Examples in Cryptography with OpenSSL

Ivan "Rambius" Ivanov
rambiusparkisanius@gmail.com

August 5, 2010

What is cryptography?

Cryptography is the practice and study of hiding information. It studies the schemes of converting some original intelligible data to some unreadable data.

Building Blocks of a Cryptographic Scheme

Plaintext

Encryption algorithm

Keys

Ciphertext

Decryption algorithm

The secrecy of the system should depend only on the secrecy of the keys and not on the secrecy of the algorithm.

Attacks on a Cryptographic Scheme

An attack on a system is an attempt to conclude the plaintext or the keys of the system.
The attacker is assumed to know:

- the scheme's encryption and decryption algorithms
- considerable amount of ciphertexts

Ciphertext only

Known plaintext

Chosen plaintext

Chosen ciphertext

Brute force

Security of cryptographic systems

Information-theoretically secure
: The ciphertext provides no information about the plaintext (except its length)

Computationally secure
: One of the following is fulfilled:

- The cost of breaking the system exceeds the gain
- The time required to break the system exceeds the lifetime of the information

One-time Pad

Example of informationally-theoretically secure scheme

Fix an alphabet A with length $|A|$. Encrypt, send and decrypt message $P = p_1 p_2 p_3 ... p_m$ of length m over A.

1. Generate key $K = k_1 k_2 k_3 ... k_m$ of completely random letters over A.
2. Exchange the key with the receiving party.
3. Encrypt P to ciphertext $C = c_1 c_2 c_3 ... c_m$ with $c_i = (p_i + k_i) mod |A|$.
4. Sending party destroys its copy of K.
5. Sending party sends C.
6. Receiving party receives C.
7. Decrypt C to P with $p_i = (c_i - k_i) mod |A|$.
8. Receiving party destroys its copy of K.

One-time Pad - Questions

- How do we generate K, or how do we generate completely random - not pseudo-random - keys?
- How do we exchange the keys?

  Securely sending a message P of length m means securely exchanging a a key K of length m. One-time pad users can generate a vast amount of data, exchange it over a slow, but secure channel. Then they specify in the message the position in the data where the key starts from.
- Why is it secure?

  Let P is of length m and $C = E(P, K)$. Then for every other P' of length m there exists a key K' such that $P' = D(C, K')$

OpenSSL

OpenSSL provides support for:

- Multithreading with mutexes
- Error handling and error queues in ERR package
- Abstract IO in BIO package
- Pseudorandom number generation in RAND package
- Arbitrary precision math with big numbers and big prime numbers in BN package
- Hardware acceleration in ENGINE package

Pseudorandom Number Generation

Many operations require random numbers. OpenSSL provides a PRNG:

- Implemented in the RAND package - `openssl/rand.h`
- Cryptographically strong - not truly random, but difficult to predict
- Has to be initialized with a **high-entropy seed**

Seeding the PRNG

- Seed with a data buffer:
  - ```
    void RAND_add(const void *buf,
                  int num, double entropy)
    ```
  - `void RAND_seed(const void *buf, int num)`
- Seed with a file, including the OS PRNG device:
  - ```
    int RAND_load_file(const char *filename,
                       long bytes)
    ```
- Seed with an alternate entropy source, for example EGD:
  - ```
    int RAND_egd_bytes(const char *filename,
                       int bytes)
    ```
- On Windows seed with Windows-specific sources as a last resort - mouse events and screen contents
    ```

Seeding the PRNG - Examples

```
if (!RAND_load_file("/dev/random", 1024)) {
  fprintf(stderr, "Cannot seed the prng");
}

if (!RAND_load_file("/dev/urandom", 1024)) {
  fprintf(stderr, "Cannot seed the prng");
}

char *pools = {"/var/run/egd-pool", ...,
               "/etc/entropy", null}
for (i = 0; pools[i]; i++)
  if (RAND_egd_bytes(pools[i], 1024))
     break;
```

Symmetric Encryption

One key is used for encryption and decryption.

Block Ciphers and Stream Ciphers

- Block ciphers

  Splits the plaintext data into fixed-size blocks and the encrypt each block. The last block is padded if needed.

- Stream ciphers

  Encrypts the plaintext data one digit at a time

Block ciphers are better studied.

Block Cipher Modes

- Electronic Code Book (ECB)
- Cipher Block Chaining (CBC)
- Cipher Feedback (CFB)
- Output Feedback (OFB)

CBC, CFB and OFB may need an initialization vector.

Key and IV Generation

- from pass-phrase
- randomly generated

```
void ce_gen_random_key(unsigned char *key,
                       int b) {
  RAND_bytes(key, b);
}

void ce_gen_random_iv(unsigned char *iv, int b) {
  RAND_pseudo_bytes(iv, b);
}
```

Initializing Symmetric Ciphers

1. Get an EVP cipher context:

   ```
   EVP_CIPHER_CTX ctx;
   ```

2. Generate the key and the iv:

   ```
   char key[EVP_MAX_KEY_LENGTH];
   char iv[EVP_MAX_IV_LENGTH];
   seed_prng("/dev/random");
   gen_random_key(key, EVP_MAX_KEY_LENGTH);
   gen_random_iv(iv, EVP_MAX_IV_LENGTH);
   ```

3. Initialize the context with the keys and the algorithm:

   ```
   EVP_EncryptInit(&ctx, EVP_des_cbc(), key, iv);
   ```

4. Ready to encrypt or decrypt

What is EVP

EVP package provides a unified interface to all symmetric encryption algorithms.

```
EVP_EncryptInit(&ctx, EVP_des_cbc(), key, iv);
EVP_EncryptInit(&ctx, EVP_bf_cbc(), key, iv);
EVP_EncryptInit(&ctx, EVP_aes_128_cbc(), key, iv);
```

Some options for some ciphers can be set in the EVP cipher context:

```
EVP_CIPHER_CTX_set_key_length(&ctx, num)
```

or using the more generic function

```
EVP_CIPHER_CTX_ctrl(&ctx, int op, int arg, void *ptr)
```

Encryption

- Update the cipher context with the available data to encrypt

```
EVP_EncryptUpdate(ctx, ctext, &ol, ptext, il);
...
EVP_EncryptUpdate(ctx, ctext, &ol, ptext, il);
```

- Finalize the cipher

```
EVP_EncryptFinal(ctx, ctext, &ol);
```

Padding is possible.

Decryption

Similar to encryption, but use `EVP_DecryptUpdate` and `EVP_DecryptFinal`

```
EVP_DecryptUpdate(ctx, ptext, &ol, ctext, il);
...
EVP_DecryptUpdate(ctx, ptext, &ol, ctext, il);
EVP_EncryptFinal(ctx, ctext, &ol);
```

Example of symmetric encryption:

```
$ ./encdes "I travel from west to east"
$ ./decdes ...
$ ./encaes "I travel from west to east"
$ ./decaes ...
```

Hashes

Basic properties:

- Take data of any length as input and produce fixed-size output
- Computationally difficult to reverse - cannot determine the input from the output
- Computationally difficult to find a second input with the same hash as the first one
- Follow other statistical requirements as one-bit change in the input causes on average changes half of the bits in the output

Usages of hashes:

- Password storage - passwords are not stored in plain; instead the hashes of the passwords combined with a salt are stored
- Digital signatures - the content to be signed is hashed and the hash is actually signed
- Integrity of encrypted data - the message along with its hash are encrypted

How hashes work

1. Get an EVP message digest context and initialize it with an alg:

```
EVP_MD_CTX ctx;
char *alg = "md5";
EVP_MD *md = EVP_get_digestbyname(alg);
EVP_DigestInit(&ctx, md);
```

2. Update the context with data:

```
EVP_DigestUpdate(&ctx, data, len);
```

3. Finalize the digest:

```
EVP_DigestFinal(&ctx, hash, olen);
```

Examples:

```
$ ./rundgst md5 "testme"
$ ./rundgst sha1 "testme"
```

Assymetric Encryption

One key is used for encryption - the public key and another key is used for decryption - the private key.

Key Pair Generation

A key pair is encapsulated in RSA structure. An instance is created using:

```
RSA *RSA_generate_key(int bits, unsigned long e,
   void (*callback)(int, int, void *),
    void *cb_arg)
```

Once the keys are generated they can be DER or PEM-encoded and stored.

Example of key pair generation;

```
void status_prime(int code, int arg, void *cb_arg) {
  if (code == 0) {
// Potential prime
  } else if (code == 1 && arg && !(arg % 10)) {
printf(".");
  } else { // code = 2
// Found prime
  }
}

RSA *rsa = RSA_generate_key(2048, RSA_F4,
                            status_prime, NULL);

RSA *rsa = RSA_generate_key(2048, RSA_3,
                            status_prime, NULL);
```

Low Level RSA API - Encrypt / Decrypt

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

Low Level RSA API - Sign / Verify

```
int RSA_sign(int mdtype,
             unsigned char *buf,
             unsigned int buf_len,
             unsigned char *sig,
             unsigned int *siglen,
             RSA *rsa)
int RSA_verify(int mdtype,
               unsigned char *buf,
               unsigned int buf_len,
               unsigned char *sig,
               unsigned int siglen,
               RSA *rsa)
```

Q & A?