Master in Computer Engineering (MEI)
Integrated Master in Informatics Engineering (MiEI)

Specialization Profile **CSI**: Cryptography and Information Security

Engenharia de Segurança

jose.miranda@devisefutures.com

Topics

- Cryptography – revisions of the basic concepts
- Applied Encryption
  - Random / pseudo-random number generator
  - Sharing / Secret Sharing / Splitting
  - Authenticated encryption
  - Key Algorithms and Size - Legacy, Future

Cryptography - Revisions of the basic concepts

- Why do we need cryptography?

Cryptography

- Modern Cryptography
  - It does not focus solely on confidentiality
  - Integrity
    - Hash Functions
    - MAC
    - Digital signatures
  - Fair trade
    - Signing of Contracts
  - Anonymity
    - Electronic Money
    - Electronic voting
  - No Repudiation
    - Asymmetric Cryptography
  - Authenticity
    - MAC
    - Digital signature
  - ...

Cryptography

- Brief History of Cryptography
  - > 2000 years ago: Replacement ciphers
    - Ex.: César cipher

USKQBCQZQGDVXWF

SOUINDICIFRAVEL

A → V	H → L	O → S	V → X
B → E	I → Q	P → R	W → M
C → Z	J → N	Q → I	X → T
D → C	K → H	R → D	Y → J
E → W	L → F	S → U	Z → P
F → G	M → A	T → Y	
G → O	N → B	U → K	

Cryptography

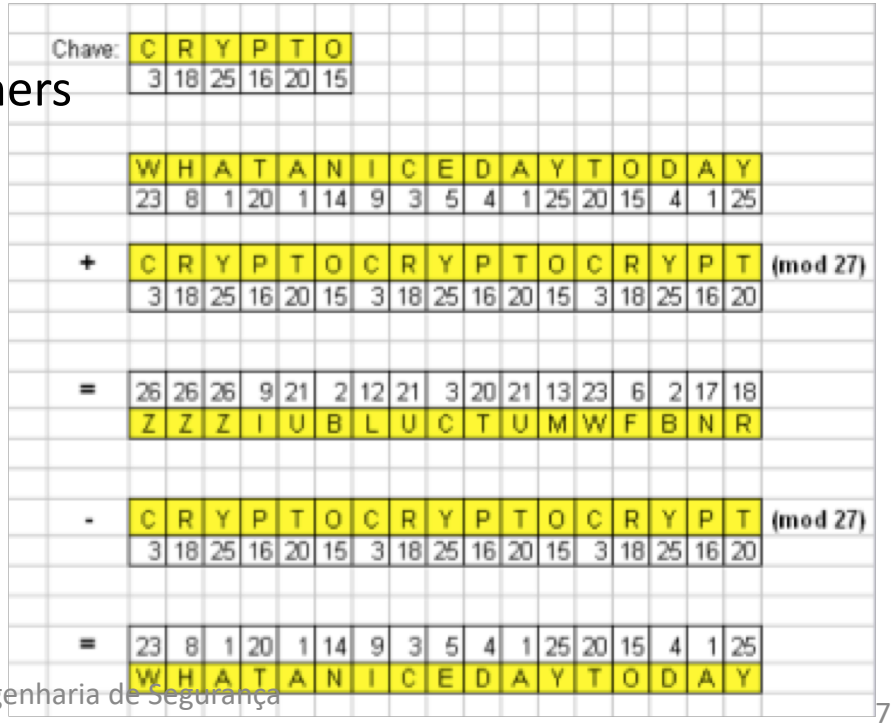
- Brief History of Cryptography
  - > 2000 years ago: Replacement ciphers
  - A few centuries later: Permutation ciphers

HELLOWORLD → LOHLERDLWO

1	2	3	4	5
3	5	4	1	2

Cryptography

- Brief History of Cryptography
  - > 2000 years ago: Replacement ciphers
  - A few centuries later: Permutation ciphers
  - Renaissance: Poli-Alphabetical Ciphers



Cryptography

- Brief History of Cryptography
  - > 2000 years ago: Replacement ciphers
  - A few centuries later: Permutation ciphers
  - Renaissance: Poli-Alphabetical Ciphers
  - 1844: Mechanization
    - Ex.: Enigma (WWII)

Cryptography

- Brief History of Cryptography
  - > 2000 years ago: Replacement ciphers
  - A few centuries later: Permutation ciphers
  - Renaissance: Poli-Alphabetical Ciphers
  - 1844: Mechanization
  - 1976: Public Key Cryptography (PKI)

Cryptography

- Cryptographic systems are generically classified according to three independent dimensions :
  - **Type of operation** used to transform *plaintext* into *ciphertext*. All encryption algorithms are based on two generic principles:
    - substitution - each element in the *plaintext* (bit, letter, group of bits or letters) is mapped to another element
    - *transposition* - *plaintext* elements are rearranged

    Fundamental requirement is that no information is lost, i.e., that all operations are reversible.

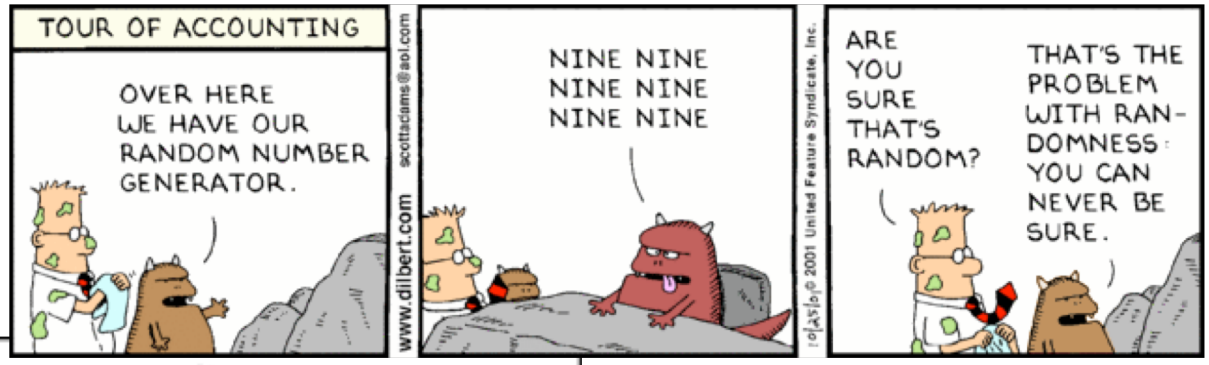
    Most encryption algorithms involve multiple phases of substitutions and transpositions.
  - **Number of keys** used:
    - symmetric, secret key or conventional encryption, if the sender and receiver use the same key
    - asymmetric or public key, if sender and receiver use different keys
  - **How** *plaintext* is **processed**:
    - Block Cipher - Processes the input, one block of elements at a time, producing one block of output per each block of input
    - Stream Cipher - Processes the input elements in a continuous way, producing the output as the input stream passes

Random / pseudorandom number generator

Pseudorandom number generator

- A large number of cryptographic-based security algorithms use random numbers. For example:
  - Session keys;
  - Initialization vectors;
  - Salts;
  - Keys of the RSA public key algorithm.

- **If random numbers are unsafe, any application is unsafe**




```
int getRandomNumber()
{
    return 4;  // chosen by fair dice roll.
               // guaranteed to be random.
}
```

Pseudorandom number generator

- **Two necessary** (not necessarily compatible) characteristics for a "strong" random number sequence:
  - Randomness;
  - Unpredictability.

Randomness

When generating a sequence of random numbers, it is necessary to ensure that the sequence of numbers is random from the point of view of statistical criteria:

- uniform distribution - the distribution of the numbers in the sequence must be uniform; i.e., the frequency of occurrence of each of the numbers should be approximately the same
- independence - no value in the sequence can be inferred from the remaining

Unpredictability

When generating session keys, the question is not so much in randomness, but in the fact that successive members of the sequence are not predictable

(note that in "true" sequences of random numbers, each number is statistically independent of the other numbers in the sequence, thus being unpredictable)

Pseudorandom number generator

- **“Source” of random numbers**
  - **Physical phenomena** expected to be random (atmospheric noise, thermal noise, and other electromagnetic and quantum phenomena), compensating for possible deviations in the measurement process. For example, cosmic radiation or radioactive decay, calculated over short time frames represent natural sources of entropy (entropy seen as a measure of uncertainty).
    - The rate at which entropy can be harvested from natural sources is dependent on the underlying physical phenomena measured. Natural occurrences of "true" entropy are said to be **blocked** until there is sufficient entropy to satisfy the request for random numbers (on Unix-like systems, the /dev/random device blocks until sufficient entropy of the environment is collected);
    - www.random.org claims that it generates "true" random numbers based on atmospheric noise.

Pseudorandom number generator

- **“Source” of random numbers**
  - **Computational algorithms** that produce long sequences of apparently random results, but which are completely determined by an initial value called *seed* or *key*. This type of "source" of random numbers is called pseudo-random number generators and can not be seen as "true" random number generators in their purest sense. However carefully designed and implemented pseudo-random number generators can be accredited for cryptographic purposes.
    - These types of generators do not normally depend on natural sources of entropy. Although they can periodically obtain seeds from natural sources, such generators are **non-blocked**, i.e., they are not limited by entropy rates of external events.

Cryptographically secure pseudorandom number generators

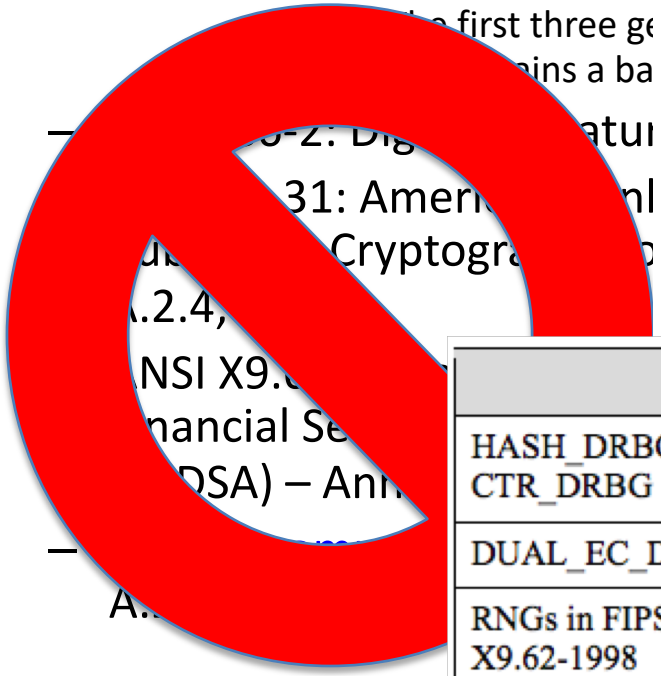
- **Cryptographically secure pseudorandom number generators** are pseudorandom number generators with **properties** suitable for use in cryptography:
- It uses entropy obtained from a high quality source, such as a random number generator in hardware or from unpredictable operating system processes (slow process, to obtain the necessary entropy);
- Satisfies the *next-bit test* - Given the first k bits of a random sequence, there is no polynomial algorithm that provides the (k + 1) -th bit with a success probability greater than 50%;
- It resists to "state compromise extensions" - if part or all of the generator's state is revealed (or calculated correctly), it is impossible to reconstruct the stream of random numbers generated before it has been compromised. In addition, if there is entropy input during generator execution, it must be impracticable to use the knowledge of the input to predict the future conditions of the generator state.

Cryptographically secure pseudorandom number generators

- Standards – Approved Algorithms, according to FIPS 140-2 Anexo C:
  - NIST Special Publication 800-90A: Recommendation for Random Number Generation Using Deterministic Random Bit Generators
    - Hash_DRBG (hash function based), HMAC_DRBG (based on *Hash-based message authentication code*), CTR_DRBG (based on block ciphers), e Dual_EC_DRBG (based on elyptic curves cryptography);
      - Note: The first three generators are considered safe, but the fourth (Dual_EC_DRBG) probably contains a backdoor inserted by the NSA and should not be used.
  - FIPS 186-2: Digital Signature Standard (DSS) – Annex 3.1 and 3.2;
  - ANSI X9.31: American Bankers Association, Digital Signatures Using Reversible Public Key Cryptography for the Financial Services Industry (rDSA) – Annex A.2.4;
  - ANSI X9.62: American Bankers Association, Public Key Cryptography for the Financial Services Industry: The Elliptic Curve Digital Signature Algorithm (ECDSA) – Annex A.4;
  - NIST Recommended Random Number Generator Based on ANSI X9.31 Appendix A.2.4 Using the 3-Key Triple DES and AES Algorithms

Cryptographically secure pseudorandom number generators

- Standards – Approved Algorithms, according to FIPS 140-2 Anexo C:
  - NIST Special Publication 800-90A: Recommendation for Random Number Generation Using Deterministic Random Bit Generators
    - Hash_DRBG (hash function based), HMAC_DRBG (based on *Hash-based message authentication code*), CTR_DRBG (based on block ciphers), e Dual_EC_DRBG (based on elyptic curves cryptography);
  - first three ge
    ins a ba
  - atur
  - 31: Ameri nl
    Cryptogra
    .2.4,
    NSI X9.
    nancial Se
    DSA) – Ann
  - A



Since 1st January 2016, according to SP800-131A Revision 1 Transitions: Recommendation for Transitioning the Use of Cryptographic Algorithms and Key Lengths

Description	Use
HASH_DRBG, HMAC_DRBG and CTR_DRBG	Acceptable
DUAL_EC_DRBG	Disallowed
RNGs in FIPS 186-2, ANS X9.31 and ANS X9.62-1998	Deprecated through 2015 Disallowed after 2015

Cryptographically secure pseudorandom number generators

- Validation of NIST Cryptographic Algorithms - Random Number Generators
- Requirements Document for Random Number Generator Testing: The NIST SP 800-90A Deterministic Random Bit Generator Validation System (DRBGVS)
- List of products (2469) accredited by NIST, including:
  - OpenSSL
  - Bouncy Castle Cryptographic Library
  - VMware Cryptographic Module
  - Windows Embedded Compact Enhanced Cryptographic Provider
  - Linux Kernel crypto API
  - Apple Secure Key Store CoreCrypto Module
  - IBM Java JCE 140-2 Cryptographic Module

Cryptographically secure pseudorandom number generators

- Example using openssl
  - openssl rand [-base64] [-hex] <number of pseudorandom bytes>

```
$ openssl rand -base64 1024
mj8k7yk3Bc0duQviN/250x6C4Hv2NjhyTTEf/L1NGUIDzCjXom1HJ05Y8n4jbN8f
z2ifPaxo2qKSrBBZgIaP3l1IQVIIxZwk2FSLkdl2uyNLf2B8HesxDEGVc1vd1yHZ
3oVrrQUjgQOik0LGLquRW5t0+Dl9zs4poF3pn/SEzHzhm/jQqxONZKcQQYssx9eP
bkkVGJTkUoCTGeaG93LSLZDtvh/QzOZR2ic7DgSaXNIkVpAOTZ/WAK/S7VIxLbrf
IaRm8voczx48dkkfMetSJxk1loasiu4ZYu9HZpZomA8Bqr0td0IkVrP0sSnhpBYd
ZhYIzwERPgW00KQQl8XmZO2Krhef7vQM3w4nSQ5KaKB2OC4cEAGf9BcEEo6Z0Vv8
B7A83I/bTBH1bZUr1Dc+04Ir/ARl/Ng4MYkxrW1YBkV9j1oduLu9hsGmm1tXVvgu
zbu19MYyk9L3Ug6VFaCoR0H2E83LFy2zImGKPx1z7796audQQxqhDhCddaZcXG50
6u8htF6N9XSoRKwsUnYkBPrLY+/u8pl/Bn9UhrJx14ZZEaIgi19HJukDQGP/0eHi
//ckGCculxlDxqKjEkpt9aa50fuhp6AhW91cnp/S3ucjZXRxHmQzU/xS0WzEPRB9
IqNtFJD6c8emeLmZ8CxTmbDtgAicsWnLo+9tlRTpA2oki/MS7PCfA0DhGGl+vkh2
LFmMHC3aUuggryGkEkKdqSJy5FQhKxIMMB4qLR/MlXsWHS5pAdbifI836cZao3F7
WCBZYe78sYtImCzimUlRCIqSlbTVXEG++45pExkqVFxdRjoYAxC5Ib0QZEmtSUll
o/Kk7iC+C/V7MBMBdfnPmYOfmH5xIWKO+F2qgpNkInlMPW3bMyudbTdtdpf+biOV
JJ1L5AgvoV0IuOPfalSVyH00mJzwIuMeYWzb8JV5QF5kRxQ/d+QMwV/J7A47UKbX
UBvaZRC9eHxYdKPkU/PhIoRVH/i0I+Kvrcuxv20f1S6dGERuno2C14d5ryg81blU
vhzSE0K1STKB1QhrY3zmPoGGcAs5zJsO7VjcrtbnU5M7EXPyJpQfr607e3k0IVv/
8F5IFCRG0tkXDm720lNgnKszCYThtXYBUd0xsM4PVIyhGP/yct4G7f4AxCWbMgSA
/Tw58sJh3VrJ64QEKF1divB95xZ2dqVhdJ50kUrfe60j+JaRlujByhAJRn1jujEv
153yk89pJvy1qd3cBe2JYu47aIqWB8DmMz2BKieP176CgArnbh2RoB0rgfi33+4/
E5HoAk4+5weMNgxMkj4qaVy7TTh+fpupCsdyMLr5KalZRJf8275Vl2FAMyaX+wLN
oLnSWbglZbygGq5X2FEXKg==
```

Cryptographically secure pseudorandom number generators

- Example using the Yarrow algorithm (non-proprietary, opensource and royalty-free) implemented by the Unix (and Unix-like) operating systems on the /dev/random device (and/or /dev/urandom)

```
$ head -c 1024 /dev/random | openssl enc -base64
QKb1WzFNzB3VBZcy/J5krYavsn+vse+r3UkcDsl8VJEca1483L7GL0u1nLksVhRy
Pz0MW4PujfJu7DGUd2QqzdzXGah6e+ImCO4KLlZC+nb2vS75TalrlNQP2+xnp3bx
mq0P5Jcp5E1A1glRDVjwNVU2L3eOIzMLsvMq5yVPMFKhgnjb72Ndypb9M/Daa4XA
XTXJksAt1oTok3TyQKjRHdlE3bbGphEicVDyfQaQvb+Cr4iVk9BaXn2Akh3DnTc6
1dczLCEnqergCYEGCybLPMiWDQZ5nE9jg+welxBTXw0lshqY65J1ukGAvWxs7ZNP
CpEH7/xCQTZc6A6XKnk0Q+9XbKNNERoC46IDCTTt1cj5Yyc+iozGsuy2YTECMJB7
MCdZ6IUVaGVuOCNO2zCE0f+/JTLXPlEYTM6m7Lq44h/v8duF54wpxjTvTXEVM3MV
A0jbjFRZTsFHYN2sW2VNQIXagFi7J+VCx+HyYr5Tu8wgZKdqg6sQWB20Ru79LrW4
2jydakeXlbyGGZo8mNM3NKEaObVHGX0qsc0myOTmnYmgsPIddeV1Jwx/sgbR/8Fb
1ALHKhf12trUwDg1Ff/4b6PDei5HUP7eDoGv3vR2HJcPYrLHt5aA6Rz1Wj84mXTM
kSvIecrtdzeCRU3dSOY7dpdYPuIeaRjADD9FKtb5RFMu+1uuw/tRlUztTmUETbPI
TI9yBCsAUcBJJHlaCMVS/MG5SapVXovZ8DEaHeGdb4B83jPC0nm/fJXWD+bubMYC
gnceujuHYFBj9Nfn9hTSJ31q4Yyz1IQTUg0GkHGnA/eT/qax3NpG5dy2TnBGLHht
rQfKzp0fe+F0K71D6/7m/MHqT03yURC3+iSPk5VjZMnnAYpvnVFjBVFC8MfqIqaT
0m7nEAts0iw0WAbxdfvrreulAtBeFxBZKmP3KsQ8mprXNUH6eStX4wyB/DreVN0b
c5BvoHeZstBGX1tDNr0k3aaXvHYVS3/JMEV/4O8q8nHWf4vuH9HSADLpSqwF/bU6
iee8XddLoZl6ayYEV7Yo7QMGf1Tprha4rUU/mDGltwqxrfPGM0oU1YpUtuzoXeAV
ts1I7B0DU8RBSxFhwrROFvBjyC2aRUfsgae9gxcn9xRjhGRXiP1PDIuNQ9Mpix2I
6+6Lrp3dEjn1YNPJhTQf6XX3xE+NnJgi1etyBQHOgsubC1rL82SXiXdMCbUtuVrw
kz8afyIaQWBMGFPSDcBWFzZWrla0nvXET6CF9hkQ448BUm/lI7CO3tT1lreQFccz
weITou4BY/cEGmW0mYPJhiFNcAY2rOQkZYjYch4PYcJEiBX1R0d2Vyo4o8hUC08H
Ls/SPQPXC4478IxoSej03Q==
```

Cryptographically secure pseudorandom number generators

- Example using [java.security.SecureRandom]
  - Indicated to be cryptographically strong but not accredited by NIST

```java
import java.security.SecureRandom;

// gera numero de bytes aleatorios
// RandomBytes <numero de Bytes>
public class RandomBytes {
    public static void main(String[] args) throws Exception {
        if (args.length != 1) { // valida que foi fornecido um argumento
            mensagemUso();
            System.exit(1);
        }

        /* Passo 1: Inicializar o Secure Random Generator */
        // Neste caso utiliza-se o algoritmo NativePRNG, que obtém números aleatórios com base no sistema operativo
        SecureRandom secureRandom = SecureRandom.getInstance("NativePRNG");

        /* Passo 2: método nextBytes gera número de bytes (do tamanho do array de bytes) random */
        byte[] bytes = new byte[Integer.parseInt(args[0])];
        secureRandom.nextBytes(bytes);

        // Imprime os bytes
        System.out.write(bytes);

    }

        public static void mensagemUso() {
                System.out.println("RandomBytes - gera bytes aleatórios");
                System.out.println("Para obter o resultado noutro formato (por exemplo base64), adicionar | openssl
                System.out.println("\tSintaxe: java RandomBytes <numero de bytes>");
                System.out.println();
        }

}
```

Secret Sharing/Splitting

Secret Sharing/Splitting

- **Sharing/Splitting of secrecy** refers to the split of a secret by a group of entities, each of which is given a part of a "code" that when together (in whole or in part) allow to rebuild the secret.
- Ideal for storing highly sensitive and highly confidential information:
  - For example, cipher keys, missile launch codes, numbered bank account identifiers, safe code, bitcoins, ...
- Traditional cipher methods are not the most appropriate to ensure both high degree of confidentiality and reliability:
  - When saving a secret (for example, encryption key) we have to choose between keeping the secret in one place for maximum confidentiality or keeping the secret in several places for maximum reliability.
- The sharing/splitting schemes allow to achieve high levels of confidentiality and reliability, according to the needs of the secret in question.

Secret Sharing/Splitting

Simple Sharing/Splitting Scheme

- S is the secret to split/share in binary format
- M are the number of entities among which S will be splitted
- N is the minimum number of entities that have to join to restore S, with N = M
- To each m (except one) entity is given a random number p_m, with the same number of bits of S
- To the last entity is given the result of (S XOR p_1 XOR p_2 XOR ... XOR p_{N-1}) = p_N

For the secret to be restored, it is necessary to join the N entities and the secret is the result of (p_1 XOR p_2 XOR ... XOR p_N)

Secret Sharing/Splitting

Simple Sharing/Splitting of a secret – Example

```
[jepm@ProOne SecretSharing]$ php genSharedSecret.php "CSI - DIUM" 5
Codigo 0: 00100101001010111101101000110110011111011100111010000110010000100100110010100000 10
Codigo 1: 00100011111110010001000000010110100010010000010100100100111001111111001101010110
Codigo 2: 10101101111011110010110110000011101000110010000101011011101000001111100110011000
Codigo 3: 00011011111100100011001110111011111000100000001010110000111100010110100111010001
Codigo 5: 11110011100111001001110100111000100111101001101110000111011110111010111100010000
```

```
[jepm@ProOne SecretSharing]$ php reconstroiSecret.php 10101101111011110010110110000011101000110010000101011011101000001111100110011000 00100101001010111101101000110110011111011100111010000110010000100100110010100000 11110011100111001001110100111000100111101001101110000111011110111010111100010000 00011011111100100011001110111011111000100000001010110000111100010110100111010001 10110100111010001 00100011111111001000100000001011010001001000001010010010011100111111100110101011 0
Segredo: CSI - DIUM
```

Nota: php code in the GitLab directory

Secret Sharing/Splitting

Secret Sharing/Splitting Scheme with asymmetric keys

- S is the secret to share/split
- P_i are public keys
- Q_i are the corresponding private keys
- M are the number of entities among which to share S
- N is the minimum number of entities that have to join to restore S, with $0 < N \leq M$
- Each entity m is given $\{P_{m1}(P_{m2}(...(P_{mN}(S)))), Q_m, m1, m2, ... mN\}$,

For the secret to be restored, it is necessary to join the N entities that have the Q_{m1} to Q_{mN} private keys.

Note: This scheme is not particularly efficient.

Secret Sharing/Splitting

Shamir Secret Sharing/Splitting Scheme

- The basic idea of the Shamir scheme is that 2 points are sufficient to define a line, 3 points to define a parabola, 4 points to define a cubic curve, and so on.
- That is, we need N points to define an N-1 degree polynomial.
- Suppose you want to divide the secret S by M entities, being N entities necessary to restore the secret.
- Without loss of generality, S is an element of a finite body (Galois body) F of size P, where
  - $0 < N \leq M < P$,
  - $S < P$ and
  - P is a prime number
- Randomly select N-1 positive integers a_1, ..., a_{N-1}, with $a_i < P$ and $a_0 = S$, and construct the polynomial function $f(x) = (a_0 + a_1 x + a_2 x^2 + a_3 x^3 + ... + a_{N-1} x^{N-1}) \bmod P$
- Then compute M points of the polynomial function (i = 1, ..., M) and get (i, f(i)).
- To each entity provide a distinct point (i, f(i)), i.e., a distinct code.
- Given any subset of N codes, you can obtain the polynomial coefficients using the Lagrange polynomial interpolation method, which tells us the following:

$$f(x) \equiv \sum_{j=1}^{N}\left[f(x_j)\prod_{i=1,i\neq j}^{N}\frac{x-x_i}{x_j-x_i}\right] \bmod p$$ where S = f (0), that is

$$S \equiv \sum_{j=1}^{N}\left[f(x_j)\prod_{i=1,i\neq j}^{N}\frac{-x_i}{x_j-x_i}\right] \bmod p \equiv \sum_{j=1}^{N}\left[f(x_j)\prod_{i=1,i\neq j}^{N}x_i\left(x_i-x_j\right)^{-1}\right] \bmod p$$

Secret Sharing/Splitting

Shamir Secret Sharing/Splitting Scheme – simple example

- Suppose that the secret is 1234 (S = 1234) and we split it into 6 parts (M = 6), and any 3 parts (N = 3) are enough to restore the secret.
- Let we choose P = 1613 e randomly N – 1 positive integers: $a_1 = 166$ e $a_2 = 94$, obtaining the polynomial function $f(x) = 1234 + 166\,x + 94\,x^2 \bmod 1613$
- Compute 6 points/codes (i, f(i) mod P):
  - $C_1 = (1, 1494)$, $C_2 = (2, 329)$, $C_3 = (3, 965)$, $C_4 = (4, 176)$, $C_5 = (5, 1188)$, $C_6 = (6, 775)$
- To restore the secret, we need any 3 codes – suppose C_2, C_4 and C_5
- Using the Lagrange polynomial interpolation method:

$$S \equiv \sum_{j=1}^{3}\left[f(x_j)\prod_{i=1,i\neq j}^{3} x_i\left(x_i - x_j\right)^{-1}\right] \bmod p \equiv \left[329\times 4(4-2)^{-1}\times 5(5-2)^{-1}\right]+\left[176\times 2(2-4)^{-1}\times 5(5-4)^{-1}\right]+\left[1188\times 2(2-5)^{-1}\times 4(4-5)^{-1}\right] \bmod 1613$$

$$\equiv \left[329\times 4(2)^{-1}\times 5(3)^{-1}\right]+\left[176\times 2(-2)^{-1}\times 5(1)^{-1}\right]+\left[1188\times 2(-3)^{-1}\times 4(-1)^{-1}\right] \bmod 1613$$

$$\equiv \left[329\times 4(2)^{-1}\times 5(3)^{-1}\right]+\left[176\times 2(1611)^{-1}\times 5(1)^{-1}\right]+\left[1188\times 2(1610)^{-1}\times 4(1612)^{-1}\right] \bmod 1613$$

$$\equiv \left[329\times 4(807)\times 5(538)\right]+\left[176\times 2(806)\times 5(1)\right]+\left[1188\times 2(1075)\times 4(1612)\right] \bmod 1613$$

$$\equiv \left[2856812280+1418560+16469481600\right] \bmod 1613$$

$$\equiv 1234$$

Note that:

- $-n \bmod p \Leftrightarrow (p - n) \bmod p$
- $n^{-1} \bmod p \Leftrightarrow$ find na integer m, susch that $m \cdot n \equiv 1 \bmod p$, or $n^{-1} = m \bmod p$ (computed using the extended Euclid algorithm)

Secret Sharing/Splitting

Shamir Secret Sharing/Splitting Scheme – Example

- Perl code available at http://charles.karney.info/misc/secret.html (and in the Gitlab directory)

```
[jepm@ProOne Shamir]$ echo "CSI - DIUM" | perl shares.pl 3 7
3:1:3100b931a4821c0c356a:
3:2:b74732e8456949840910:
3:3:d427b64311d6cbb0d240:
3:4:88a1434408c8a1908efa:
3:5:d4b4dbeb2a3fcc243e3c:
3:6:b7607c36773c4b6de308:
3:7:31a62727efbf1f6a7b5e:
```

```
[jepm@ProOne Shamir]$ perl reconstruct.pl <<EOF
> 3:2:b74732e8456949840910:
> 3:6:b7607c36773c4b6de308:
> 3:1:3100b931a4821c0c356a:
> EOF
CSI - DIUM
```

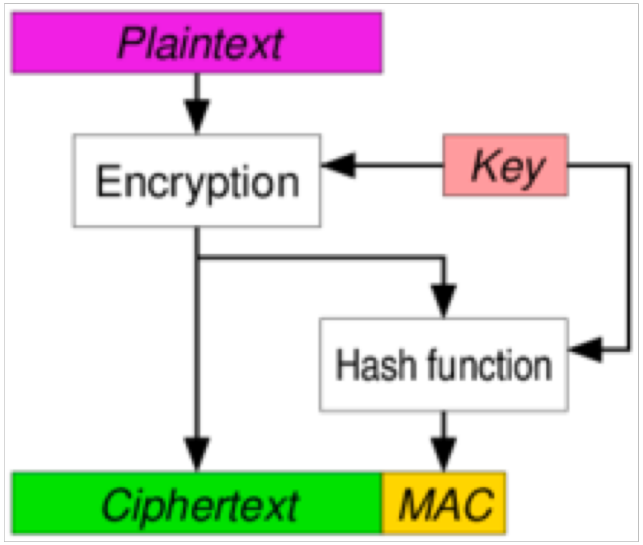
Authenticated Encryption

Authenticated Encryption

- A **form of cipher that simultaneously guarantees confidentiality, integrity and authenticity of the data**, typically through:
  - Cipher
    - Input: *plaintext* to encrypt, encryption key, and optionally a header in *plaintext* that will not be encrypted, but the authenticity is ensured.
    - Output: *ciphertext* and authentication field (MAC or HMAC).
  - Decipher
    - Input: *ciphertext*, decipher key, authentication field, and optionally a header.
    - Output: *plaintext*, or an error if the authentication field does not conform to the ciphertext or header.
  - Note: The header can be entered to ensure metadata integrity and authenticity, for which confidentiality is not required, but authenticity is desirable.
- Authenticated Encryption is most commonly used with symmetric block ciphers, but generically combines encryption with authentication (MAC), provided that:
  - The cipher is semantically secure, under a chosen *plaintext* attack.
  - The MAC function is impossible to fake under a chosen message attack.

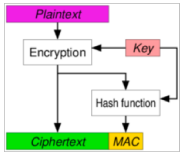
Authenticated Encryption

- Authenticated Encryption schemes:
  - EtM (Encrypt-then-MAC)
    - Used in IPsec, amongst others.

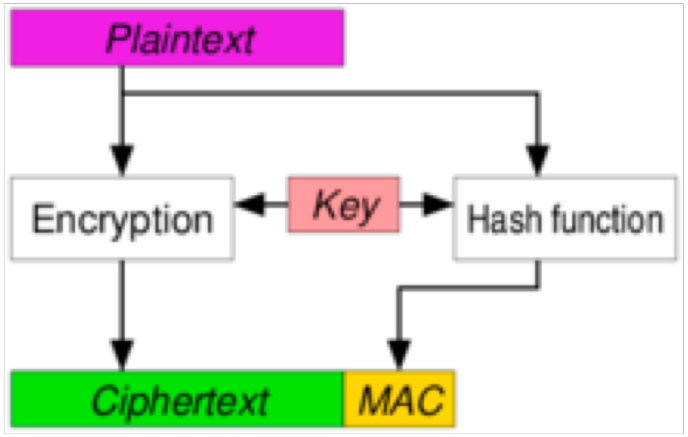


Authenticated Encryption

- Authenticated Encryption schemes :
  - EtM (Encrypt-then-MAC)
    - Used in IPsec, amongst others.

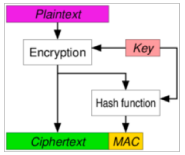


  - E&M (Encrypt-and-MAC)
    - Used in ssh, amongst others.

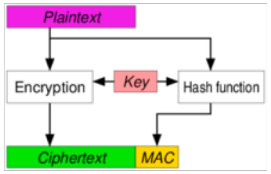


Authenticated Encryption

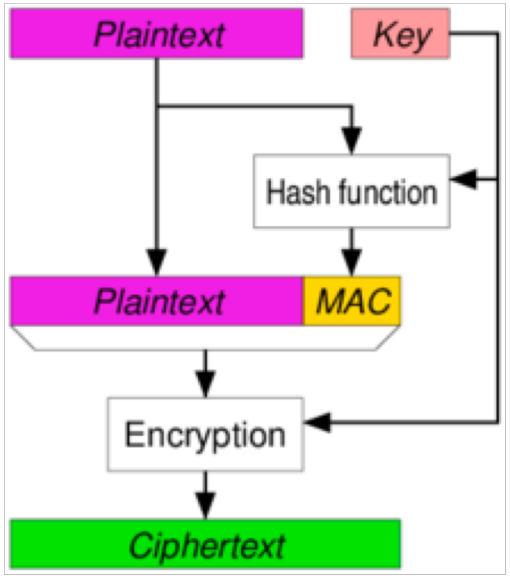
- Authenticated Encryption schemes :
  - EtM (Encrypt-then-MAC)
    - Used in IPsec, amongst others.



  - E&M (Encrypt-and-MAC)
    - Used in ssh, amongst others.

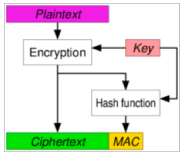


  - MtE (MAC-then-Encrypt)
    - Used in SSL/TLS, amongst others.

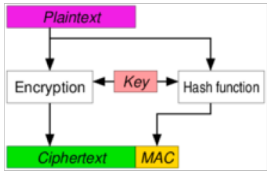


Authenticated Encryption

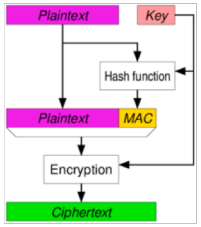
- Authenticated Encryption schemes :
  - EtM (Encrypt-then-MAC)
    - Used in IPsec, amongst others.



  - E&M (Encrypt-and-MAC)
    - Used in ssh, amongst others.



  - MtE (MAC-then-Encrypt)
    - Used in SSL/TLS, amongst others.



- Additionally, Authenticated Encryption can introduce security against chosen ciphertext attacks. How?

Algorithms and Key Size - Legacy, Future

Cryptography

- Encryption system is computationally secure if and only the following criteria are simultaneously verified:
  - the cost of breaking the cipher exceeds the value of the encrypted information;
  - the time required to break the cipher exceeds the useful life of the information.

Size of the Key	Nº of different keys	One cipher per µs	10^6 ciphers per µs
32 bits	$2^{32} = 4{,}3 \times 10^9$	2^{31} µs = 35,8 minutos	2,15 ms
56 bits (DES)	$2^{56} = 7{,}2 \times 10^{16}$	2^{55} µs = 1142 anos	10,01 h
128 bits	$2^{128} = 3{,}4 \times 10^{38}$	2^{127} µs = $5{,}4 \times 10^{24}$ anos	$5{,}4 \times 10^{18}$ anos
26 chars (permutation)	$26! = 4{,}03 \times 10^{26}$	2×10^{26} µs = $6{,}4 \times 10^{12}$ anos	$6{,}4 \times 10^6$ anos

[Time required for exhaustive search in key space]

Symmetric Cryptography

- Symmetric Cryptography Algorithms – Rating ENISA (European Union Agency for Network and Information Security)
  - Algorithms, key size and parameters, Nov. 2014

Primitive	Classification Legacy	Future
HC-128	✓	✓
Salsa20/20	✓	✓
ChaCha	✓	✓
SNOW 2.0	✓	✓
SNOW 3G	✓	✓
SOSEMANUK	✓	✓
Grain	✓	✗
Mickey 2.0	✓	✗
Trivium	✓	✗
Rabbit	✓	✗
A5/1	✗	✗
A5/2	✗	✗
E0	✗	✗
RC4	✗	✗

Table 3.4: Stream Cipher Summary

Primitive	Classification Legacy	Future
AES	✓	✓
Camellia	✓	✓
Three-Key-3DES	✓	✗
Two-Key-3DES	✓	✗
Kasumi	✓	✗
Blowfish$^{\geq 80}$-bit keys	✓	✗
DES	✗	✗

Table 3.2: Block Cipher Summary

Classification	Meaning
Legacy ✗	Attack exists or security considered not sufficient. Mechanism should be replaced in fielded products as a matter of urgency.
Legacy ✓	No known weaknesses at present. Better alternatives exist. Lack of security proof or limited key size.
Future ✓	Mechanism is well studied (often with security proof). Expected to remain secure in 10-50 year lifetime.

Cryptographic Hash Functions

- Cryptographic Hash Function Algorithms – Rating ENISA
  - Algorithms, key size and parameters, Nov. 2014

Primitive	Output Length	Classification Legacy	Classification Future
SHA-2	256, 384, 512	✓	✓
SHA3	256,384,512	✓	✓
Whirlpool	512	✓	✓
SHA3	224	✓	✗
SHA-2	224	✓	✗
RIPEMD-160	160	✓	✗
SHA-1	160	✓[1]	✗
MD-5	128	✗	✗
RIPEMD-128	128	✗	✗

Table 3.3: Hash Function Summary

Classification	Meaning
Legacy ✗	Attack exists or security considered not sufficient. Mechanism should be replaced in fielded products as a matter of urgency.
Legacy ✓	No known weaknesses at present. Better alternatives exist. Lack of security proof or limited key size.
Future ✓	Mechanism is well studied (often with security proof). Expected to remain secure in 10-50 year lifetime.

Public Key Cryptography

- Public Key Cryptography Algorithms – Rating ENISA
  - Algorithms, key size and parameters, Nov. 2014

Primitive	Parameters	Legacy System Minimum	Future System Minimum
RSA Problem	N, e, d	$\ell(n) \geq 1024$, $e \geq 3$ or 65537, $d \geq N^{1/2}$	$\ell(n) \geq 3072$ $e \geq 65537$, $d \geq N^{1/2}$
Finite Field DLP	p, q, n	$\ell(p^n) \geq 1024$ $\ell(p), \ell(q) > 160$	$\ell(p^n) \geq 3072$ $\ell(p), \ell(q) > 256$
ECDLP	p, q, n	$\ell(q) \geq 160, \star$	$\ell(q) > 256, \star$
Pairing	p, q, n, d, k	$\ell(p^{k \cdot n}) \geq 1024$ $\ell(p), \ell(q) > 160$	$\ell(p^{k \cdot n}) \geq 3072$ $\ell(p), \ell(q) > 256$

Table 3.5: Public Key Summary

Classification	Meaning
Legacy ✗	Attack exists or security considered not sufficient. Mechanism should be replaced in fielded products as a matter of urgency.
Legacy ✓	No known weaknesses at present. Better alternatives exist. Lack of security proof or limited key size.
Future ✓	Mechanism is well studied (often with security proof). Expected to remain secure in 10-50 year lifetime.

Algorithms and Key Size

- Key size
  - Algorithms, key size and parameters, ENISA, Nov. 2014

1. Block Ciphers: For near term use we advise AES-128 and for long term use AES-256.
2. Hash Functions: For near term use we advise SHA-256 and for long term use SHA-512.
3. Public Key Primitive: For near term use we advise 256 bit elliptic curves, and for long term use 512 bit elliptic curves.

	Parameter	Legacy	Future System Use	
			Near Term	Long Term
Symmetric Key Size	k	80	128	256
Hash Function Output Size	m	160	256	512
MAC Output Size	m	80	128	256*
RSA Problem	$\ell(n) \geq$	1024	3072	15360
Finite Field DLP	$\ell(p^n) \geq$	1024	3072	15360
	$\ell(p), \ell(q) \geq$	160	256	512
ECDLP	$\ell(q) \geq$	160	256	512
Pairing	$\ell(p^{k \cdot n}) \geq$	1024	3072	15360
	$\ell(p), \ell(q) \geq$	160	256	512

Algorithms and Key Size

- Key size (https://www.keylength.com)
  - NIST Recommendation, 2012

TDEA (Triple Data Encryption Algorithm) and AES are specified in [10].
Hash (A): Digital signatures and hash-only applications.
Hash (B): HMAC, Key Derivation Functions and Random Number Generation.
The security strength for key derivation assumes that the shared secret contains sufficient entropy to support the desired security strength. Same remark applies to the security strength for random number generation.

Date	Minimum of Strength	Symmetric Algorithms	Factoring Modulus	Discrete Logarithm Key	Discrete Logarithm Group	Elliptic Curve	Hash (A)	Hash (B)
2010 (Legacy)	80	2TDEA*	1024	160	1024	160	SHA-1** SHA-224 SHA-256 SHA-384 SHA-512	SHA-1 SHA-224 SHA-256 SHA-384 SHA-512
2011 - 2030	112	3TDEA	2048	224	2048	224	SHA-224 SHA-256 SHA-384 SHA-512	SHA-1 SHA-224 SHA-256 SHA-384 SHA-512
> 2030	128	AES-128	3072	256	3072	256	SHA-256 SHA-384 SHA-512	SHA-1 SHA-224 SHA-256 SHA-384 SHA-512
>> 2030	192	AES-192	7680	384	7680	384	SHA-384 SHA-512	SHA-224 SHA-256 SHA-384 SHA-512
>>> 2030	256	AES-256	15360	512	15360	512	SHA-512	SHA-256 SHA-384 SHA-512

Algorithms and Key Size

- Key size (https://www.keylength.com)
  - NIST Recommendation, 2016

TDEA (Triple Data Encryption Algorithm) and AES are specified in [10].
Hash (A): Digital signatures and hash-only applications.
Hash (B): HMAC, Key Derivation Functions and Random Number Generation.
The security strength for key derivation assumes that the shared secret contains sufficient entropy to support the desired security strength. Same remark applies to the security strength for random number generation.

Date	Minimum of Strength	Symmetric Algorithms	Factoring Modulus	Discrete Logarithm		Elliptic Curve	Hash (A)	Hash (B)
				Key	Group			
(Legacy)	80	2TDEA*	1024	160	1024	160	SHA-1**	
2016 - 2030	112	3TDEA	2048	224	2048	224	SHA-224 SHA-512/224 SHA3-224	
2016 - 2030 & beyond	128	AES-128	3072	256	3072	256	SHA-256 SHA-512/256 SHA3-256	SHA-1
2016 - 2030 & beyond	192	AES-192	7680	384	7680	384	SHA-384 SHA3-384	SHA-224 SHA-512/224
2016 - 2030 & beyond	256	AES-256	15360	512	15360	512	SHA-512 SHA3-512	SHA-256 SHA-512/256 SHA-384 SHA-512 SHA3-512

Algorithms and Key Size

- Key size (https://www.keylength.com)
  - ANSSI (France) Recommendation, 2014

Date	Symmetric	Factoring Modulus	Discrete Logarithm Key	Discrete Logarithm Group	Elliptic Curve GF(p)	Elliptic Curve GF(2^n)	Hash
2014 - 2020	100	2048	200	2048	200	200	200
2021 - 2030	128	2048	200	2048	256	256	256
> 2030	128	3072	200	3072	256	256	256

Algorithms and Key Size

- Key size (https://www.keylength.com)
  - NSA Recommendation, 2014

Type	Symmetric	Elliptic Curve	Hash
Secret	128	256	256
Top Secret	256	384	384

All key sizes are provided in bits. These are the minimal sizes for security.
Click on a value to compare it with other methods.

Suite B includes cryptographic algorithms for encryption, hashing, digital signatures and key exchange:

Encryption: Advanced Encryption Standard (AES) - FIPS 197
Hashing: Secure Hash Algorithm (SHA) - FIPS 180-4
Digital Signature: Elliptic Curve Digital Signature Algorithm (ECDSA) - FIPS 186-4
Key Exchange: Elliptic Curve Diffie-Hellman (ECDH) - NIST SP 800-56A

Algorithms and Key Size

- Key size (https://www.keylength.com)
  - NSA Recommendation, 2016

IAD-NSA's goal in presenting the Commercial National Security Algorithm (CNSA) Suite [6] is to provide industry with a common set of cryptographic algorithms that they can use to create products that meet the needs of the widest range of US Government needs.

Type	Symmetric	Factoring (modulus)	Elliptic Curve	Hash
Up to Top Secret	256	3072	384	384

All key sizes are provided in bits. These are the minimal sizes for security.
Click on a value to compare it with other methods.

NSA will initiate a transition to quantum resistant algorithms in the not too distant future. Until this new suite is developed and products are available implementing the quantum resistant suite, NSA will rely on current algorithms. For those partners and vendors that have not yet made the transition to CNSA suite elliptic curve algorithms, the NSA recommend not making a significant expenditure to do so at this point but instead to prepare for the upcoming quantum resistant algorithm transition.

This FAQ provides answers to commonly asked questions regarding the Commercial National Security Algorithm (CNSA) Suite, Quantum Computing and CNSS Advisory Memorandum 02-15.

CNSA suite includes cryptographic algorithms for encryption, hashing, digital signatures and key exchange:
Encryption: Advanced Encryption Standard (AES) - FIPS 197
Hashing: Secure Hash Algorithm (SHA) - FIPS 180-4
Digital Signature: Elliptic Curve Digital Signature Algorithm (ECDSA) - FIPS 186-4
Digital Signature: RSA - FIPS 186-4
Key Exchange: Elliptic Curve Diffie-Hellman (ECDH) - NIST SP 800-56A
Key Exchange: Diffie-Hellman (DH) - IETF RFC 3526
Key Exchange: RSA - NIST SP 800-56B rev 1

Algorithms and Key Size

- Key size (https://www.keylength.com)
  - BSI (Germany) Recommendation, 2015

Date	Factoring Modulus	Discrete Logarithm Key	Discrete Logarithm Group	Elliptic Curve	Hash	
2014 - 2015	1976	224	2048	224	SHA-1(*) RIPEND-160(*) SHA-224	SHA-256 SHA-384 SHA-512 SHA-512/256
2016 - 2021	1976	256	2048	250	SHA-256 SHA-384	SHA-512 SHA-512/256
> 2021	1976	256	2048	250	SHA-256 SHA-384	SHA-512 SHA-512/256

All key sizes are provided in bits. These are the minimal sizes for security.
Click on a value to compare it with other methods.
(*) For digital certificates verification only.

Remarks for RSA:

Recommended algorithm: ISO/IEC 14888-2
For long-term security level, 2048 bits is recommended.

Remarks for discrete logarithm:

Recommended algorithms: ISO/IEC 14888-3 and FIPS 186-4

Remarks and recommended algorithms for elliptic curve:

EC-DSA: ISO/IEC 14888-3, IEEE P1363, FIPS 186-4 and ANSI X9.62-2005
EC-KDSA and EC-GDSA: ISO/IEC 14888-3
Nyberg-Rueppel (before end of 2020): ISO/IEC 9796-3

This report must be used for cryptographic key length in electronic signatures only.

Algorithms and Key Size

- Key size (https://www.keylength.com)
  - BSI (Germany) Recommendation, 2017

Date	Symmetric	Factoring Modulus	Discrete Logarithm Key	Discrete Logarithm Group	Elliptic Curve	Hash	
2017 - 2022	128	2000	250	2000	250	SHA-256 SHA-512/256 SHA-384 SHA-512	SHA3-256 SHA3-384 SHA3-512
> 2022	128	3000	250	3000	250	SHA-256 SHA-512/256 SHA-384 SHA-512	SHA3-256 SHA3-384 SHA3-512