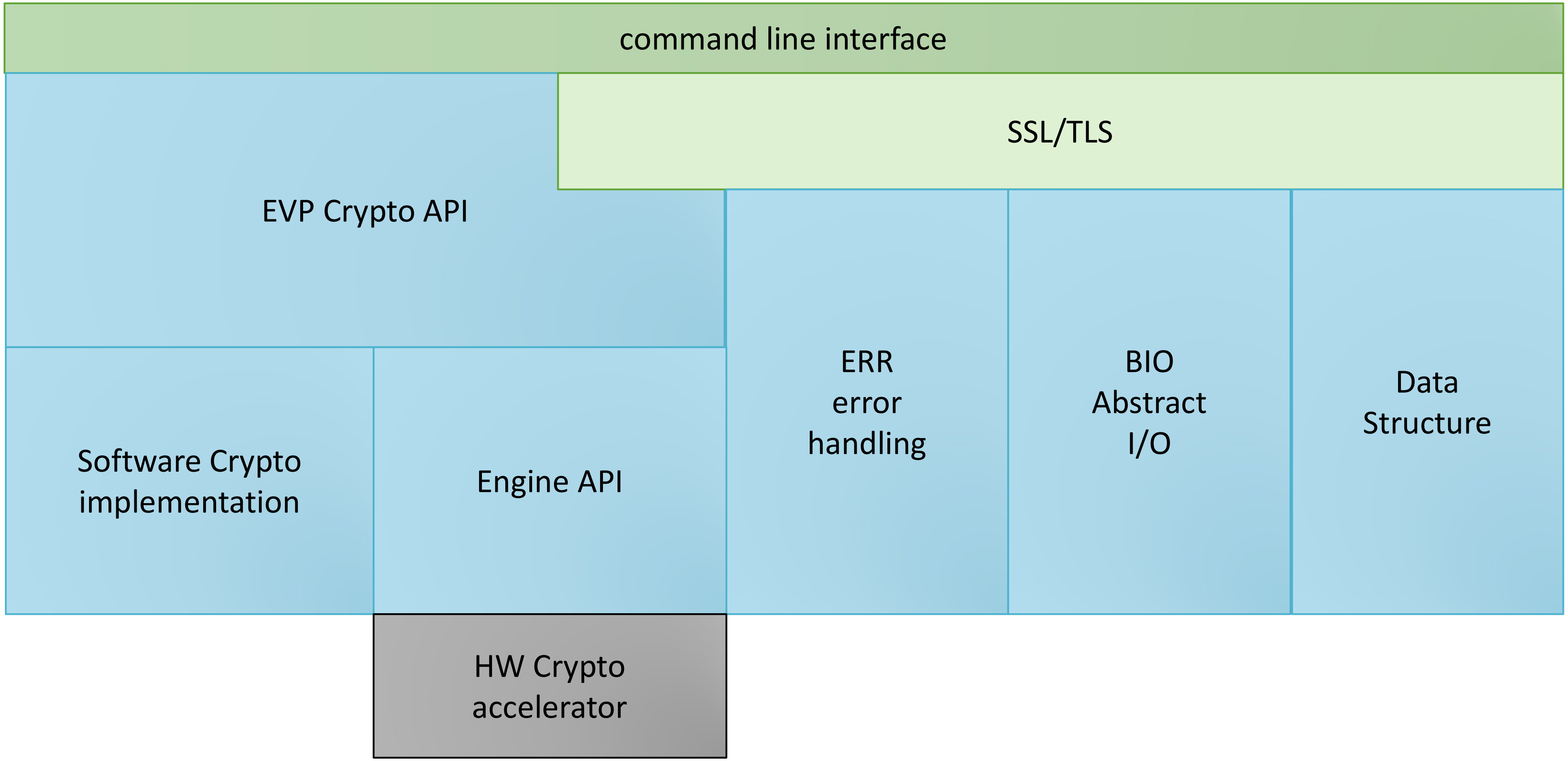
- in Debian

```
$ 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 in logical modules, among which:

- *openssl/crypto.h*: basic cryptographic algorithms
  - symmetric, asymmetric, hash, elliptic crypto curves
- *openssl/evp.h*: Envelope API
  - wraps the low-level crypto.h operations in a higher-level interface
- *openssl/ssl.h*: SSL,TLS, DTLS protocols
  - wraps TLS protocols operations
- *openssl/rand.h*: pseudo-random number generator

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> standard openssl commands
```

Command-line interface of OpenSSL

Batch mode

- by running openssl (+ openssl commands with parameters)

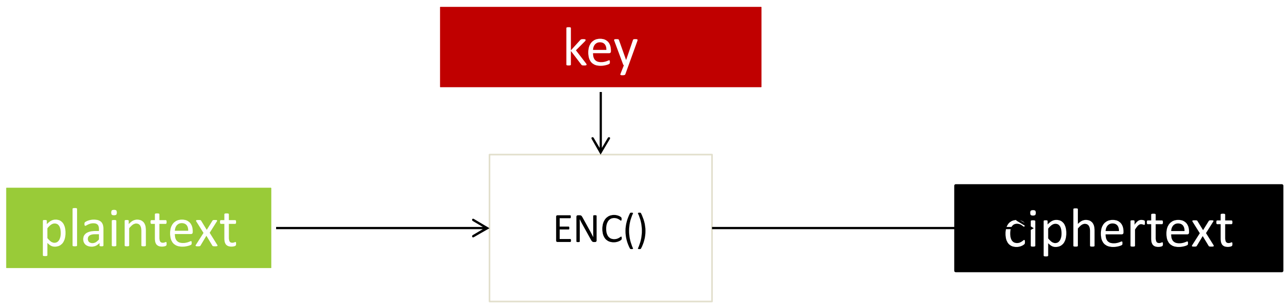
```
$ openssl command  [command_opts]  [command_args]
```

Help

```
$ man openssl
```

Symmetric encryption with OpenSSL

Symmetric encryption



used to enforce confidentiality

- only who knows a secret (e.g., a key) can perform some operations

main algorithm design: block vs. stream

- AES vs. ChaCha20

additional decisions for block algorithms

- padding yes/no (the standard is PKCS#5)
- modes of operations: ECB, CBC, OFB, CFB, …

Symmetric encryption in OpenSSL

encrypting all the data at once is neither efficient nor practical

- e.g., encrypt a 4GB file → allocate 8GB memory
  - 4GB in input and 4GB to store the output
- needs the entire plaintext before starting
  - e.g. encrypted sockets

symmetric encryption is «naturally structured» as an operation on a limited number of bytes

- e.g., block algorithms
- e.g., stream algorithm can work with a limited number of bytes at the time

OpenSSL implements symmetric encryption with incremental functions

- the encryption context manages the incremental updates

Incremental symmetric encryption

Encryption (pseudo-code):

```
ctx = context_initialize(encrypt, cipher, mode, key, iv, …);
cycle:
    ciphertext_fragment = encrypt_update(ctx, plaintext_fragment);
end:
ciphertext_fragment = encrypt_finalize(ctx);
```

Decryption:

```
ctx = context_initialize(decrypt, cipher, mode, key, iv, …);
cycle:
    plaintext_fragment = decrypt_update(ctx, ciphertext_fragment);
end:
plaintext_fragment = decrypt_finalize(ctx);
```

An example with block ciphers

- block ciphers process blocks of fixed input length
  - depending on the algorithm 64bit (old e.g., DES) and 128 bit (new, e.g., AES)
    - length of the output fragments (both encryption and decryption) multiple of the block size
- initialization: prepare algorithms for execution, initialize constant values, etc.
- update: does nothing until an entire block is full
    - e.g., “encryption update” on 1 byte with AES → output fragment = 0 bytes
      - not enough bytes to fill an AES block
    - unprocessed bytes temporarily stored in the context
- finalization: encrypts the remaining bits and returns the final block
  - padding bytes are added if the last block in not full
    - *ciphertext length > plaintext length*
  - during decryption the validity of padding is checked
    - invalid may indicate a corrupted ciphertext

Incremental encryption in Openssl

done with the EVP API

- can be used by including *openssl/evp.h*
- a unique interface for all symmetric encryption algorithms
  - https://github.com/openssl/openssl/blob/master/include/openssl/evp.h
  - https://www.openssl.org/docs/man1.0.2/man3/EVP_EncryptInit.html

EVP API provides

- a data structure (the context): *EVP_CIPHER_CTX*
- functions for:
  - context creation/destruction: *EVP_CIPHER_CTX_new* / EVP_CIPHER_CTX_cleanup / …
  - context initialization: e.g., *EVP_EncryptInit*/*EVP_DecryptInit* or *EVP_CipherInit*
  - encryption/decryption: *EVP_EncryptUpdate* or *EVP_CipherUpdate*)
  - finalization: *EVP_EncryptFinal* or *EVP_CipherFinal*

Ciphers in Openssl

symmetric algorithms are “objects” to be loaded in the context

- to use an algorithm, obtain a pointer to the proper object
  - and pass it to the context
- the EVP provides means to get these pointers
  - e.g., to have a reference to a Blowfish-CBC object you can use:

```
EVP_CIPHER *c = EVP_bf_cbc();
```

  - HINT: names look like the ones you use in OpenSSL in command line
    - e.g. -aes-128-cbc → EVP_aes_128_cbc
    - start with the ”EVP_” prefix then substitute s/”-”/”_”

EVP_EncryptInit() and EVP_DecryptInit()

```
int EVP_EncryptInit(EVP_CIPHER_CTX *ctx, const EVP_CIPHER *type, unsigned char *key,
                    unsigned char *iv);
int EVP_DecryptInit(EVP_CIPHER_CTX *ctx, const EVP_CIPHER *type, unsigned char *key,
                    unsigned char *iv);
int EVP_CipherInit_ex(EVP_CIPHER_CTX *ctx, const EVP_CIPHER *type,
                      ENGINE *impl, const unsigned char *key, const unsigned char *iv, int enc);
```

- initialize the context data structure
  - prepares the input and output buffers, initializes the constants and algorithm specific structures
  - ctx = the pointer to the EVP cipher context object to be used.
  - type = the symmetric algorithm to use
    - the pointer to be obtained by calling one of the EVP functions
  - iv = the initialization vector to use
  - enc = 1 for encryption, 0 for decryption and -1 to leave the value unchanged (the value used in the last call)
  - ENGINE = an impl.ctx context or NULL for default

EVP_EncryptUpdate() and EVP_CipherUpdate()

```
int EVP_EncryptUpdate(EVP_CIPHER_CTX *ctx, unsigned char *out, int *outl, unsigned char *in, int inl);
```

- encrypts *inl* bytes of the buffer in and
- writes *outl* encrypted data in out
  - to be called repeatedly to encrypt all the input data

- *ctx* = the EVP cipher context object to be used
- *out* = buffer used to save the encrypted data
- *outl* = pointer to the size (in bytes) actually written in the buffer containing the encrypted data
- *in* = buffer that contains the data to encrypt
- *inl* = size (in bytes) of the buffer that contains the data to be encrypted

EVP_EncryptFinal() function

```
int EVP_CipherFinal(EVP_CIPHER_CTX *ctx, unsigned char *outm,  int *outl);
```

encrypts the leftover data and performs the last operations

- i.e., adding and checking padding
- *ctx* = the EVP cipher context object to be used
- *outm* = buffer used to save the (leftover) encrypted data
- *outl* = pointer to the size (in bytes) actually written in the buffer containing the encrypted data