CATALDO BASILE

< CATALDO.BASILE @ POLITO.IT >

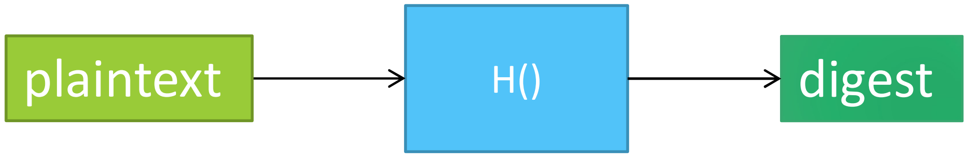
POLITECNICO DI TORINO

Computing digests and HMACs with OpenSSL

Agenda

- digests in OpenSSL
- computing MACs
- useful functions and error handling
- programs in C

Digest in OpenSSL with hash functions



takes as input the plaintext and outputs a digest

- finding another plaintext with the same digest is computationally unfeasible
- finding two plaintexts with the same digest is also computationally unfeasible

implementing hashing “as one step” is neither efficient nor practical

- same reasons as for symmetric encryption… memory and availability of data

hash calculation is implemented in OpenSSL with incremental functions

- initialize, then
- update a hash context step-by-step, then
- finalize it

Incremental hashing

- Hashing (pseudo-code):

```
md_ctx = context_initialize(hash_algorithm);
cycle:
    context_update(md_ctx, plaintext_fragment);
end:
digest = context_finalize(md_ctx);
```

- Hash verification:

```
computed_digest = <the same as above>;
compare(computed_digest, received_digest);
```

EVP API for hash functions

- EVP API provides a single interface towards all hash algorithms supported by OpenSSL
- included in *openssl/evp.h*
  - https://www.openssl.org/docs/manmaster/man3/EVP_Digest.html
- functions:
  - context creation: *EVP_MD_CTX_new*
  - hashing:
    - initialization: *EVP_DigestInit* to specify the hash algorithm to be used (e.g., SHA1)
    - update: *EVP_DigestUpdate*
    - finalize: *EVP_DigestFinal*

- the explicit API available for each hash algorithm is also available
  - openssl/sha.h
  - but it is deprecated from v3.0

Some useful functions

```
const EVP_MD *EVP_sha1(void);
```

- in general, *EVP_digestname()* functions are pointers to the EVP_MD structure that contains the implementation of the actual digest algorithms

```
EVP_MD_size(EVP_md5());
```

- returns the size (in bytes) of a digest (e.g. 16 bytes for MD5)

Useful functions for hash verification

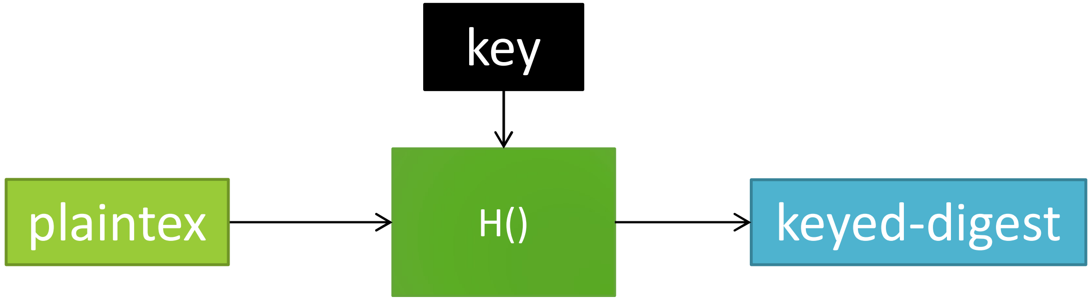
```
int CRYPTO_memcmp(computed_digest, received_digest, digest_len);
```

- compares two portions of memory in fixed time
- returns 0 if they are equal
- defined in <openssl/crypto.h>

- NOTE: don't use memcmp()
  - NOT safe to use because it makes the system vulnerable to timing attacks
  - send several (wrong) digests and measure the runtime → learn the value of the correct digest

Keyed-digests

Computing keyed-digests in OpenSSL



keyed hash algorithms provide both integrity and (data) authentication

HMAC is supported in OpenSSL
- it is part of a generic EVP interface for 'Signing and Verifying'
  - https://wiki.openssl.org/index.php/EVP_Signing_and_Verifying

two implementations
- a dedicated HMAC function
- a set of functions that follow the incremental approach
  - initialize the context, update the hash context step-by-step, finalize the context

Incremental keyed-hashing

- Keyed-hashing (pseudo-code):

```
hmac_ctx = context_initialize(hash_algorithm,key);
cycle:
    context_update(hmac_ctx, plaintext_fragment);
end:
keyed-digest = context_finalize(hmac_ctx);
```

- Keyed-hash verification:

```
computed_keyed_digest = <the same as above>;
compare(computed_keyed_digest, received_keyed_digest);
```

Error handling

Handling OpenSSL library errors

As most OpenSSL functions, hash and MAC functions return 1 on success or 0 on error

- best practice: check all the return codes and handle them as appropriate
  - some functions do not follow this principle
    - check a signature with some functions you get 1 if the signature is correct, 0 if it is not correct and -1 if something bad happened like a memory allocation failure

OpenSSL provides a set of functions to manage the errors

- load all the strings with the errors
  - don't waste time with the *printf*
- manage a stack with all the errors
  - you don't lose all the errors when you have to debug
    - or manage the exceptions…

Useful functions

error strings are already available in the library

- ERR_load_crypto_strings();
- ERR_free_strings();

common functions to get errors from the maintained stack

- void ERR_print_errors_fp(FILE *fp);
  - void ERR_print_errors(BIO *bp);

implement a default error function and use it in the whole program

```
void handleErrors(void)
{
   ERR_print_errors_fp(stderr);
   abort();
}
```