

# Adatbiztonság a gazdaságinformatikában

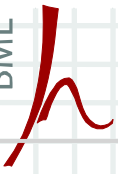
*Cryptolibs – slides from Laszlo Dora*

Dr. Bencsáth Boldizsár  
adjunktus

BME Híradástechnikai Tanszék  
bencsath@crysys.hit.bme.hu



2011. november  
19.  
Budapest

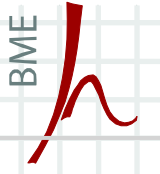


# Crypto libraries available on the market

Name, link	Language	Web, comments
<a href="#">Botan</a>	C++	BSD license
<a href="#">BeeCrypt</a>	C, C++	entropy sources for initializing pseudo-random generators
<a href="#">BouncyCastle</a>	Java, C#	open license
<a href="#">borZoi</a>	C++, Java	ECC library
<a href="#">Cryptix</a>	Java	Berkeley style license
<a href="#">Cryptlib</a>	C	hardware support
<a href="#">Crypto++</a>	C++	
<a href="#">Flexiprovider</a>	Java	
<a href="#">GNU Classpath</a>	Java	
<a href="#">libgcrypt</a>	C	

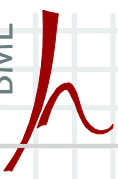
# Crypto libraries cont.

Name, link	Language	Web, comments
<a href="#">MatrixSSL</a>	C	embedded SSL and TLS implementation designed for small footprint applications and devices
<a href="#">MIRACL</a>	C, C++	support for very constrained environments
<a href="#">NaCl</a>	C, Python	
<a href="#">Nettle</a>	C++, Python, Pike, ...	
<a href="#">OpenSSL</a>	<b>C</b>	<b>Apache-style licence</b>
<a href="#">PBC Library</a>	C	Pairing-based cryptography
<a href="#">pidCrypt</a>	Java script	
<a href="#">SJCL</a>	Java script	



# Hash functions – reminder

- a hash function is a function  $H: \{0, 1\}^* \rightarrow \{0, 1\}^n$  that maps arbitrary long messages into a fixed length output
- notation and terminology:
  - $x$  – (input) message
  - $y = H(x)$  – hash value, message digest, fingerprint
- typical application:
  - the hash value of a message can serve as a compact representative image of the message (similar to fingerprints)
    - $H$  is a many-to-one mapping  $\rightarrow$  collisions are unavoidable
    - however, finding collisions are very difficult (practically infeasible)
  - increase the efficiency of digital signatures by signing the hash instead of the message (expensive operation is performed on small data)
- examples:
  - (MD5 and) SHA-1



# Hash functions – example code entered (see the attached files)

- Code

```
/* Variable initialization */
EVP_MD_CTX ctx_hash;
unsigned char md[EVP_MAX_MD_SIZE];
unsigned int md_size;
/* Hashing */
EVP_DigestInit(&ctx_hash, EVP_sha1());
EVP_DigestUpdate(&ctx_hash, plain_text,
plain_size);
EVP_DigestFinal(&ctx_hash, md, &md_size);
```

- Manual

- [http://www.openssl.org/docs/crypto/EVP\\_DigestInit.html](http://www.openssl.org/docs/crypto/EVP_DigestInit.html)

- Output

```
Plain text: Network security 2010
Plain text (hex) (21):
 4E6574776F726B2073656375726974792032303130
Hash calculation delay: 0.010 ms
Message digest (20):
```

BFF91C436E312704098CDEABC5E741C7D8E75A5D

# Block ciphers – reminder

- a block cipher is a function

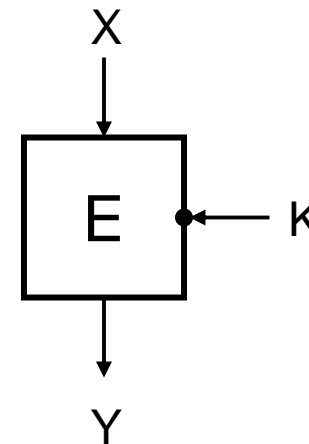
$$E: \{0, 1\}^n \times \{0, 1\}^k \rightarrow \{0, 1\}^n,$$

such that for each  $K \in \{0, 1\}^k$ ,  $E(X, K) = E_K(X)$

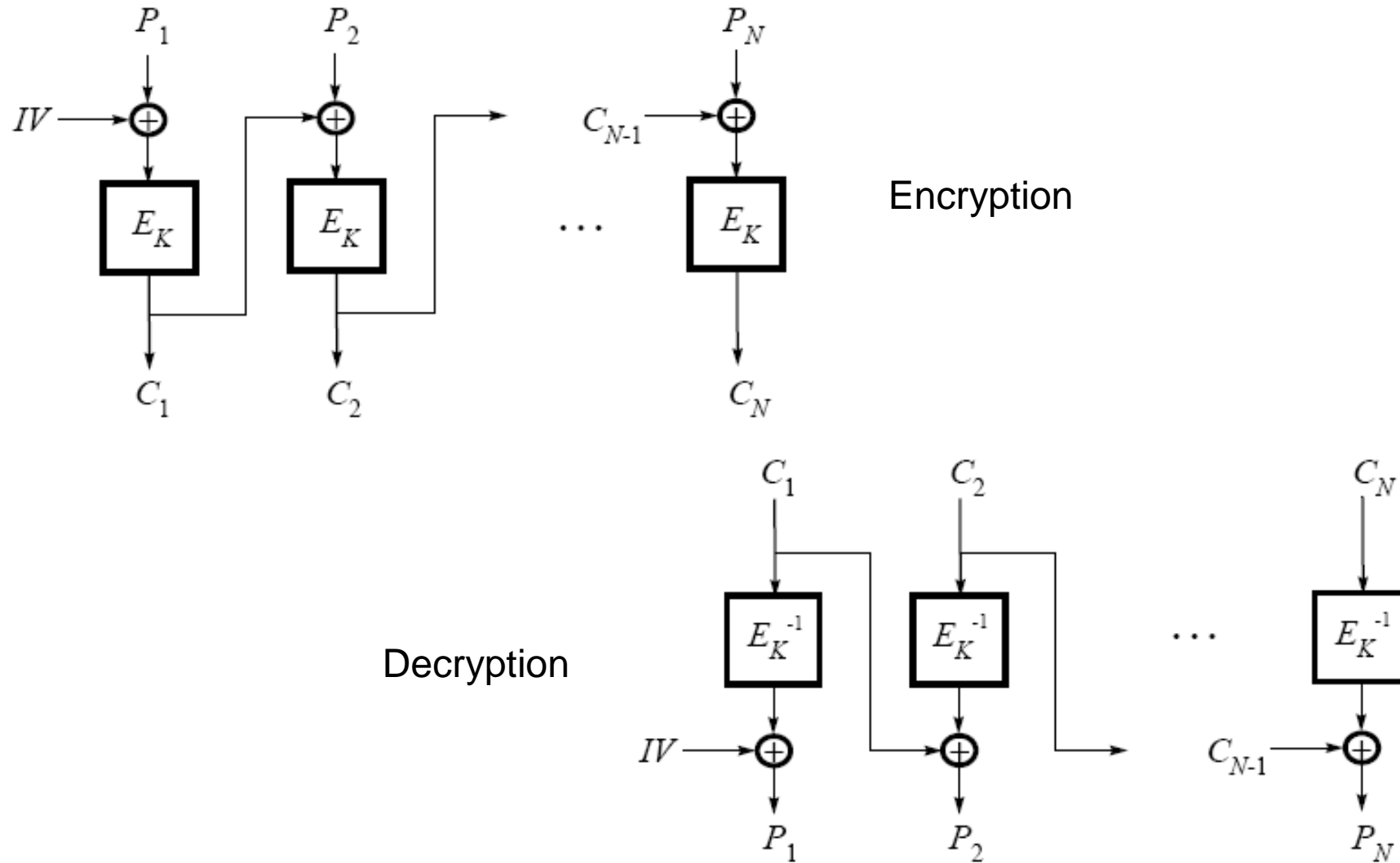
- is an invertible mapping from  $\{0, 1\}^n$  to  $\{0, 1\}^n$  ( $E_K^{-1}(Y) = D_K(Y)$ )
- cannot be efficiently distinguished from a random permutation

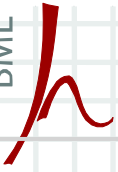
- terminology

- $X$  – plaintext block
- $Y$  – ciphertext block
- $E$  – encryption/coding alg.
- $D$  – decryption/decoding alg.
- $K$  – key
- $\mathcal{K} = \{0, 1\}^k$  – key space



# CBC cipher block mode – reminder





# AES key generation with openssl (see attached code)

- Code

```
/* Generate KEY and IV */  
gen_random_char(key, EVP_MAX_KEY_LENGTH);  
gen_random_char(iv, EVP_MAX_IV_LENGTH);
```

- Manual

- [http://www.openssl.org/docs/crypto/EVP\\_EncryptInit.html](http://www.openssl.org/docs/crypto/EVP_EncryptInit.html)

- Output

Key (32):

395861A955A3CC7E8CD0166CFE589372345DD2F6FA54331  
7B3CF52A3F5301017

IV (16): F28BD273DD72DDB3BE819A435833A633

# AES CBC encryption. OpenSSL implementation

(see attached code)

- Code

```
/* Variable initialization */
EVP_CIPHER_CTX ctx_encrypt;
/* Initialize encryption */
EVP_EncryptInit(&ctx_encrypt,
EVP_aes_256_cbc(), key, iv);
/* Encryption */
EVP_EncryptUpdate(&ctx_encrypt, cipher_text,
&cipher_size, plain_text, plain_size);
EVP_EncryptFinal(&ctx_encrypt,
cipher_text+cipher_size, &output_size);
cipher_size += output_size;
```

- Output

Plain text: Network security 2010

Plain text (hex) (21):

4E6574776F726B2073656375726974792032303130

**Encryption delay: 0.005 ms**

Cipher text (32):

AFBBE3C9A07181C4DDE335A255E44D7D1DF78DB46831D676B  
8E052267E612F01

# AES CBC decryption

- Code

```
/* Variable initialization */
EVP_CIPHER_CTX ctx_decrypt;
/* Initialize decryption */
EVP_DecryptInit(&ctx_decrypt, EVP_aes_256_cbc(),
key, iv);
/* Decryption */
EVP_DecryptUpdate(&ctx_decrypt, new_plain_text,
&new_plain_size, cipher_text, cipher_size);
EVP_DecryptFinal(&ctx_decrypt,
new_plain_text+new_plain_size, &output_size);
new_plain_size += output_size;
```

- Output

Cipher text (32):

AFBBE3C9A07181C4DDE335A255E44D7D1DF78DB46831D676B  
8E052267E612F01

**Decryption delay: 0.005 ms**

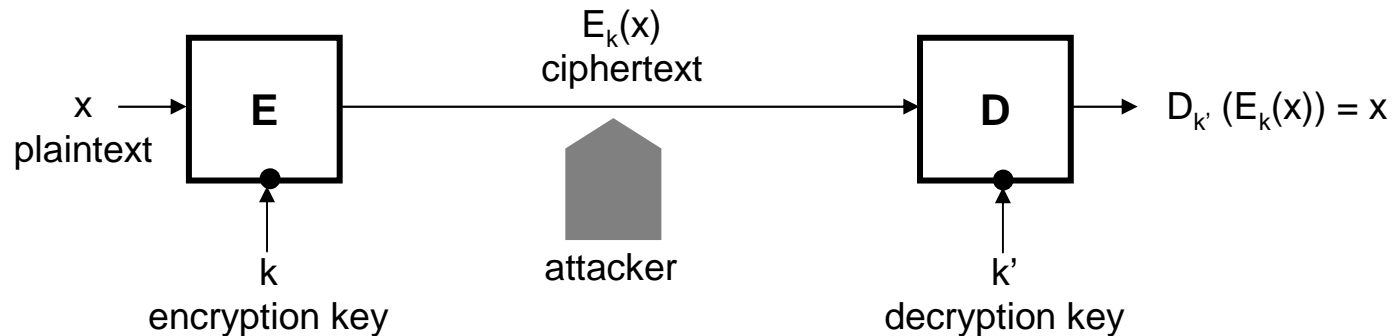
Plain text(hex) (21):

4E6574776F726B2073656375726974792032303130

Plain text: Network security 2010

# Public key encryption – reminder

- classical model of encryption



- symmetric-key encryption:  $k = k'$ 
  - problem: how to setup the same key at the two ends?
- asymmetric-key encryption:  $k \neq k'$ 
  - it is hard (computationally infeasible) to compute  $k'$  from  $k$
  - $k$  can be made public (public-key cryptography)
  - anyone can send messages encrypted with  $k$ , only the intended receiver can decrypt with  $k'$
  - instead of the secrecy of  $k$ , “only” its authenticity and integrity must be ensured

# RSA key generation with OpenSSL

- Code

```
#define RSA_KEY_SIZE 1024
#define RSA_PUB_KEY 65537
/* Variable initialization */
RSA *rsa;
/* Generate RSA key */
rsa = RSA_generate_key(RSA_KEY_SIZE, RSA_PUB_KEY, NULL,
NULL);
```

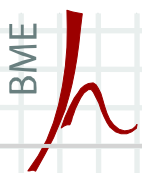
- RSA structure

```
struct {
BIGNUM *n; // public modulus
BIGNUM *e; // public exponent
BIGNUM *d; // private exponent
BIGNUM *p; // secret prime factor
BIGNUM *q; // secret prime factor
BIGNUM *dmp1; // d mod (p-1)
BIGNUM *dmq1; // d mod (q-1)
BIGNUM *iqmp; // q-1 mod p //
...
}; RSA
```

- Only variables with bold letters are defined when only the public part of the key is known, others are set to NULL

- Manual

- <http://www.openssl.org/docs/crypto/rsa.html>



# Sample RSA key – printed with OpenSSL

Private-Key: (1024 bit)

modulus:

```
00:9b:44:bd:42:c8:3d:4f:2d:6b:d4:2f:6d:ba:17:
78:c7:a4:db:85:36:c7:ca:56:2e:5b:2a:4f:99:35:
f2:a3:33:2f:e6:68:f2:83:fd:59:ef:e7:3d:d4:3c:
a6:08:0a:a5:68:ca:5f:14:40:6d:86:48:a4:55:f6:
39:02:6e:b9:ce:f5:26:1a:32:97:a3:1f:b4:cf:9b:
03:da:9e:11:a3:f2:7c:9b:9a:6d:f2:31:92:ac:00:
6f:c3:c3:f9:5c:96:86:5e:74:26:95:d7:49:eb:da:
c8:79:0d:5a:80:6f:9b:8a:ed:05:4d:fe:3c:97:22:
86:65:2b:61:27:3e:b0:a7:65
```

publicExponent: 65537 (0x10001)

privateExponent:

```
5f:af:0d:bf:10:ed:0f:55:1b:65:28:51:43:63:e1:
8f:8e:9d:a5:4b:6b:f5:da:04:39:34:2a:d7:6c:f2:
78:f6:3b:67:8f:77:1c:35:cd:ed:d4:a0:3a:a1:a1:
63:c5:43:c7:ff:26:76:b1:79:8d:4e:48:cf:9c:ab:
e7:3e:db:0b:a4:34:48:ba:c2:49:60:36:b2:77:25:
7b:69:be:91:b1:2e:df:2a:11:a9:9a:58:38:b7:9e:
d8:95:9a:6a:de:1f:91:20:b2:ff:77:42:2a:25:f9:
2f:39:fd:e1:21:7d:b1:b2:0d:86:0b:d4:11:af:29:
02:64:45:b3:09:85:76:a1
```

prime1:

```
00:c9:53:a6:01:40:3b:96:36:8c:67:63:b7:30:2f:
5b:a0:ca:9c:08:d6:a9:4e:95:c9:31:34:16:9a:a9:
fb:74:d4:48:10:15:7b:15:02:ce:4f:3e:d0:6a:ee:
```

```
03:71:b2:1c:7f:d8:71:d4:99:54:25:fc:6d:87:67:
9c:d7:71:35:49
```

prime2:

```
00:c5:6f:18:78:cc:14:5d:3a:d3:d0:2a:6e:99:1e:
c0:2c:1d:b7:2c:d3:7f:73:2e:5b:e2:b0:73:d6:80:
6b:33:49:ff:9e:e5:2e:cc:d7:22:42:9a:54:03:16:
7f:3d:5e:91:a0:70:e7:b0:c2:2d:af:e4:c5:8c:f0:
7f:ee:30:4d:3d
```

exponent1:

```
72:a8:5d:06:a1:5a:4e:36:4a:c8:27:16:11:2f:27:
73:ef:6f:e5:e8:bb:0b:b2:6a:9e:c7:17:88:85:fe:
5b:8e:fa:6b:8e:90:46:6b:0e:ac:3b:0d:df:98:26:
05:fe:76:14:a9:64:4a:bb:f8:1c:9a:22:96:d6:ff:
90:03:3d:61
```

exponent2:

```
00:88:41:44:a5:51:4a:98:90:d9:cf:67:09:4e:f9:
4e:ec:e3:51:20:49:92:42:be:72:c1:7e:bf:63:00:
db:7a:d9:0d:e5:ee:0b:1f:69:35:86:bb:95:51:50:
3c:5d:f1:1f:15:97:f5:fe:21:f4:7f:d8:a5:91:c0:
04:ba:30:71:8d
```

coefficient:

```
53:42:21:a7:25:98:5f:98:29:dc:bf:8c:23:7d:e4:
c1:c1:f3:64:a6:8e:a3:7a:cf:9d:a5:30:db:cc:46:
5c:a8:a5:98:dd:e2:cd:a0:d8:b5:29:be:cd:fc:05:
99:b2:ee:0d:0a:b4:af:f2:b1:c2:3f:ab:f8:49:52:
74:92:f3:96
```

## OpenSSL

(see attached code)

## ■ Code

```
cipher_size = RSA_public_encrypt(plain_size,  
plain_text, cipher_text, rsa,  
RSA_PKCS1_PADDING);
```

## ■ Output

Plain text: Network security 2009

Plain text (hex) (21):

4E6574776F726B2073656375726974792032303039

**Encryption delay: 0.119 ms**

Cipher text (128):

134DD0722A70F859FA539B2D6D981F202E597772FE5A5DF

96D643603E76B5018367367F85E20300BA2FC2DF22EFE58

E1D926DA8AB03E646A64239C12703A52AAC1E5D20D97978

65B5404A038EB7CF2C44A67F62514497159D36E866F18BD

A11B13FB40CD1C640EF8EF2DF6E9C7F2920E15882B93C58

27CE75767085B604A4DFE

- Code

```
new_plain_size =  
RSA_private_decrypt(cipher_size, cipher_text,  
new_plain_text, rsa, RSA_PKCS1_PADDING);
```

- Output

Cipher text (128):

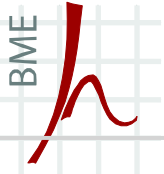
```
134DD0722A70F859FA539B2D6D981F202E597772FE5A5DF  
96D643603E76B5018367367F85E20300BA2FC2DF22EFE58  
E1D926DA8AB03E646A64239C12703A52AAC1E5D20D97978  
65B5404A038EB7CF2C44A67F62514497159D36E866F18BD  
A11B13FB40CD1C640EF8EF2DF6E9C7F2920E15882B93C58  
27CE75767085B604A4DFE
```

**Decryption delay: 2.153 ms**

Plain text(hex) (21):

```
4E6574776F726B2073656375726974792032303039
```

Plain text: Network security 2009



# Digital signature – reminder

- functions (algorithms) and terminology:
  - key-pair generation function  $G() = (K^+, K^-)$ 
    - $K^+$  -- public key
    - $K^-$  -- private key
  - signature generation function  $S(K^-, m) = s$ 
    - $m$  – message
    - $s$  – signature
  - signature verification function  $V: V(K^+, m, s) = \text{accept or reject}$
- services:
  - **message authentication and integrity protection:** after successful verification of the signature, the receiver is assured that the message has been generated by the sender and it has not been altered
  - **non-repudiation of origin:** the receiver can prove this to a third party (hence the sender cannot repudiate)
- examples: RSA, DSA, ECDSA (shorter key and signature length!)

# DSA – reminder

- $H()$  is a hash function
- Public key is  $(p, q, g, y = g^x \text{ mod } p)$
- Private key is  $x$
- Signing
  - Generate a random per-message value  $k$  where  $0 < k < q$
  - Calculate  $r = (g^k \text{ mod } p) \text{ mod } q$
  - Calculate  $s = (k^{-1}(H(m) + x*r)) \text{ mod } q$
  - Recalculate the signature in the unlikely case that  $r=0$  or  $s=0$
  - The signature is  $(r,s)$

# DSA key generation with OpenSSL functions

- Code

```
#define DSA_KEY_SIZE 1024
/* Variable initialization */
DSA *dsa;
/* Generate DSA key */
dsa = DSA_generate_parameters(DSA_KEY_SIZE, NULL, 0,
NULL, NULL, NULL, NULL);
DSA_generate_key(dsa);
```

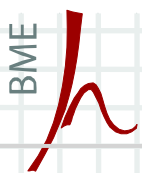
- DSA structure

```
struct {
BIGNUM *p; // prime number
BIGNUM *q; // 160-bit subprime,  $q \mid p-1$ 
BIGNUM *g; // generator of subgroup
BIGNUM *priv_key; // private key  $x$ 
BIGNUM *pub_key; // public key  $y = g^x$ 
...
} DSA;
```

- Only variables with bold letters are defined when only the public part of the key is known, others are set to NULL

- Manual

- <http://www.openssl.org/docs/crypto/dsa.html>



# Sample DSA key printout with OpenSSL tools

Private-Key: (1024 bit)

priv:

3f:2a:b1:c0:e9:d6:0a:64:4e:24:b0:e6:24:04:45:  
ed:fd:6d:2a:4c

pub:

00:e7:74:5e:07:70:7f:37:3e:3d:1e:c9:5c:d8:60:  
eb:9a:eb:f3:fb:ef:3b:db:16:de:5a:65:66:a3:c8:  
68:1c:73:05:28:f1:82:5b:8d:8f:9b:13:bf:b2:bb:  
0a:82:fa:62:65:17:96:3a:5d:4f:88:7e:02:f1:66:  
77:10:02:d3:f3:14:8c:4e:f2:b9:7d:91:f2:73:86:  
9c:36:14:f1:77:02:bc:84:4c:c5:ee:aa:d7:8a:7d:  
e1:52:00:eb:42:23:c8:49:fc:82:37:04:24:50:3a:  
24:a6:af:a9:5a:a4:4f:77:7e:da:80:14:45:2c:07:  
da:fd:21:13:c6:e4:89:e8:57

P:

00:ff:85:83:d2:ec:54:3d:d3:94:bb:79:7b:f0:c4:  
e7:99:b2:94:1f:6d:54:2e:1b:16:81:8c:e1:d5:82:  
21:0d:cd:a3:7c:37:ce:49:a0:0e:69:e7:2b:26:d4:  
8c:20:46:3a:cb:a1:f0:9e:bb:e5:6a:c3:20:15:d1:  
fc:79:f5:f6:dc:79:76:70:04:09:3b:69:b9:c4:e9:  
54:3b:d7:56:0f:84:4d:b8:71:1c:46:1f:f0:47:1d:  
2f:5a:ce:e3:c2:19:b4:5d:9c:c5:77:07:0a:15:7f:  
51:e6:a4:46:43:f6:a5:d0:2d:0e:80:f5:7e:22:78:

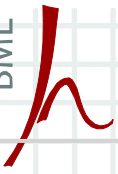
11:29:75:42:ff:a5:5b:8f:8f

Q:

00:bd:e4:41:9e:25:e0:a7:b5:3e:de:71:b5:e4:cf:  
98:56:40:b1:22:6b

G:

79:22:3c:11:f3:d8:2a:14:b0:e9:2c:54:7d:51:84:  
b7:f4:48:a6:ee:29:bb:f5:e6:2d:ae:7a:2d:52:70:  
62:b8:9c:29:18:b3:a8:63:9f:4d:95:4e:0b:09:bf:  
47:e0:dd:5b:4d:a2:1d:b2:b5:94:16:17:fb:89:63:  
92:ca:44:e2:e6:d6:95:64:9a:83:e0:e3:79:e2:7c:  
8a:37:84:42:71:60:fa:85:79:19:d0:cc:55:ba:e3:  
f6:f9:08:a8:61:b7:cd:a9:be:e6:d4:23:aa:a6:da:  
dd:e3:a2:db:4b:6a:4f:e5:79:fe:62:cc:d7:11:ee:  
cf:16:a2:16:fd:98:7d:7d



# DSA signature generation with OpenSSL

- Code

```
/* Optimization */
BIGNUM *rp = NULL;
BIGNUM *kinvp = NULL;
DSA_sign_setup(dsa, NULL, &kinvp, &rp);
dsa->kinv = kinvp;
dsa->r = rp;
/* Digital signature */
DSA_sign(0, plain_text, plain_size, digital_signature,
&digsig_size, dsa);
```

- Comments

- Optimization

- Optional
- Not part of the signature delay

- Hash function is not calculated in this example

- Output

Text: Network security

Text (hex) (16): 4E6574776F726B207365637572697479

**Digital signature delay: 0.023 ms**

Digital signature (46):

302C0214127D7D805A143DB244AD066128F192E05FEC2CBA0214

3A6D3A641739FC6DF59CB00F6F81903BB3D5E600

# DSA signature verification

- Code

```
dsa_verify_result = DSA_verify(0, plain_text,  
plain_size, digital_signature, digsig_size,  
dsa);
```

- Output

Text: Network security

Text (hex) (16):

4E6574776F726B207365637572697479

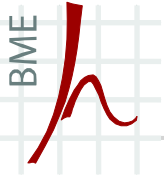
Digital signature (46):

302C0214127D7D805A143DB244AD066128F192E05FEC2CB

A02143A6D3A641739FC6DF59CB00F6F81903BB3D5E600

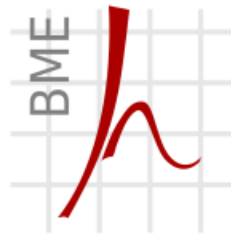
**Verification delay: 0.994 ms**

Verification results: OK



# Kérdések?

## **KÖSZÖNÖM A FIGYELMET!**



Híradástechnikai Tanszék

Dr. Bencsáth Boldizsár  
adjunktus  
BME Híradástechnikai Tanszék  
[bencsath@crysys.hit.bme.hu](mailto:bencsath@crysys.hit.bme.hu)

