Crypto Programming

mbedTLS

Olivier Tuchon

ESIEA MS-SIS

07 fevrier 2018

Prerequis

Ce que vous devriez deja connaitre

- C ;
- crypto symetrique ;
- crypto asymetrique.

Prerequis

Ce que vous devriez deja connaitre

- C ;
- crypto symetrique ;
- crypto asymetrique.

Let's Go !

mbedTLS

La lib crypto qui ne fait pas le café

Features

sym	AES, 3DES, DES, ARC4, Camelia, XTEA
modes	ECB, CBC, CTR, CFB
hash	MD2, MD4, MD5, SHA1, SHA2, SHA4
prng	Havege
others	big numbers, base64
asym	RSA (PKCS#1 v1.5 & v2.1), DH
protocols	SSLv3, TLS v1.0 TLS v1.1 TLS v1.2
pki	x509
smart cards	PKCS#11 (OpenSC helper library)
self tests	

mbedTLS

La lib crypto qui ne fait pas le cafe

download & install (linux)

```
$ git clone https://github.com/ARMmbed/mbedtls.git
$ make
```

Windows

Il y aussi une solution *Visual Studio* pour ca !
Le repertoire `visualc` est votre ami.

mbedTLS

La lib crypto qui ne fait pas le cafe

Browsing repository

doxygen/	doxygen documentation
library/	sources files
includes/	headers files
programs/	usefull executables
...	

Exercise #1

Key generator

Generate a *length* bytes random key

- using the *havege* prng
- print the result in hexadecimal

```
$ ./gen_key 20    ⤳ key (20) = e4f035b085a685e30e957aeb8507d75a546e9223
```

```
/**
 * @param [out] key         key generated
 * @param [in]  key_length  key length in bytes
 * @return      0 if OK, 1 else
 */
int gen_key(unsigned char *key, int key_length);
```

Key derivation

Password ⇒ Key

- pseudorandom function (hash, cipher, hmac)
- (good) password/passphrase
- (good) salt
- loop enough

see *PBKDF2, PKCS#5, /etc/shadow*

Key derivation

Make our own function :-)

- $H_0 = SHA256(password||salt||0)$
- $H_i = SHA256(H_{i-1}||password||salt||i)$
- $0 < i < 2^5$
- $K = H_i$

Key derivation

Make our own function :-)

```
$ ./deriv_passwd mYsUp3rPssW0rd
⇝ salt = 62a68d960350a6e0
⇝ key = 89744574dbd0dc0f278b1b57fa8386a138aca4cdc0865be256271ded7dc8d6c0
```

```
/**
 * @param [out] key (32 bytes)
 * @param [in]  passwd       user password
 * @param [in]  salt         salt
 * @param [in]  salt_len     salt length in bytes
 * @param [in]  iterations   number of iterations
 * @return      0 if OK, 1 else
 */
int deriv_passwd(unsigned char *key,
                 char *password,
                 unsigned char *salt, int salt_len,
                 unsigned int iterations);
```

Symetric file protection

Confidentiality + Integrity

Protect confidentiality

- which algorithm ?
- which mode ?
- which padding ?

Protect integrity

- which algorithm ?
- plain or cipher ?

Symetric file protection

Confidentiality + Integrity

- $F = plainTextFile$
- $C = cipherFile$
- $P = password$
- $Salt$
- $K = KDF(P, Salt, 2^5)$ //master key
- $K_c = HASH(K||0x00)$ //cipher key
- $K_i = HASH(K||0x01)$ //integrity key
- $C = CIPHER_{K_c}(F)||HMAC_{K_i}(CIPHER_{K_c}(F))$

Symetric file protection

Confidentiality + Integrity

```
/**
 * @param [out] output       ciphered buffer
 * @param [out] output_len   ciphered buffer length in bytes
 * @param [in]  input        plain text buffer
 * @param [in]  input_len    plain text buffer length in bytes
 * @param [in]  master_key   master key (km)
 * @param [in]  key_len      master key length in bytes
 * @param [in]  salt         salt
 * @param [in]  salt_len     salt length in bytes
 * @return      0 if OK, 1 else
 */
int protect_buffer(unsigned char **output, int *output_len,
                   unsigned char *input, int input_len,
                   unsigned char *master_key, int key_len,
                   unsigned char *salt, int salt_len);


HASH    :: SHA-256
HMAC    :: HMAC-SHA-256
CIPHER  :: AES-256-CBC
PADDING :: 0x80
```

Symetric file protection

Confidentiality + Integrity

```
/**
 * @param [out] output       plain text buffer
 * @param [out] output_len   plain text buffer length in bytes
 * @param [in]  input        ciphered text buffer
 * @param [in]  input_len    ciphered text buffer length in bytes
 * @param [in]  master_key   master key (km)
 * @param [in]  key_len      master key length in bytes
 * @param [in]  salt_len     salt length in bytes
 * @return      0 if OK, 1 else
 */
int unprotect_buffer(unsigned char **output, int *output_len,
                     unsigned char *input, int input_len,
                     unsigned char *master_key, int master_key_len,
                     int salt_len);


HASH    :: SHA-256
HMAC    :: HMAC-SHA-256
CIPHER  :: AES-256-CBC
PADDING :: 0x80
```

Hybrid file protection

Confidentiality

"Use the speed of symetric algorithm and the reliatbility of asymetric algorithm"

- $K_c = symetricCipherKey$ // random
- $K_{pub} = asymetricPublicKey$
- $K_{pri} = asymetricPrivateKey$
- $Cipher = SYM_{K_c}(Plain)$
- $WK_c = ENCRYPTASYM_{K_{pub}}(K_c)$
- $Sign = SIGNASYM_{K_{priv}}(IV||WK_c||Cipher)$

Hybrid file protection

Confidentiality

```
/**
 * @param [out] output       ciphered buffer
 * @param [out] output_len   ciphered buffer length in bytes
 * @param [in]  input        plain text buffer
 * @param [in]  input_len    plain text buffer length in bytes
 * @param [in]  path_pubkey_enc
 * @param [in]  path_privkey_sign
 * @return      0 if OK, 1 else
 */
int protect_asym_buffer(
                unsigned char **output, int *output_len,
                unsigned char *input, int input_len,
                char *path_pubkey_enc,
                char *path_privkey_sign);


SYM         :: AES-256-CBC
PADDING     :: 0x80
HASH        :: SHA256
ENCRYPTASYM :: RSA-2048 (PKCS#1 v2.1: OAEP)
SIGNASYM    :: RSA-2048 (PKCS#1 v2.1: PSS)
```