CRYPTOGRAPHY #03.3

RSA

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

1. RSA – KEM

RSA Key Encapsulation Mechanism assumes usage of RSA for transferring keys used in symmetric encryption. A scheme assumes that RSA keys were generated and both sides established the same hashing function *H*.

RSA – KEM, Sender:

```
1) Find random value RAND, such that: 1 < RAND < n
2) BRAND = convertToBytes(RAND)
3) BKEY = H(BRAND)
4) BC = RSA_OAEP_ENCODING(BKEY)
5) return BC
```

RSA – KEM, Receiver get *BC* and than compute:

```
1) KEY = RSA_OAEP_DECODING(BC)
2) BKEY = H(KEY)
```

Now both sides have the same key *BKEY* which can be used for further symmetric communication. Note, that the size of *BKEY* is strictly related to the number of bytes returned by hashing function *H*. It is not a problem. The most popular symmetric ciphering scheme, AES, assumes usage of 16B, 24B or 32B key. Thus usage, for example, SHA-256 (which is returning 32B key) in the RSA-KEM scheme, and sometimes truncating the digest (when necessary) is solving this problem.

2. EMSA - PSS

Encoding Method for Signature with Appendix it is a scheme that is used to create RSA – PSS (RSA Provable Secure Signature). It is appropriately randomized to reach the highest possible security of a signature. It is also described in RFC 3448.

EMSA – PSS encoding parameters:

H – chosen hashing function

hLen – length (in bytes) of a hash returned by *H*

sLen – length os the salt. Should be set to *hLen.*

M – bytes of a message to be sgined **(input parameter)**

mgf1 – mask generator function.

emLen – assumed length of a signature. Can not exceed modulus *n.*

```
1) if emLen < hLen + sLen + 2, return error.
2) mHash = Hash(M). mHash length is equal to hLen.
3) Generate a random octet string salt of length sLen.
4) M' = (0x)00 00 00 00 00 00 00 00 || mHash || salt. Length of M' is
   equal to 8 + hLen + sLen.
5) H = Hash(M'). Length of H is hLen.
6) Generate an octet string PS consisting of emLen - sLen - hLen - 2
   zero octets.
7) DB = PS || 0x01 || salt. Length of DB is equal to emLen – hLen –
   1.
8) dbMask = mgf1(H, emLen - hLen - 1).
9) maskedDB = DB \xor dbMask.
10)EM = maskedDB || H || 0xbc.
11)return EM.
```

EMSA – PSS verification parameters:

H – chosen hashing function

hLen – length (in bytes) of a hash returned by *H*

sLen – length os the salt. Should be set to *hLen.*

EM – bytes of a signature **(input parameter)**

mgf1 – mask generator function.

emLen – length of *EM*

M – original message **(input parameter)**

```
1) if emLen < hLen + sLen + 2, return signature_invalid.
2) mHash = Hash(M). mHash length is equal to hLen.
3) if the last byte (most significant) of EM is not equal to 0xbc,
   return signature_invalid.
4) Let maskedDB be the leftmost emLen - hLen - 1 octets of EM, and
   let H be the next hLen octets (SEE POINT 10 IN CODING ALGORITHM).
5) dbMask = mgf1(H, emLen - hLen - 1).
6) DB = maskedDB \xor dbMask.
7) If the emLen - hLen - sLen - 2 leftmost octets of DB are not zero,
   return sginature_invalid
8) if the octet at position emLen - hLen - sLen – 1 (when indexing
   from 1) is not equal to 0x01, return signature_invalid
9) Let salt be the last sLen octets of DB.
10) M' = (0x)00 00 00 00 00 00 00 00 || mHash || salt.
11) H' = Hash(M'). Length of H' is equal to hLen.
12) if H' = H, return signature_valid.
```

```
                          +-----------+
                          |     M     |
                          +-----------+
                                |
                                V
                              Hash
                                |
                                V
                  +--------+----------+----------+
            M' =  |Padding1|  mHash   |   salt   |
                  +--------+----------+----------+
                                 |
      +--------+----------+      V
DB =  |Padding2|maskedseed|     Hash
      +--------+----------+      |
                 |               |
                 V               |    +--+
                xor <--- MGF <---|    |bc|
                 |               |    +--+
                 |               |      |
                 V               V      V
      +--------------------+----------+--+
EM =  |     maskedDB       |maskedseed|bc|
      +--------------------+----------+--+
```

FIg. 1 EMSA-PSS coding presented in RFC-3448

3. RSA - PSS

RSA Provable Secure Signature is combining the RSA algorithm and EMSA mentioned in chapter 6. to provide a secure digital signature. The scheme assumes that RSA keys were generated and EMSA parameters established.

RSA – PSS creation for a message *m*:

```
1) BEM = EMSA_Encode(m)
2) EM = convertToNumber(BEM)
3) EM = EMᵈ mod n
4) SIG = convertToBytes(EM)
```

RSA – PSS verification for a message *m* and signature *SIG*:

```
1) EM = convertToNumber(SIG)
2) EM = EMᵉ mod n
3) BEM = convertToBytes(EM)
4) return EMSA_Verify(BEM, m)
```

Exercise 1 (warm up):

Write two functions that will simulate the RSA – KEM scheme accordingly to chapter 1. The first one will be generating potential 256 bits (32B) symmetric key, ciphering it with RSA – OAEP usage and return as a byte array. The second one will be deciphering, retrieving, and returning this key. Use SHA-256. Function structure:

1) `byte[] generateRSAKEM()`
2) `byte[] receiveRSAKEM(byte[] cryptogram)`

Verify that both functions work properly, and key generated in *generateRSAKEM* is the same key as in *receiveRSAKEM.* You can use *DatatypeConverter.printHexBinary(byte[] bytes)* located *in javax.xml.bind.DatatypeConverter* to display hash values in both functions.

Verification in *main* function:

```
byte[] cipher = generateRSAKEM();

byte[] key = receiveRSAKEM (cipher);
```

Exercise 2:

Write two functions: one should generate EMSA-PSS signature and return it in byte form, the second one should verify it, and return true if signature is valid (false otherwise). Functions structure:

```
1) byte[] createEMSAPSS(byte[] message)
2) boolean verifyEMSAPSS(byte[] EM, byte[] message)
```

Parameters for *createEMSAPSS*:

1) *H* – is a SHA-256 hashing function. You can find it in *java.security.MessageDigest*.
2) Accordingly to the previous point, *hLen* = 32.
3) *sLen* = 32.
4) *emLen* = 255.

Parameters for *verifyEMSAPSS:*

1) H – is a SHA-256 hashing function. You can find it in java.security.MessageDigest.
2) Accordingly to the previous point, *hLen* = 32.
3) *sLen* = 32.

Verification in *main* function:

```
String m = ”message”;

byte[] EM = createEMSAPSS(m.getBytes());

System.out.println(verifyEMSAPSS(EM, m.getBytes()));
```

Exercise 3:

Combine RSA and EMSA-PSS in the same way as described in chapter 3. You should write two functions:

1) `byte[] createRSAPSS(byte[] msg)`
2) `boolean verifyRSAPSS(byte[] msg, byte[] signature)`

The first one should create RSA-PSS signature and return it in byte form. The second one should verify sginature and return true if it is valid (false otherwise).

Verification in *main* function:

```
String m = ”message”;

byte[] signature = createRSAPSS(m.getBytes());

System.out.println(verifyRSAPSS(m.getBytes(), signature));
```