# Intro to asymmetric crypto in OpenSSL (RSA)

CATALDO BASILE

< CATALDO.BASILE@ POLITO.IT >

POLITECNICO DI TORINO

# Agenda

OpenSSL EVP interface for asymmetric cryptography

- generating RSA keys
- operations with RSA

examples in C

# Asymmetric cryptography: Programming

## EVP interface for the asymmetric algorithms

- more convenient than directly calling the API functions for RSA, DSA, DH, EC
  - mandatory in OpenSSL 3.0

## main functions for RSA

- e.g., prototypes of RSA functions available in /usr/local/include/openssl/rsa.h
- *int RSA_generate_key_ex(RSA *rsa, int bits, BIGNUM *e, BN_GENCB *cb);*
- *int RSA_public_encrypt(int flen, const unsigned char *from, unsigned char *to, RSA *rsa, int padding);*
- *int RSA_private_decrypt(int flen, const unsigned char *from, unsigned char *to, RSA *rsa, int padding);*

# Asymmetric in OpenSSL 3.0

the RSA generation has been completely changed

- https://www.openssl.org/docs/manmaster/man7/EVP_PKEY-RSA.html
- the easy way
  - *EVP_PKEY *EVP_RSA_gen(unsigned int bits);*
- more control
  - *EVP_PKEY *pkey = NULL;*
  - *EVP_PKEY_CTX *pctx = EVP_PKEY_CTX_new_from_name(NULL, "RSA", NULL);*
  - *EVP_PKEY_keygen_init(pctx);*
  - *EVP_PKEY_generate(pctx, &pkey);*
  - *EVP_PKEY_CTX_free(pctx);*

# Asymmetric in OpenSSL 3.0

the RSA generation has been completely changed

- https://www.openssl.org/docs/manmaster/man7/EVP_PKEY-RSA.html
- even more control
  - ```
    unsigned int primes = 3;
    unsigned int bits = 4096;
    OSSL_PARAM params[3];
    EVP_PKEY *pkey = NULL;
    EVP_PKEY_CTX *pctx = EVP_PKEY_CTX_new_from_name(NULL, "RSA", NULL);

    EVP_PKEY_keygen_init(pctx);
    params[0] = OSSL_PARAM_construct_uint("bits", &bits);
    params[1] = OSSL_PARAM_construct_uint("primes", &primes);
    params[2] = OSSL_PARAM_construct_end();
    EVP_PKEY_CTX_set_params(pctx, params);

    EVP_PKEY_generate(pctx, &pkey);
    EVP_PKEY_print_private(bio_out, pkey, 0, NULL);
    EVP_PKEY_CTX_free(pctx);
    ```
    ```

# RSA_public_encrypt

*int RSA_public_encrypt(int flen, unsigned char *from, unsigned char *to, RSA *rsa, int padding);*

- flen = specifies the number of bytes in the buffer to be encrypted.
- from = a buffer containing the data to be encrypted.
- to = a buffer that will be used to hold the encrypted data. It should be large enough to hold the largest possible amount of encrypted data,
  - can be determined by calling RSA_size and passing the RSA object (used to encrypt) as its only argument
- rsa = the RSA object that contains the public key to use to perform encryption
- padding = specifies which of the built-in padding types supported by OpenSSL should be used

# Types of padding for RSA encryption

RSA_PKCS1_PADDING

- the length of the data to be encrypted must be smaller than *RSA_size(rsa)-11*.
- used for compatibility with older apps

RSA_PKCS1_OAEP_PADDING

- the length of data to be encrypted must be smaller than *RSA_size(rsa)-41*.
- recommended for new apps

RSA_SSLV23_PADDING

- SSL-specific modification to the RSA_PKCS1_PADDING
- rarely used

RSA_NO_PADDING

- data to be encrypted is exactly *RSA_size(rsa)* bytes

# OpenSSL API for managing PEM data

- available with *#include <openssl.pem.h>*
  - several functions available
    - https://www.openssl.org/docs/man1.1.1/man3/PEM_read_PrivateKey.html

- *EVP_PKEY_RSA* PEM_read_RSAPrivateKey(FILE* fp, NULL, NULL, NULL);*
  - allocates an RSA private key and loads it from a PEM file
    - fp -> file where to read (opened with fopen())
    - it returns the EVP_PKEY_RSA data structure (or NULL if error)

- *EVP_PKEY_RSA* PEM_read_RSAPublicKey(FILE* fp, NULL, NULL, NULL);*
  - allocates a public key and loads it from a PEM file
    - fp -> file where to read (opened with fopen())
    - it returns the EVP_PKEY_RSA data structure (or NULL if error)

- EVP_PKEY *PEM_read_PrivateKey(FILE *fp, EVP_PKEY **x, pem_password_cb *cb, void *u);
  - saves the RSA key in the general-purpose private key data structure
    - *x* is the type of data that will be output (can be omitted with NULL)
    - *pem_password_cb* is the callback to use the passphrase *u* to edecrypt the private key

# Signatures with EVP_DigestSign*

follows a standard workflow

- create a message digest context EVP_MD_CTX_new
- init the context with
  - *EVP_DigestSignInit()*
- pass the date to digest with
  - *EVP_DigestSignUpdate()*
- conclude the computation of the digest with
  - *EVP_DigestSignFinal(context, NULL, &digest_len)*
- computate the signature calling again
  - *EVP_DigestSignFinal(context, signature, &digest_len)*