# CRYPTOGRAPHY #03.2

# RSA OAEP

Jacek Tchórzewski, jacek.tchorzewski@pk.edu.pl

## 1. RSAES-OAEP

Optimal Asymmetric Encryption Padding is a padding scheme used in RSA and defined in RFC 3447. OAEP allows to:

- add the element of randomness into the classical encryption scheme
- reduce risk related to the RSA homomorphism
- prevent partial decryption of a cryptogram

**Padding –** extending a message into the given size. There are many padding schemes, however, not all are considered as secure. Padding is used in symmetric block ciphers and in asymmetric block ciphers.

Accordingly to RFC 3447 let's define parameters for OAEP encoding:

*hLen* – length of the chosen hashing function in bytes

*H* – chosen hashing function

*k* – demanded length of the message after padding (can not exceed the size of modulus *n*) in bytes (**input parameter, however can be fixed if modulus n length is known**).

*mLen* – length of a message in bytes

*mgf1* – mask function menationed in chapter 2.

*M* – bytes of a message **(input parameter).**

<u>NOTE: *k* parameter should be chosen carefully. It should be as big as possible. Optimal value is number of bytes of modulus n.</u>

## OAEP encoding scheme presented in RFC 3448:

```
1) if mLen > k – 2*hLen – 2, return error.
2) lHash = H() (lHash is a byte table returned by H. Hash is
   calculated form an empty string).
3) Generate an octet string PS consisting of k - mLen – 2*hLen – 2
   zero octets.  The length of PS may be zero.
4) DB = lHash || PS || 0x01 || M. Length of DB is equal to k – hLen –
   1.
5) Generate a random octet string seed of length hLen.
6) dbMask = mgf1(seed, k - hLen - 1).
7) maskedDB = DB \xor dbMask.
8) seedMask = mgf1(maskedDB, hLen).
9) maskedSeed = seed \xor seedMask.
10)EM = 0x00 || maskedSeed || maskedDB. The length of EM is k.
11)return EM.
```

## OAEP decoding parameters:

*hLen* – length of the chosen hashing function in bytes

*H* – chosen hashing function

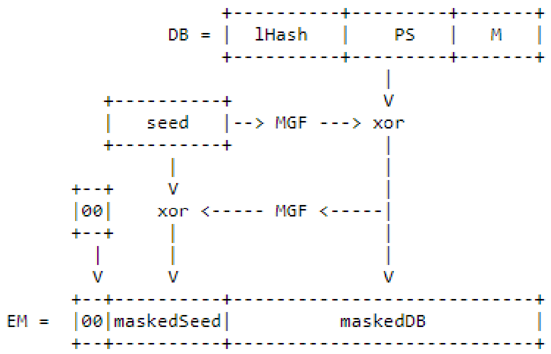
*k* – length of *EM*

*mgf1* – Mask function menationed in chapter 2.

*EM* – bytes of an encoded message (**input parameter**)

## OAEP decoding scheme presented in RFC 3448:

```
1) Separate the encoded message EM into a single octet Y, an octet
   string maskedSeed of length hLen, and an octet string maskedDB
   of length k - hLen – 1: EM = Y || maskedSeed || maskedDB.
2) lHash = H() (lHash is a byte table returned by H. Hash is
   calculated form an empty string).
3) seedMask = mgf1(maskedDB, hLen).
4) seed = maskedSeed \xor seedMask.
5) dbMask = mgf1(seed, k - hLen - 1).
6) DB = maskedDB \xor dbMask.
7) Separate DB into an octet string lHash' of length hLen, a
   (possibly empty) padding string PS consisting of octets with
  hexadecimal value 0x00, and a message M:
  DB = lHash' || PS || 0x01 || M.
8) If there is no octet with hexadecimal value 0x01 to separate PS
   from M, return error.
9) if lHash' != lHash, return error.
10)if Y != 0x00, return error.
11)return M.
```



```
                    +----------+---------+-------+
               DB = |  lHash   |    PS   |   M   |
                    +----------+---------+-------+
                                   |
         +----------+              V
         |   seed   |--> MGF ---> xor
         +----------+              |
               |                   |
      +--+     V                   |
      |00|    xor <----- MGF <-----|
      +--+     |                   |
        |      |                   |
        V      V                   V
      +--+----------+----------------------------+
EM =  |00|maskedSeed|          maskedDB          |
      +--+----------+----------------------------+
```

*FIg. 1 OAEP coding scheme presented in RFC 3447*

## 2. RSA – OAEP ciphering and deciphering scheme.

As we mentioned earlier, usage of simple RSA scheme is not 100% secure, that's why it involves OAEP. Before communication begins, RSA keys should be generated appropriately, and both sides should establish OAEP parameters.

RSA – OAEP ciphering scheme for a message *m,* and public key *(n, e)*:

```
1) BEM = OAEP_Encoding(m)
2) C = convertToNumber(BEM)
3) C = $C^e$ mod n
4) BC = convertToBytes(C)
5) return BC
```

RSA – OAEP deciphering scheme for a cryptogram *BC,* and private key *(d, e)*:

```
1) C = convertToNumber(BC)
2) C = $C^d$ mod n
3) BC = convertToBytes(C)
4) m = OAEP_Decoding(BC)
5) return m
```

**Exercise 1:**

Write a function that will be coding and decoding messages with OAEP usage. Assumptions for both functions:

1) *H* – is a SHA-256 hashing function.
2) Accordingly to the previous point, *hLen* = 32.
3) *k* = 256.

Functions structures:

```
1) byte[] OAEPEncoding(byte[] message)
2) byte[] OAEPDecoding(byte[] cipher)
```

Verification in *main* function:

```
String m = ”message”;

byte[] cipher = OAEPEncoding(bytearray(m));

byte[] decoded = OAEPDecoding(cipher);

print(String(decoded));
```

**Exercise 2:**

Improve RSA ciphering and deciphering functions done on previous classes. Both should combine RSA with OAEP as described in chapter 2. Verify your work in the same way as in Ex. 2. Create a class containing all methods and parameters described in this and previous classes.