

Mestrado em Engenharia Informática (MEI)

Mestrado Integrado em Engenharia Informática

(MiEI)

Perfil de Especialização **CSI** : Criptografia e Segurança da
Informação

Engenharia de Segurança

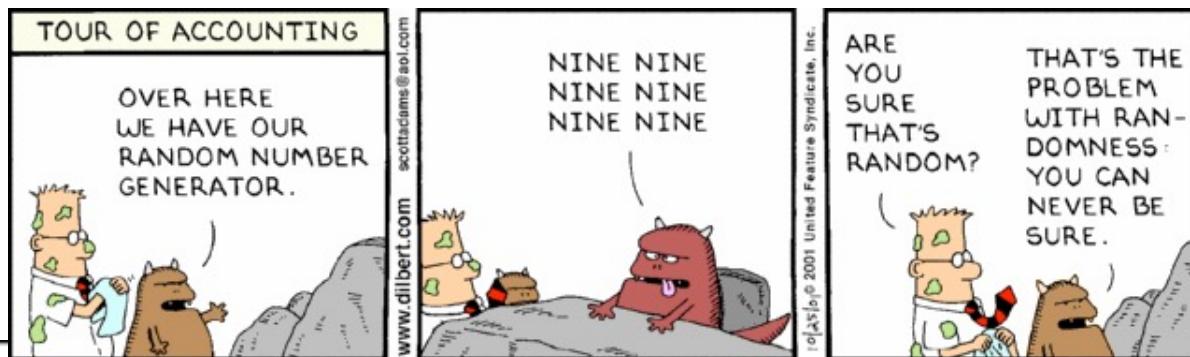
Tópicos

- **Parte IX: Criptografia Aplicada**
 - Gerador de número aleatórios / pseudo-aleatórios
 - Partilha/Divisão de segredo (Secret Sharing/Splitting)
 - Authenticated encryption
 - Algoritmos e tamanho de chaves - Legacy, Futuro
 - Assinaturas cegas (blind signatures)
 - Criptografia homomórfica



Geradores de números pseudoaleatórios

- Um grande número de algoritmos de segurança baseados em criptografia, utiliza números aleatórios. Por exemplo:
 - Chaves de sessão;
 - Vectores de inicialização;
 - Salts;
 - Chaves para o algoritmo de chave pública RSA.
- Se os números aleatórios forem inseguros, toda a aplicação é insegura**



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



Geradores de números pseudoaleatórios

- **Duas características** necessárias (não necessariamente compatíveis) para uma sequência de números aleatórios “forte”:
 - Aleatoriedade;
 - imprevisibilidade.

Aleatoriedade

Na geração de uma sequencia de números aleatórios, é necessário garantir que a sequência dos números é aleatória do ponto de vista de critérios estatísticos:

- distribuição uniforme - a distribuição dos números na sequência deve ser uniforme; i.e., a frequência de ocorrência de cada um dos números deve ser aproximadamente a mesma
- independência - nenhum número na sequência pode ser inferido dos restantes

Imprevisibilidade

Na geração de chaves de sessão, a questão não se põe tanto na aleatoriedade, mas no facto que membros sucessivos da sequência não são previsíveis

(note-se que em “verdadeiras” sequências de números aleatórios, cada número é estatisticamente independente dos outros números da sequência, sendo desse modo imprevisível)



Geradores de números pseudoaleatórios

- “Fonte” de números aleatórios
 - **Fenómenos físicos** expectáveis de serem aleatórios (ruído atmosférica, ruído térmico, e outros fenômenos eletromagnéticos e quânticos), sendo compensados possíveis desvios no processo de medição. Por exemplo, a radiação cósmica ou desintegração radioativa, calculados ao longo de prazos curtos representam **fontes de entropia** (entropia vista como medida de incerteza) naturais.
 - A velocidade a que a entropia pode ser colhida a partir de fontes naturais é dependente dos fenômenos físicos subjacentes medidos. Fontes de ocorrência natural de “verdadeira” entropia dizem-se **bloqueadas** até que exista entropia suficiente para satisfazer o pedido de números aleatórios (em sistemas Unix-like, o dispositivo /dev/random bloqueia até ser recolhida entropia suficiente do ambiente).

Geradores de números pseudoaleatórios

- “Fonte” de números aleatórios
 - Algoritmos computacionais que produzem longas sequências de resultados aparentemente aleatórios, mas que são completamente determinados por um valor inicial denominado de semente ou chave. Este tipo de “fonte” de números aleatórios é designada por geradores de números pseudo-aleatórios e não podem ser vistos como “verdadeiros” geradores de números aleatórios, no seu sentido mais puro. Contudo geradores de números pseudo-aleatórios cuidadosamente desenhados e implementados podem ser credenciados para fins criptográficos.
 - Este tipos de geradores não dependem normalmente de fontes naturais de entropia. Embora possam periodicamente obter *sementes* de fontes naturais, este tipo de geradores são **não-bloqueados**, i.e., não são limitados por taxas de entropia de eventos externos.



Geradores de números pseudoaleatórios criptograficamente seguros

- **Geradores de números pseudoaleatórios criptograficamente seguros** são geradores de números pseudoaleatórios com **propriedades** adequadas para poderem ser utilizados em criptografia:
 - Utiliza entropia obtida de uma fonte de alta qualidade, como um gerador de números aleatórios em hardware ou a partir de processos imprevisíveis do sistema operativo (processo lento, para obter a entropia necessária);
 - Satisfaz o next-bit test – Dado os primeiros bits k de uma sequência aleatória, não existe qualquer algoritmo polinomial que preveja o $(k + 1)$ -ésimo bit com probabilidade de sucesso superior a 50%;
 - Resiste a “extensões de compromisso do estado” – caso parte ou a totalidade do estado do gerador seja revelado (ou calculado corretamente), é impossível reconstruir o fluxo de números aleatórios gerados antes de ter sido comprometido. Além disso, se existe input de entropia durante a execução do gerador, tem de ser inviável utilizar o conhecimento do input para prever as condições futuras do estado do gerador.



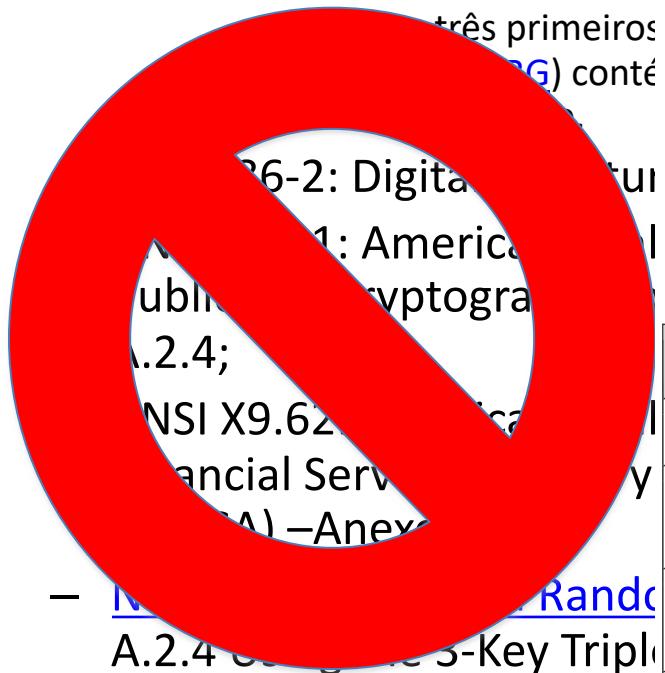
Geradores de números pseudoaleatórios criptograficamente seguros

- Standards – Algoritmos aprovados, de acordo com o [FIPS 140-2 Anexo C](#):
 - [NIST Special Publication 800-90A](#): Recommendation for Random Number Generation Using Deterministic Random Bit Generators
 - Hash_DRBG (baseada em funções de hash), HMAC_DRBG (baseada em *Hash-based message authentication code*), CTR_DRBG (baseada em cifra de blocos), e Dual_EC_DRBG (baseada em criptografia de curvas elípticas);
 - Nota: os três primeiros geradores são considerados seguros, mas o quarto ([Dual EC DRBG](#)) contém provavelmente um *backdoor* inserido pela NSA pelo que não deve ser utilizado.
 - FIPS 186-2: Digital Signature Standard (DSS) – Anexos 3.1 e 3.2;
 - ANSI X9.31: American Bankers Association, Digital Signatures Using Reversible Public Key Cryptography for the Financial Services Industry (rDSA) – Anexo A.2.4;
 - ANSI X9.62: American Bankers Association, Public Key Cryptography for the Financial Services Industry: The Elliptic Curve Digital Signature Algorithm (ECDSA) –Anexo 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



Geradores de números pseudoaleatórios criptograficamente seguros

- Standards – Algoritmos aprovados, de acordo com o [FIPS 140-2 Anexo C](#):
 - [NIST Special Publication 800-90A](#): Recommendation for Random Number Generation Using Deterministic Random Bit Generators
 - Hash_DRBG (baseada em funções de hash), HMAC_DRBG (baseada em *Hash-based message authentication code*), CTR_DRBG (baseada em cifra de blocos), e Dual_EC_DRBG (baseada em criptografia de curvas elípticas);



Desde 1 de Janeiro de 2016 (reiterado em Março de 2019), de acordo com o [SP800-131A Revision 2 Transitioning the Use of Cryptographic Algorithms and Key Lengths](#)

Algorithm	Status
Hash_DRBG and HMAC_DRBG	Acceptable
CTR_DRBG with three-key TDEA	Deprecated through 2023 Disallowed after 2023
CTR_DRBG with AES-128, AES-192 and AES-256AES-256	Acceptable
DUAL_EC_DRBG	Disallowed
RNGs in FIPS 186-2 ⁴² , ANS X9.31 ⁴³ and ANS X9.62-1998	Disallowed

Geradores de números pseudoaleatórios criptograficamente seguros

- Validação de algoritmos criptográficos do NIST – [Geradores de números aleatórios](#)
 - Documento de requisitos para testes dos geradores de números aleatórios:
[The NIST SP 800-90A Deterministic Random Bit Generator Validation System \(DRBGVS\)](#)
- [Lista de produtos](#) (>5000) credenciados pelo NIST, entre os quais:
 - 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



Geradores de números pseudoaleatórios criptograficamente seguros

- NIST está a implementar um serviço público de números aleatórios:
 - [**NIST Randomness Beacon**](#)
- Características:
 - Imprevisibilidade – impossível prever bits, antes dos mesmos serem disponibilizados;
 - Autonomia – resistente a tentativas de alterar a distribuição aleatória de bits;
 - Periodicidade – Periodicamente (e.g., uma vez por minuto) emite/pulsa aleatoriedade;
 - Cada pulsar emite uma nova string aleatória de 512-bit, que combina criptograficamente entropia de pelo menos dois geradores de números aleatórios distintos.
 - Cada pulsar está indexado, tem selo temporal e está assinado.
 - Todos os pulsar emitidos estão acessíveis publicamente.
 - A sequência de pulsares forma uma cadeia de hashs. O que significa?
- Os valores gerados pelo Beacon não devem ser utilizados como chaves secretas criptográficas. Porquê?



Geradores de números pseudoaleatórios criptograficamente seguros



- **NIST Randomness Beacon** (<https://beacon.nist.gov/beacon/2.0/pulse/last>)



Geradores de números pseudoaleatórios criptograficamente seguros

- Exemplo utilizando openssl
 - openssl rand [-base64] [-hex] <número de bytes pseudo-aleatórios>

```
$ openssl rand -base64 1024
mj8k7yk3Bc0duQviN/250x6C4Hv2NjhyTTEf/L1NGUIDzCjXom1HJ05Y8n4jbN8f
z2ifPaxo2qKSrBBZgIaP3l1IQVIIxZwk2FSLkd12uyNLf2B8HesxDEGVc1vd1yHZ
3oVrrQUjgQ0ik0LGLquRW5t0+D19zs4poF3pn/SEzHzhm/jQqx0NZKcQQYssx9eP
bkkVGJTkUoCTGeaG93LSLZDtvh/Qz0ZR2ic7DgSaXNIkVpA0TZ/WAK/S7VIxLbrf
IaRm8voczx48dkkfMetSJxk1loasiu4ZYu9HZpZomA8Bqr0td0IkVrP0sSnhpBYd
ZhYIzwERPgW00KQQ18XmZ02Krhef7vQM3w4nS05KaKB20C4cEAGf9BcEEo6Z0Vv8
B7A83I/bTBH1bZUr1Dc+04Ir/AR1/Ng4MYkxrW1YbkV9j1oduLu9hsGmm1tXVvgu
zbu19MYyk9L3Ug6VFaCoR0H2E83LFy2zImGKPx1z7796audQQxqhDhCddaZcXG50
6u8htF6N9XSoRKwsUnYkBPrLY+/u8pl/Bn9UhrJx14ZZEaIgi19HJukDQGP/OeHi
//ckGCculx1DxqKjEkpt9aa50fuhp6AhW91cnp/S3ucjZXRxHmQzU/xS0WzEPRB9
IqNtFJD6c8emeLmZ8CxTmbDtgaicsWhLo+9t1RTpA2oki/MS7PCfa0DhGG1+vkh2
LFmMHC3aUggryGkEkKdqSJy5FQhKxIMMB4qLR/MlxswHs5pAdbifI836cZao3F7
WCBZYe78sYtImCzimUlRCIqS1bTVXEG++45pExkqVFxdRjoYAxC5Ib0QZEmtSU11
o/Kk7iC+C/V7MBMBdfnPmY0fmH5xIWk0+F2qgpNkIn1MPW3bMyfdbTdt dpf+bi0V
JJ1L5AgvoV0IuOPfa1SVyH00mJzwIuMeYWzb8JV5QF5kRxQ/d+QMnV/J7A47UKbX
UBvaZRC9eHxYdKPkU/PhIoRVH/i0I+Kvrcuxv20f1S6dGERuno2C14d5ryg81b1U
vhzSE0K1STKB1QhrY3zmPoGGcAs5zJs07VjcrtbnU5M7EXPYJpQfr607e3k0IVv/
8F5IFCRG0tkXDm7201NgnKszCYThXYBUd0xsM4PVIyhGP/yct4G7f4AxCWbMgSA
/Tw58sJh3VrJ64QEKF1divB95xZ2dqVhdJ50kUrfe60j+JaR1ujByhAJRn1jujEv
153yk89pJvy1qd3cBe2JYu47aIqWB8DmMz2BKieP176CgArnbh2RoB0rgfi33+4/
E5HoAk4+5weMNgxMkj4qaVy7TTh+fpupCsdylMr5Ka1ZRJf8275V12FAMyaX+wLN
oLnSWbglZbygGq5X2FEXKg==
```

Geradores de números pseudoaleatórios criptograficamente seguros

- Exemplo utilizando o [algoritmo de Yarrow](#) (algoritmo não patenteado, opensource e royalty-free), implementado pelos sistemas operativos Unix (e Unix-like) no dispositivo /dev/random (e/ou /dev/urandom)

```
$ head -c 1024 /dev/random | openssl enc -base64
QKb1WzFNzB3VBZcy/J5krYavsn+vse+r3UkcDs18VJEca1483L7GL0u1nLksVhRy
Pz0MW4PujfJu7DGUD2QqzdzXGah6e+ImC04KL1ZC+nb2vS75Ta1r1NQP2+xnp3bx
mq0P5Jcp5E1A1g1RDVjwNVU2L3e0IzMLsvMq5yVPMFKhgnjb72Ndypb9M/Daa4XA
XTXJksAt1oTok3TyQKjRHdLE3bbGphEicVDyfQaQvb+Cr4iVkJ9BaXn2Akh3DnTc6
1dczLCEnqergCYEGCybLPMiWDQZ5nE9jg+welxBTXw01shqY65J1ukGAvWxs7ZNP
CpEH7/xCQTZc6A6XKnk0Q+9XbKNNERoC46IDCTTt1cj5Yyc+iozGsuy2YTECMJB7
MCdZ6IUvAGVuOCN02zCE0f+/JTLXP1EYTM6m7Lq44h/v8duF54wpkjTvTXEVMBV
A0jbjFRZTsFHYN2sW2VNQIXagFi7J+VCx+HyYr5Tu8wgZKdqg6sQWB20Ru79LrW4
2jydakeXlbyGGZo8mNM3NKEa0bVHG0qsc0my0TmnYmgsPIddeV1Jwx/sgbR/8Fb
1ALHKhf12trUwDg1Ff/4b6PDei5HUP7eDoGv3vR2HJcPYrLht5aA6Rz1Wj84mXTM
kSvIecrtdeCRU3dS0Y7dpdYPuIeaRjADD9FKtb5RFMu+1uuw/tR1UztTmUETbPI
TI9yBCsAUcBJJHlaOMVS/MG5SapVXovZ8DEaHeGdb4B83jPC0nm/fJXND+bubMYC
gnceujuHYFBj9Nfn9hTSJ31q4Yyz1IQTUg0GkHGrA/eT/qax3NpG5dy2TnBGLHht
rQfKzp0fe+F0K71D6/7m/MHqT03yURC3+iSPk5VjZMnnAYpvnVFjBVFC8MfqIqaT
0m7nEAts0iw0WAwdxrvrreulAtBeFxBZKmP3KsQ8mprXNUH6eStX4wyB/DreVN0b
c5BvoHeZstBGX1tDNr0k3aaXvHYVS3/JMEV/408q8nHwf4vuH9HSADLpSqwF/bU6
iee8XddLoZ16ayYEV7Yo7QMGf1Tprha4rUU/mDG1twqxrfPGM0oU1YpUtuzoXeAV
ts1I7B0DU8RBSxFhwrR0FvBjyC2aRUfsgae9gxcn9xRjhGRXiP1PDIuNQ9Mpix2I
6+6Lrp3dEjn1YNPJhTQf6XX3x+E+NnJgi1etyBQHOgsubC1rL82SXixdMCbUtuVrw
kz8afyIaQWBMGFPScBWFzZWrla0nvXET6CF9hkQ448BuM/LI7C03tT1lreQFccz
weITou4BY/cEGmN0mYPJhiFNcAY2r0QkZYjYch4PYcJEiBX1R0d2Vyo4o8hUC08H
Ls/SPQPXC4478IxoSj03Q==
```



Geradores de números pseudoaleatórios criptograficamente seguros

- Exemplo utilizando [java.security.SecureRandom](#)
 - Indicada como sendo criptograficamente forte, mas não está credenciado pelo NIST

```
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();
    }
}
```

Tópicos

- **Parte IX: Criptografia Aplicada**

- Gerador de número aleatórios / pseudo-aleatórios
- **Partilha/Divisão de segredo (Secret Sharing/Splitting)**
- Authenticated encryption
- Algoritmos e tamanho de chaves - Legacy, Futuro
- Assinaturas cegas (blind signatures)
- Criptografia homomórfica

Partilha/Divisão de segredo (Secret Sharing/Splitting)

- **Partilha/Divisão de segredo** refere-se à divisão de um segredo por um grupo de entidades, a cada uma das quais é entregue uma parte de um “código” que quando junto (na totalidade ou em parte) permitem reconstruir o segredo.
- Ideal para guardar informação altamente sensível e altamente confidencial:
 - Por exemplo, chaves de cifra, códigos de lançamento de mísseis, identificadores de contas bancárias numeradas, código de cofre, bitcoins, ...
- Métodos tradicionais de cifra não são os mais adequados para garantir simultaneamente alto grau de confidencialidade e confiabilidade:
 - Ao guardar um segredo (por exemplo, chave de cifra) temos que escolher entre guardar o segredo num único local para máxima confidencialidade ou, guardar o segredo em vários locais para máxima confiabilidade.
- Os esquemas de partilha/divisão de segredo permitem alcançar altos níveis de confidencialidade e confiabilidade, de acordo com as necessidades do segredo em causa.

Partilha/Divisão de segredo (Secret Sharing/Splitting)

Esquema simples de Partilha/Divisão de segredo

- S é o segredo a dividir/partilhar em formato binário
- M são o número de entidades entre as quais se vai dividir S
- N é o número mínimo de entidades que têm de se juntar para aceder a S, com $N = M$
- A cada entidade m (excepto uma) é entregue um número aleatório p_m , com o mesmo número de bits de S
- À última entidade é entregue o resultado de $(S \text{ XOR } p_1 \text{ XOR } p_2 \text{ XOR } \dots \text{ XOR } p_{N-1}) = p_N$

Para o segredo ser novamente reconstituído, é necessário juntar as N entidades e o segredo é o resultado de $(p_1 \text{ XOR } p_2 \text{ XOR } \dots \text{ XOR } p_N)$

Partilha/Divisão de segredo (Secret Sharing/Splitting)

Esquema simples de Partilha/Divisão de segredo – Exemplo

```
[jepm@ProOne SecretSharing]$ php genSharedSecret.php "CSI - DIUM" 5
Codigo 0: 00100101001010111101101000110110011110111001110100001100100001001001100101000010
Codigo 1: 0010001111111001000100000001011010001001000001010010011100111111001101010110
Codigo 2: 101011011110111100101101100001110100011001000010101101110100001111100110011000
Codigo 3: 00011011111100100011001110111100010000000101011000111100010110100111010001
Codigo 5: 11110011100111001001110001001111010011110000111011110111010111100010000
```

```
[jepm@ProOne SecretSharing]$ php reconstroisSecret.php 10101101110111001011011000011101000110010000101011011101000011
11100110011000 00100101001010111101101000110110011110111001110100001100100001001001100101000010 11110011100111001001110
00111000100111101001101110000111011110111010111100010000 000110111110010001100111011110001000000010101100001111000
10110100111010001 001000111111001000100000010110100010010000010100100111100111111001101010110
```

Segredo: CSI - DIUM

Nota: código php na directória do GitLab



Partilha/Divisão de segredo (Secret Sharing/Splitting)

Esquema de Partilha/Divisão de segredo com chaves assimétricas

- S é o segredo a dividir/partilhar
- P_i são chaves públicas
- Q_i são as correspondentes chaves privadas
- M são o número de entidades entre as quais se vai partilhar S
- N é o número mínimo de entidades que têm de se juntar para aceder a S, com $0 < N \leq M$
- A cada entidade m é entregue $\{P_{m1}(P_{m2}(\dots(P_{mN}(S))))\}, Q_m$, m1, m2, ... mN},

Para o segredo ser novamente reconstituído, é necessário juntar as N entidades que tenham as Q_{m1} a Q_{mN} chaves privadas

Nota: este esquema não é particularmente eficiente.



Partilha/Divisão de segredo (Secret Sharing/Splitting)

Esquema de Shamir para Partilha/Divisão de segredo

- A ideia base do esquema de Shamir é que 2 pontos são suficientes para definir uma linha, 3 pontos para definir uma parábola, 4 pontos para definir uma curva cúbica e assim por diante.
- Isto é, necessitamos de N pontos para definir um polinómio de grau N-1.
- Suponha que quer dividir o segredo S por M entidades sendo necessárias N para recuperar o segredo.
- Sem perda de generalidade, S é um elemento de um corpo finito (corpo de Galois) F de tamanho P, em que
 - $0 < N \leq M < P$,
 - $S < P$ e
 - P é um número primo
- Escolha aleatoriamente N-1 inteiros positivos a_1, \dots, a_{N-1} , com $a_i < P$ e $a_0 = S$, construindo deste modo a função polinomial $f(x) = (a_0 + a_1 x + a_2 x^2 + a_3 x^3 + \dots + a_{N-1} x^{N-1}) \bmod P$
- De seguida, calcule M pontos da função polinomial ($i = 1, \dots, M$) e obtenha $(i, f(i))$.
- A cada entidade forneça um ponto $(i, f(i))$ distinto, i.e., um código distinto.
- Dado qualquer subconjunto de N códigos, pode obter os coeficientes do polinómio utilizando o método de interpolação polinomial de Lagrange, que nos indica o seguinte:

$$f(x) \equiv \sum_{j=1}^N \left[f(x_j) \prod_{i=1, i \neq j}^N \frac{x - x_i}{x_j - x_i} \right] \bmod p \quad \text{sendo } S = f(0), \text{ ou seja}$$

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

Partilha/Divisão de segredo (Secret Sharing/Splitting)

Esquema de Shamir para Partilha/Divisão de segredo – Exemplo simples

- O segredo é 1234 ($S = 1234$) e vamos dividi-lo em 6 partes ($M = 6$), e quaisquer 3 partes ($N = 3$) são suficientes para reconstruir o segredo.
- Escolhemos $P = 1613$ e aleatoriamente $N - 1$ inteiros positivos: $a_1 = 166$ e $a_2 = 94$, construindo a função polinomial $f(x) = 1234 + 166x + 94x^2 \pmod{1613}$
- Calculamos 6 pontos/códigos ($i, f(i) \pmod{P}$):
 - $C_1 = (1, 1494), C_2 = (2, 329), C_3 = (3, 965), C_4 = (4, 176), C_5 = (5, 1188), C_6 = (6, 775)$
- Para reconstruirmos o segredo, necessitamos de quaisquer 3 códigos – assumimos que são C_2, C_4 e C_5
- Utilizando o método de interpolação polinomial de Lagrange obtemos

$$\begin{aligned}
 S &= \sum_{j=1}^3 \left[f(x_j) \prod_{i=1, i \neq j}^3 x_i (x_i - x_j)^{-1} \right] \pmod{p} = [329 \times 4(4-2)^{-1} \times 5(5-2)^{-1}] + [176 \times 2(2-4)^{-1} \times 5(5-4)^{-1}] + [1188 \times 2(2-5)^{-1} \times 4(4-5)^{-1}] \pmod{1613} \\
 &= [329 \times 4(2)^{-1} \times 5(3)^{-1}] + [176 \times 2(-2)^{-1} \times 5(1)^{-1}] + [1188 \times 2(-3)^{-1} \times 4(-1)^{-1}] \pmod{1613} \\
 &= [329 \times 4(2)^{-1} \times 5(3)^{-1}] + [176 \times 2(1611)^{-1} \times 5(1)^{-1}] + [1188 \times 2(1610)^{-1} \times 4(1612)^{-1}] \pmod{1613} \\
 &= [329 \times 4(807) \times 5(538)] + [176 \times 2(806) \times 5(1)] + [1188 \times 2(1075) \times 4(1612)] \pmod{1613} \\
 &= [2856812280 + 1418560 + 16469481600] \pmod{1613} \\
 &= 1234
 \end{aligned}$$

Note que:

- $-n \pmod{p} \Leftrightarrow (p - n) \pmod{p}$
- $n^{-1} \pmod{p} \Leftrightarrow$ encontrar um inteiro m , t.q. $m \cdot n \equiv 1 \pmod{p}$, ou dito de outro modo $n^{-1} = m \pmod{p}$ (calcula-se utilizando o algoritmo estendido de Euclides)



Partilha/Divisão de segredo (Secret Sharing/Splitting)

Esquema de Shamir para Partilha/Divisão de segredo – Exemplo

- Código Perl disponibilizado em <http://charles.karney.info/misc/secret.html> e colocado no github

```
[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
```

Tópicos

- **Parte IX: Criptografia Aplicada**

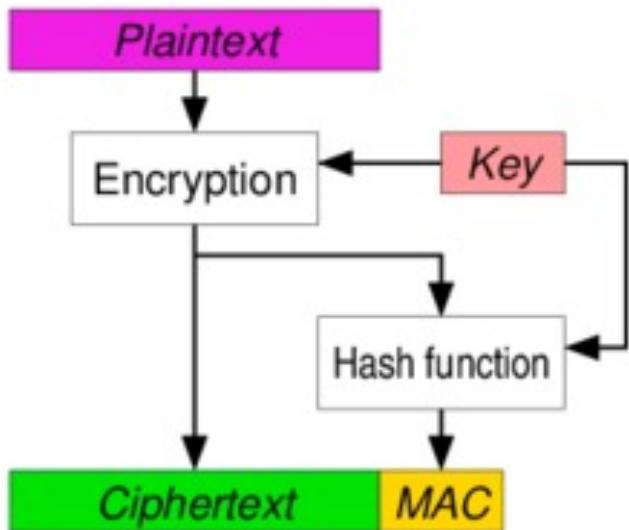
- Gerador de número aleatórios / pseudo-aleatórios
- Partilha/Divisão de segredo (Secret Sharing/Splitting)
- **Authenticated encryption**
- Algoritmos e tamanho de chaves - Legacy, Futuro
- Assinaturas cegas (blind signatures)
- Criptografia homomórfica

Authenticated Encryption

- Forma de cifra que simultaneamente garante confidencialidade, integridade e autenticidade sobre os dados, tipicamente através de:
 - Cifra
 - Input: *plaintext* a cifrar, chave de cifra, e opcionalmente um cabeçalho em *plaintext* que não será cifrado, mas ficará sob a proteção da autenticidade.
 - Output: *ciphertext* e campo de autenticação (MAC ou HMAC).
 - Decifra
 - Input: *ciphertext*, chave de decifra, campo de autenticação, e opcionalmente um cabeçalho.
 - Output: *plaintext*, ou um erro se o campo de autenticação não estiver de acordo com o *ciphertext* ou cabeçalho.
 - Nota: O cabeçalho pode ser introduzido, para garantir integridade e autenticidade de metadados, para os quais a confidencialidade não é necessária, mas a autenticidade é desejável.
- A Authenticated Encryption é mais utilizada com cifras simétricas de blocos, mas genericamente combina cifra com autenticação (MAC), desde que:
 - A cifra seja semanticamente segura, sob um ataque de *plaintext* escolhido.
 - A função MAC seja impossível de falsificar, sob um ataque de mensagem escolhida.

Authenticated Encryption

- Esquemas de Authenticated Encryption:
 - EtM (Encrypt-then-MAC)
 - Utilizado no IPsec, entre outros.

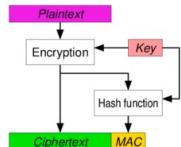


Authenticated Encryption

- Esquemas de Authenticated Encryption:

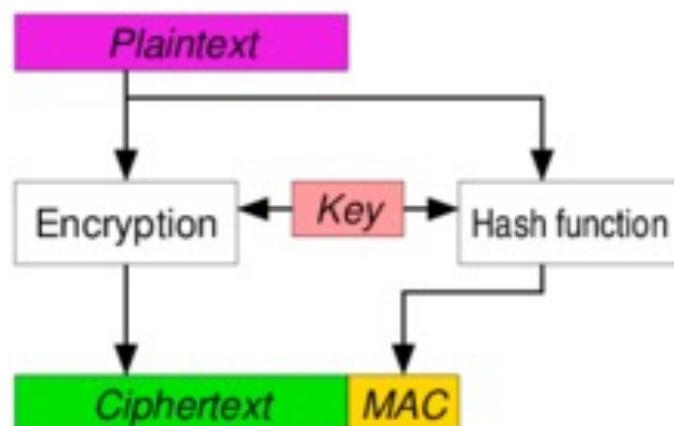
- EtM (Encrypt-then-MAC)

- Utilizado no IPsec, entre outros.



- E&M (Encrypt-and-MAC)

- Utilizado no ssh, entre outros.

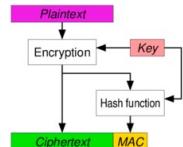


Authenticated Encryption

- Esquemas de Authenticated Encryption:

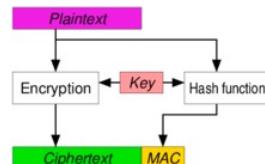
- EtM (Encrypt-then-MAC)

- Utilizado no IPsec, entre outros.



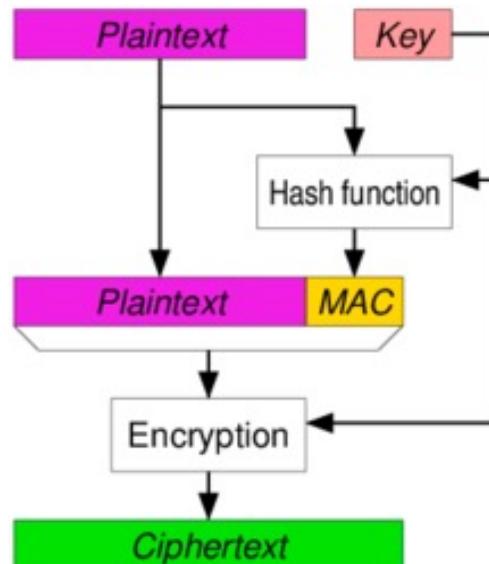
- E&M (Encrypt-and-MAC)

- Utilizado no ssh, entre outros.



- MtE (MAC-then-Encrypt)

- Utilizado no SSL/TLS, entre outros



Tópicos

- **Parte IX: Criptografia Aplicada**

- Gerador de número aleatórios / pseudo-aleatórios
- Partilha/Divisão de segredo (Secret Sharing/Splitting)
- Authenticated encryption
- **Algoritmos e tamanho de chaves - Legacy, Futuro**
- Assinaturas cegas (blind signatures)
- Criptografia homomórfica

Criptografia

- Sistema de cifragem é computacionalmente seguro se e só se verificar simultaneamente os seguintes critérios:
 - o custo de quebrar a cifra, excede o valor da informação cifrada;
 - o tempo necessário para quebrar a cifra excede o tempo de vida útil da informação.

Tamanho da chave	Nº de chaves alternativas	Uma cifragem por μs	10^6 cifragens por μs
32 bits	$2^{32} = 4,3 \times 10^9$	$2^{31} \mu\text{s} = 35,8$ minutos	2,15 ms
56 bits (DES)	$2^{56} = 7,2 \times 10^{16}$	$2^{55} \mu\text{s} = 1142$ anos	10,01 h
128 bits	$2^{128} = 3,4 \times 10^{38}$	$2^{127} \mu\text{s} = 5,4 \times 10^{24}$ anos	$5,4 \times 10^{18}$ anos
26 caracteres (permutação)	$26! = 4,03 \times 10^{26}$	$2 \times 10^{26} \mu\text{s} = 6,4 \times 10^{12}$ anos	$6,4 \times 10^6$ anos

[Tempo necessário para procura exaustiva no espaço de chaves]

Criptografia

- Como é que sabemos o que é computacionalmente seguro e durante quantos anos o será?
 - **Não sabemos!!**
 - Existem "previsões" sobre a segurança das várias técnicas criptográficas, feitas por organismos que têm muito bons cientistas, que são revistas regularmente.
- Documentos de referência:
 - *NIST Special Publication 800-57 Part 1 Revision 5 - Recommendation for Key Management: Part 1 – General* (Maio 2020) <https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-57pt1r5.pdf>
 - NIST Special Publication 800-131A Revision 2 - Transitioning the Use of Cryptographic Algorithms and Key Lengths (Mar 2019) <https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-131Ar2.pdf>
 - SOG-IS Crypto Evaluation Scheme Agreed Cryptographic Mechanisms (v. 1.2, Jan 2020)
<https://www.sogis.eu/documents/cc/crypto/SOGIS-Agreed-Cryptographic-Mechanisms-1.2.pdf>
 - ETSI TS 119 312 V1.4.2 - Electronic Signatures and Infrastructures (ESI); Cryptographic Suites (V1.4.2, 2022-02)
https://www.etsi.org/deliver/etsi_ts/119300_119399/119312/01.04.02_60/ts_119312v010402p.pdf
 - Algorithms, key size and parameters report, ENISA (Nov. 2014)
<https://www.enisa.europa.eu/publications/algorithms-key-size-and-parameters-report-2014/@@download/fullReport>

Criptografia

- NIST Special Publication 800-57 Part 1 Revision 5 - Recommendation for Key Management: Part 1 – General
 - Fornece informação sobre período de utilização de técnicas criptográficas, baseada na comparação da robustez de segurança dessas técnicas.

Table 2: Comparable security strengths of symmetric block cipher and asymmetric-key algorithms

Security Strength	Symmetric Key Algorithms	FFC (DSA, DH, MQV)	IFC* (RSA)	ECC* (ECDSA, EdDSA, DH, MQV)
≤ 80	2TDEA	$L = 1024$ $N = 160$	$k = 1024$	$f = 160-223$
112	3TDEA ⁶⁸	$L = 2048$ $N = 224$	$k = 2048$	$f = 224-255$
128	AES-128	$L = 3072$ $N = 256$	$k = 3072$	$f = 256-383$
192	AES-192	$L = 7680$ $N = 384$	$k = 7680$	$f = 384-511$
256	AES-256	$L = 15360$ $N = 512$	$k = 15360$	$f = 512+$

Utilização não permitida a partir de 2023 (SP 800-131A) 

 Utilização não permitida

* The security-strength estimates will be significantly affected when quantum computing becomes a practical consideration.

Criptografia

- NIST Special Publication 800-57 Part 1 Revision 5 - Recommendation for Key Management: Part 1 – General
 - Fornece informação sobre período de utilização de técnicas criptográficas, baseada na comparação da robustez de segurança dessas técnicas.

Table 3: Maximum security strengths for hash and hash-based functions

Security Strength	Digital Signatures and Other Applications Requiring Collision Resistance	HMAC, ⁷⁰ KMAC, ⁷¹ Key Derivation Functions, ⁷² Random Bit Generation ⁷³
≤ 80	SHA-1 ⁷⁴	
112	SHA-224, SHA-512/224, SHA3-224	
128	SHA-256, SHA-512/256, SHA3-256	SHA-1, KMAC128
192	SHA-384, SHA3-384	SHA-224, SHA-512/224, SHA3-224
≥ 256	SHA-512, SHA3-512	SHA-256, SHA-512/256, SHA-384, SHA-512, SHA3-256, SHA3-384, SHA3-512, KMAC256

Criptografia

- NIST Special Publication 800-57 Part 1 Revision 5 - Recommendation for Key Management: Part 1 – General
 - Fornece informação sobre período de utilização de técnicas criptográficas, baseada na comparação da robustez de segurança dessas técnicas.

Table 4: Security strength time frames

Processamento permitido em aplicações legacy

Security Strength		Through 2030	2031 and Beyond
< 112	Applying protection	Disallowed	
	Processing	Legacy use	
112	Applying protection	Acceptable	Disallowed
	Processing		Legacy use
128	Applying protection and processing information that is already protected	Acceptable	Acceptable
192		Acceptable	Acceptable
256		Acceptable	Acceptable

← Utilização não permitida

Nota:

- *Applying protection*, significa cifrar, assinar, gerar hash, ...
- *Processing*, significa decifrar, validar assinatura, validar hash, ...

Criptografia

- SOG-IS Crypto Evaluation Scheme Agreed Cryptographic Mechanisms
 - Fornece informação sobre período de utilização de técnicas criptográficas.
 - Referência a nível europeu (ETSI e ENISA recorrem a este documento para emitirem as suas recomendações)

Cifra de Blocos

Primitive	Parameters' sizes	R/L	Notes
AES [FIPS197, ISO18033-3]	k = 128 bits	R	
	k = 192 bits	R	
	k = 256 bits	R	
Triple-DES [FIPS46-3, ISO18033-3]	k = 168 bits	L	2-SmallBlocksize
	k = 112 bits	L [2024]	2-SmallBlocksize 3-TripleDES2key

R, significa Recomendado
L, significa Legacy



Criptografia

- SOG-IS Crypto Evaluation Scheme Agreed Cryptographic Mechanisms
 - Fornece informação sobre período de utilização de técnicas criptográficas.
 - Referência a nível europeu (ETSI e ENISA recorrem a este documento para emitirem as suas recomendações)

Esquemas de cifra simétrica

Scheme	R/L	Notes
CTR [SP800-38A, ISO10116]	R*	6-StreamMode
OFB [SP800-38A, ISO10116]	R*	6-StreamMode
CBC [SP800-38A, ISO10116]	R*	7-Padding
CBC-CS (CiphertextStealing) [SP800-38A-Addendum]	R*	
CFB [SP800-38A, ISO10116]	R*	7-Padding

Esquemas de cifra simétrica específica para discos

Scheme	R/L	Notes
XTS [SP800-38E]	R	9-UniqueTweak, 10-AddressTweak
CBC-ESSIV	L	11-UniqueSectorNumber, 12-AddressSectorNumber, 13-CBCMalleability

R, significa Recomendado

L, significa *Legacy*



Criptografia

- SOG-IS Crypto Evaluation Scheme Agreed Cryptographic Mechanisms
 - Fornece informação sobre período de utilização de técnicas criptográficas.
 - Referência a nível europeu (ETSI e ENISA recorrem a este documento para emitirem as suas recomendações)

Funções de Hash

Primitive	Parameters' sizes (hash length h)	R/L	Notes
SHA-2 [FIPS180-4, ISO10118-3]	$h = 256$ bits (SHA-256)	R	
	$h = 384$ bits (SHA-384)	R	
	$h = 512$ bits (SHA-512)	R	
	$h = 256$ (SHA-512/h)	R	
SHA-3 [FIPS202]	$h = 256$ bits	R	
	$h = 384$ bits	R	
	$h = 512$ bits	R	
SHA-2 [FIPS180-4, ISO10118-3]	$h = 224$ bits (SHA-224)	L [2025]	
	$h = 224$ bits (SHA-512/224)	L [2025]	

R, significa Recomendado

L, significa Legacy



Criptografia

- SOG-IS Crypto Evaluation Scheme Agreed Cryptographic Mechanisms
 - Fornece informação sobre período de utilização de técnicas criptográficas.
 - Referência a nível europeu (ETSI e ENISA recorrem a este documento para emitirem as suas recomendações)

MAC/HMAC

Scheme	R/L	Notes
CMAC [SP800-38B, ISO9797-1]	R	
CBC-MAC [ISO9797-1, Algorithm 1, Padding 2]	R	16-FixedInputLength

Scheme	R/L	Notes
GMAC [SP800-38D]	R	20-GMAC-GCMNonce 21-GMAC-GCMOptions 23-GMAC-GCM-Bounds

Scheme	Key size	R/L	Notes
HMAC [RFC2104, ISO9797-2]	$k \geq 125$	R	
	$k \geq 100$	L	
HMAC-SHA-1 [RFC2104, ISO9797-2, FIPS180-4]	$k \geq 100$	L	17-HMAC-SHA-1

R, significa Recomendado
 L, significa Legacy



Criptografia

- SOG-IS Crypto Evaluation Scheme Agreed Cryptographic Mechanisms
 - Fornece informação sobre período de utilização de técnicas criptográficas.
 - Referência a nível europeu (ETSI e ENISA recorrem a este documento para emitirem as suas recomendações)

Key Derivation Functions

Scheme	R/L	Notes
NIST SP800-56 ABC [SP800-56A, SP800-56B, SP800-56C]	R	
ANSI-X9.63-KDF [ANSIX9.63]	R	
PBKDF2 [RFC2898]	R	24-PBKDF2-PRF

Password Hashing Mechanisms

Scheme	R/L	Notes
PBKDF2 [RFC2898]	R	26-Salt, 25-NumberOfIterations

R, significa Recomendado
L, significa Legacy



Criptografia

- SOG-IS Crypto Evaluation Scheme Agreed Cryptographic Mechanisms
 - Fornece informação sobre período de utilização de técnicas criptográficas.
 - Referência a nível europeu (ETSI e ENISA recorrem a este documento para emitirem as suas recomendações)

Chaves assimétricas – RSA

Primitive	Parameters' sizes	R/L	Notes
RSA	$n \geq 3000, \log_2(e) > 16$	R	
	$n \geq 1900, \log_2(e) > 16$	L [2025]	27-LegacyRSA

Chaves assimétricas – Curvas elípticas

Curve Family	Curve	R/L	Notes
Brainpool [RFC5639]	BrainpoolP256r1	R	
	BrainpoolP384r1	R	
	BrainpoolP512r1	R	
NIST [FIPS186-4, Appendix D.1.2]	NIST P-256	R	
	NIST P-384	R	33-SpecialP
	NIST P-521	R	
FR [JORG]	FRP256v1	R	

R, significa Recomendado
 L, significa Legacy



Criptografia

- SOG-IS Crypto Evaluation Scheme Agreed Cryptographic Mechanisms
 - Fornece informação sobre período de utilização de técnicas criptográficas.
 - Referência a nível europeu (ETSI e ENISA recorrem a este documento para emitirem as suas recomendações)

Esquemas de cifra assimétrica

Primitive	Scheme	R/L	Notes
RSA	OAEP (PKCS#1v2.1) [RFC3447, PKCS1]	R	36-OAEP-PaddingAttack
RSA	PKCS#1v1.5 [RFC3447, PKCS1]	L	35-PaddingAttack

Esquemas de assinatura digital

Primitive	Scheme	R/L	Notes
RSA	PSS (PKCS#1v2.1) [RFC3447, PKCS1, ISO9796-2]	R	
FF-DLOG	KCDSA [ISO14888-3]	R	
	Schnorr [ISO14888-3]	R	40-DSARandom
	DSA [FIPS186-4, ISO14888-3]	R	
EC-DLOG	EC-KCDSA [ISO14888-3]	R	
	EC-DSA [FIPS186-4, ISO14888-3]	R	40-DSARandom
	EC-GDSA [TR-03111]	R	
	EC-Schnorr [ISO14888-3]	R	
	PKCS#1v1.5 [RFC3447, PKCS1, ISO9796-2]	L	39-PKCSFormatCheck



Criptografia

- SOG-IS Crypto Evaluation Scheme Agreed Cryptographic Mechanisms
 - Fornece informação sobre período de utilização de técnicas criptográficas.
 - Referência a nível europeu (ETSI e ENISA recorrem a este documento para emitirem as suas recomendações)

Esquemas para estabelecimento/troca de chaves

Primitive	Scheme	R/L	Notes
FF-DLOG	DH [SP800-56A, ISO11770-3]	R	42-DHAuth, 43-DHSubgroupAttacks
	DLIES-KEM [ISO18033-2]	R	
EC-DLOG	EC-DH [SP800-56A, ISO11770-3]	R	42-DHAuth, 43-DHSubgroupAttacks
	ECIES-KEM [ISO18033-2]	R	

Gerador de números aleatórios

Scheme	R/L	Notes
HMAC-DRBG [SP800-90A, ISO18031]	R	
Hash-DRBG [SP800-90A, ISO18031]	R	
CTR-DRBG [SP800-90A, ISO18031]	R	

R, significa Recomendado
 L, significa Legacy



Criptografia

- SOG-IS Crypto Evaluation Scheme Agreed Cryptographic Mechanisms
 - Fornece informação sobre período de utilização de técnicas criptográficas.
 - Referência a nível europeu (ETSI e ENISA recorrem a este documento para emitirem as suas recomendações)

Protocolos criptográficos – TLS

TLS protocol version	R/L
TLSv1.3 [RFC8446]	R
TLSv1.2 [RFC5246]	R

TLS Cipher suites

TLS code	Cipher Suite	R/L	Notes
TLS v1.3 Cipher Suite			
0x1302	TLS_AES_256_GCM_SHA384	R	
0x1301	TLS_AES_128_GCM_SHA256	R	
0x1304	TLS_AES_128_CCM_SHA256	R	
TLS v1.2 Cipher Suite			
0xC02C	TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384	R	
0xC02B	TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256	R	
0xCOAD	TLS_ECDHE_ECDSA_WITH_AES_256_CCM	R	
0xCOAC	TLS_ECDHE_ECDSA_WITH_AES_128_CCM	R	
0xC024	TLS_ECDHE_ECDSA_WITH_AES_256_CBC_SHA384	L[2025]	45-TLSEncryptThenMAC
0xC023	TLS_ECDHE_ECDSA_WITH_AES_128_CBC_SHA256	L[2025]	45-TLSEncryptThenMAC
0xC028	TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA384	L[2025]	45-TLSEncryptThenMAC
0xC027	TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA256	L[2025]	45-TLSEncryptThenMAC
0x009F	TLS_DHE_RSA_WITH_AES_256_GCM_SHA384	L	46-TLSDHE



Algoritmos e Tamanhos de chaves

- Tamanho de chaves (<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	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

Algoritmos e Tamanhos de chaves

- Tamanho de chaves (<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 Key	Elliptic Curve Group	Hash (A)	Hash (B)
(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
2016 - 2030 & beyond	192	AES-192	7680	384	7680	384	SHA-384 SHA3-384
2016 - 2030 & beyond	256	AES-256	15360	512	15360	512	SHA-512 SHA3-512 SHA-256 SHA-512/256 SHA3-256 SHA-384 SHA-512 SHA3-512

Algoritmos e Tamanhos de chaves

- Tamanho de chaves (<https://www.keylength.com>)
 - [NIST Recommendation](#), 2020

TDEA (Triple Data Encryption Algorithm) and AES are specified in [10].

Hash (A): Digital signatures and other applications requiring collision resistance.

Hash (B): HMAC, KMAC, key derivation functions and random bit generation.

Date	Security Strength	Symmetric Algorithms	Factoring Modulus	Discrete Logarithm Key	Group	Elliptic Curve	Hash (A)	Hash (B)
Legacy ⁽¹⁾	80	2TDEA	1024	160	1024	160	SHA-1 ⁽²⁾	
2019 - 2030	112	(3TDEA) ⁽³⁾ AES-128	2048	224	2048	224	SHA-224 SHA-512/224 SHA3-224	
2019 - 2030 & beyond	128	AES-128	3072	256	3072	256	SHA-256 SHA-512/256 SHA3-256	SHA-1 KMAC128
2019 - 2030 & beyond	192	AES-192	7680	384	7680	384	SHA-384 SHA3-384	SHA-224 SHA-512/224 SHA3-224
2019 - 2030 & beyond	256	AES-256	15360	512	15360	512	SHA-512 SHA3-512	SHA-256 SHA-512/256 SHA-384 SHA-512 SHA3-256 SHA3-384 SHA3-512 KMAC256



Algoritmos e Tamanhos de chaves

- Tamanho de chaves (<https://www.keylength.com>)
 - [ANSSI \(França\) Recommendation](#), 2014

Date	Symmetric	Factoring Modulus	Discrete Logarithm Key	Group	Elliptic Curve GF(p)	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

Algoritmos e Tamanhos de chaves

- Tamanho de chaves (<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](#)

Algoritmos e Tamanhos de chaves

- Tamanho de chaves (<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](#)

Algoritmos e Tamanhos de chaves

- Tamanho de chaves (<https://www.keylength.com>)
 - [BSI \(Alemanha\) Recommendation, 2015](#)

Date	Factoring Modulus	Discrete Logarithm Key	Group	Elliptic Curve	Hash
2014 - 2015	1976	224	2048	224	SHA-1(*) RIPEMD-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-DSS: [ISO/IEC 14888-3](#), [IEEE P1363](#), [FIPS 186-4](#) and [ANSI X9.62-2005](#)

EC-KDSR and EC-GDSA: [ISO/IEC 14888-3](#)

Nyberg-Rueppel (before end of 2020): [ISO/IEC 9796-3](#)



Algoritmos e Tamanhos de chaves

- Tamanho de chaves (<https://www.keylength.com>)
 - [BSI \(Alemanha\) Recommendation](#), 2017

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

Algoritmos e Tamanhos de chaves

- Tamanho de chaves (<https://www.keylength.com>)
 - [BSI \(Alemanha\) Recommendation, 2020](#)

Date	Symmetric	Factoring Modulus	Discrete Logarithm Key	Group	Elliptic Curve	Hash
2020 - 2022	128	2000	250	2000	250	SHA-256 SHA-512/256 SHA-384 SHA-512
2023 - 2026	128	3000	250	3000	250	SHA-256 SHA-512/256 SHA-384 SHA-512

Tópicos

- **Parte IX: Criptografia Aplicada**

- Gerador de número aleatórios / pseudo-aleatórios
- Partilha/Divisão de segredo (Secret Sharing/Splitting)
- Authenticated encryption
- Algoritmos e tamanho de chaves - Legacy, Futuro
- **Assinaturas cegas (blind signatures)**
- Criptografia homomórfica



Assinaturas cegas (*Blind signatures*)

- Introduzidas por David Chaum, em 1982, para utilização na área da moeda e pagamentos eletrónicos.
- A assinatura cega possibilita que uma entidade peça a uma terceira entidade para assinar digitalmente uma mensagem, sem lhe revelar o conteúdo da mensagem;
- Tipicamente é efetuada a seguinte analogia com o Mundo físico:
 - É colocado um papel com a mensagem dentro de um envelope;
 - É inserido no envelope, entre a frente do envelope e o papel com a mensagem, um papel químico;
 - O envelope é fechado e fornecido ao assinante;
 - O assinante assina a frente do envelope e devolve o envelope;
 - O dono da mensagem retira o papel de dentro do envelope, e o mesmo contém a assinatura do assinante.
 - Conclusão: O assinante não viu a mensagem, mas uma terceira parte que receba a mensagem pode validar a assinatura.

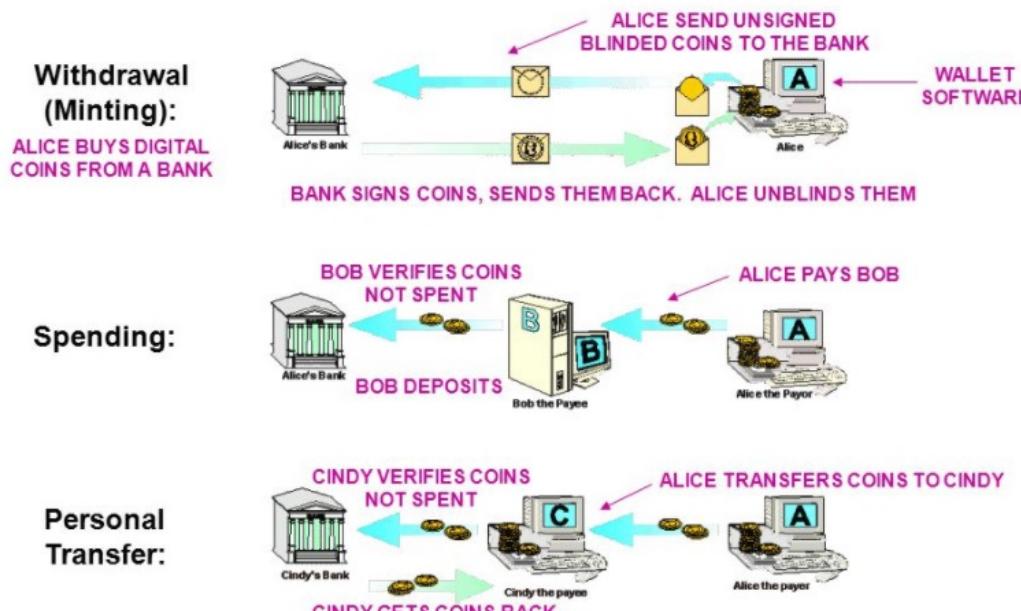


Assinaturas cegas (*Blind signatures*)

Em que situações necessito de assinaturas cegas?

- Voto eletrónico, em que os boletins de voto são assinados pelo sistema de voto antes de serem depositados na urna eletrónica, mas sem que seja revelado o conteúdo do voto.
- Em sistemas de dinheiro eletrónico, não rastreável.

eCash (Formerly DigiCash)



Assinaturas cegas (*Blind signatures*)

Como funciona?

- Suponha que Alice quer que Bob assine uma mensagem m , mas não quer que o Bob conheça o conteúdo da mensagem.
- Alice “cega/ofusca” a mensagem m , com um número aleatório b (fator de ofuscação): **$blind(m, b)$**
- Bob assina a mensagem com a sua chave privada d , tendo como resultado **$sign(blind(m, b), d)$**
- Alice desofusca a mensagem utilizando b , tendo como resultado **$unblind(sign(blind(m, b), d), b)$**
- As funções utilizadas estão desenhadas de tal modo que **$unblind(sign(blind(m, b), d), b) = sign(m, d)$** , i.e., a assinatura de m pelo Bob.
- Qualquer entidade/pessoa pode utilizar a chave pública de Bob para verificar se a assinatura é autêntica.



Assinaturas cegas (*Blind signatures*)

Assinaturas cegas RSA

- Seja e o expoente público RSA do assinante Bob, d o expoente secreto RSA do assinante Bob, e N o modulus RSA.
- Alice escolhe um valor aleatório r , tal que r é primo de N (i.e., $\gcd(r, N) = 1$).
- Alice utiliza $r^e \bmod N$ como fator de ofuscação
 - Note que como r é um valor aleatório, $r^e \bmod N$ também é aleatório.
- Seja m a mensagem que Alice quer que Bob assine, sem conhecer o seu conteúdo.

Assinaturas cegas (*Blind signatures*)

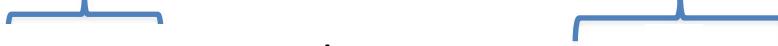
Assinaturas cegas RSA

- Alice ofusca a mensagem m com o fator de ofuscação

$$m' \equiv m r^e \pmod{N}$$
 - Note que como $r^e \pmod{N}$ é aleatório, m' não revela nenhuma informação de m
- Alice envia m' ao assinante Bob.
- O assinante utiliza o expoente secreto d para calcular a assinatura cega s'

$$s' \equiv (m')^d \pmod{N}$$
- Bob devolve s' a Alice.
- Alice remove o fator de ofuscação (utilizando r^{-1}), obtendo s , que é a assinatura de m por Bob

$$\begin{aligned}
 s' r^{-1} \pmod{N} &\equiv (m')^d r^{-1} \pmod{N} \equiv (m r^e)^d r^{-1} \pmod{N} \equiv \\
 &\equiv m^d r^{ed} r^{-1} \pmod{N} \equiv m^d r r^{-1} \pmod{N} \equiv m^d \pmod{N}
 \end{aligned}$$



Note que as chaves RSA satisfazem $r^{ed} \equiv r \pmod{N}$

Assinaturas cegas (*Blind signatures*)

Assinaturas cegas RSA

- É possível ser atacada.
- O atacante fornece ao Bob a versão cega da mensagem m' ofuscada, i.e., m'' .

$$\begin{aligned} m'' &= m'r^e \pmod{n} \\ &= (m^e \pmod{n}) \cdot r^e \pmod{n} \\ &= (mr)^e \pmod{n} \end{aligned}$$
- Quando o atacante desofusca a assinatura cega de m'' , é possível obter facilmente a mensagem inicial.

$$m = s' \cdot r^{-1} \pmod{n}, \text{ já que}$$

$$\begin{aligned} s' &= m''^d \pmod{n} \\ &= ((mr)^e \pmod{n})^d \pmod{n} \\ &= (mr)^{ed} \pmod{n} \\ &= m \cdot r \pmod{n}, \text{ since } ed \equiv 1 \pmod{\phi(n)} \end{aligned}$$

- Solução: Em vez de assinar a mensagem, assinar hash da mensagem.



Assinaturas cegas (*Blind signatures*)

Assinaturas cegas baseadas no *Elliptic Curve Discrete Logarithm Problem* (ECDLP)

- Sistema eficiente de assinatura cega, baseado na criptografia de curvas elípticas (ECC), descrito em <http://www.ijimt.org/papers/556-IT302.pdf>.
- Tem as seguintes fases:
 - Inicialização;
 - Ofuscação;
 - Assinatura;
 - Desofuscação;
 - Verificação.
- Tem três participantes:
 - Requerente (Alice),
 - Assinante (Bob),
 - Verificador.



Assinaturas cegas (*Blind signatures*)

Assinaturas cegas baseadas no ECDLP

- Parâmetros da curva elíptica sobre o Corpo finito F_p são definidos da seguinte forma:
 - $T = (p, F_p, a, b, G, n, h)$, em que:
 - p é um inteiro,
 - $a, b \in F_p$ especificam a curva elíptica $E(F_p)$ definida por $y^2 = x^3 + ax + b \text{ (mod } p\text{)},$
 - $G = (x_G, y_G)$ é um ponto base de $E(F_p)$,
 - n é um número primo que define a ordem de G (i.e., número de pontos do subgrupo gerado pelo ponto base),
 - h é um inteiro que define o cofator, $h = \#E(F_p)/n$ (i.e., número de pontos da curva dividido pelo número de pontos do subgrupo gerado pelo ponto base)

Assinaturas cegas (*Blind signatures*)

Assinaturas cegas baseadas no ECDLP

<i>Requester</i>	<i>Signer</i>
 Blinding Phase <p>Calculates r' from the elliptic curve point R' Integers v (randomly selected in the range $(1, n-1)$) Compute $R = v^{-1}R' = (x_0, y_0)$ $r = x_0 \bmod n$ (blinding factor) $m' = H(m) r^{-1} r' v \pmod{n}$ H is the hash function with SHA-256 algorithm</p>	 Initialization Phase <p>Publish $E_p(a, b)$ Base Point $G = (x, y)$ Integer k (randomly selected in the range $(1, n-1)$) $R' = kG = (x_1, y_1)$ and</p>
$\xrightarrow{\hspace{1cm}}$ R' $\xrightarrow{\hspace{1cm}}$ m'	$\xleftarrow{\hspace{1cm}}$ $\xrightarrow{\hspace{1cm}}$ $r' - x_1 \pmod{n}$ if $(r' \neq 0)$, Else if choose another k and find r'
Unblinding	Signing Phase
$s' = sv^{-1}r'^{-1}r \pmod{n}$ $\xleftarrow{\hspace{1cm}}$ (s, m') <p>The unblinding operation is needed to obtain the digital signature (s', R) on message m.</p>	$\xleftarrow{\hspace{1cm}}$ $Private\ key = d$ $(d$ randomly selected in the range $(1, n-1))$ $Public\ key = Q = dG = (x_Q, y_Q)$ Check (k, m') in database? If yes, re-select k . Otherwise, compute $s = dm' + kr' \pmod{n}$
Verifying Phase	
Any party who has the elliptic domain parameter T of the Signer and the public key of the signer can verify the signature is genuine. $s' G^2 = QH(m) + Rr$	

Assinaturas cegas (*Blind signatures*)

Assinaturas cegas baseadas no ECDLP

- Inicialização

```
user@CSI:~/Aulas/Aula3/BlindSignature$ python initSigner-app.py
Output
Init components: 7cb3aebdeaa9723f21df988a3b09fe9475f1e2253011d4cbfda09d62baa1ffe4.b22ad9e44cb42256724e82078ce59
3281bddf440d88267987af3287406bbea30
pRDashComponents: 7cb3aebdeaa9723f21df988a3b09fe9475f1e2253011d4cbfda09d62baa1ffe4.9e7f7b8503308a7565e7650f1648
b61b0b45ad2227928e770520884311d8a87d
```

- Ofuscação

```
user@CSI:~/Aulas/Aula3/BlindSignature$ python generateBlindData-app.py
Input
Data: Vamos assinar esta mensagem sem o Bob a ver
pRDash components: 7cb3aebdeaa9723f21df988a3b09fe9475f1e2253011d4cbfda09d62baa1ffe4.9e7f7b8503308a7565e7650f1648
b61b0b45ad2227928e770520884311d8a87d
Output
Blind message: 79c597b9d890706b6f48fa6279ecd5fd93ccde333257578bf74d0aed4717e374
Blind components: 2934575c9dd4f8fb8c3595e6af21bbfaa82f8aa02276acce29ee6200ca56b699.ed59ff6e354944d0c8b7d9f93fd3
f287abc56222d92a49994d600a1ccf3744ab
pRComponents: ed59ff6e354944d0c8b7d9f93fd3f287abc56222d92a49994d600a1ccf3744ab.a6157cffafe5fbbb7a41f5cda2aeb858
2568d61a1374f3c2d9cd9ddb37deb611
```



Assinaturas cegas (*Blind signatures*)

Assinaturas cegas baseadas no ECDLP

- Assinatura

```
user@CSI:~/Aulas/Aula3/BlindSignature$ python generateBlindSignature-app.py key.pem
Input
Passphrase:
Blind message: 79c597b9d890706b6f48fa6279ecd5fd93ccde333257578bf74d0aed4717e374
Init components: 7cb3aebdeaa9723f21df988a3b09fe9475f1e2253011d4cbfda09d62baa1ffe4.b22ad9e44cb42256724e82078ce59
3281bddf440d88267987af3287406bbea30
Output
Blind signature: 608d64e75bd4b08733bbc03ed60ad977d063d21943de5875cce27d0d80e42a667a03f7cb347699164cf91e75b13bb
a47e226f0bd5fdbb3e762686df0cabef04
```

- Desofuscação

```
user@CSI:~/Aulas/Aula3/BlindSignature$ python unblindSignature-app.py
Input
Blind signature: 608d64e75bd4b08733bbc03ed60ad977d063d21943de5875cce27d0d80e42a667a03f7cb347699164cf91e75b13bb
a47e226f0bd5fdbb3e762686df0cabef04
Blind components: 2934575c9dd4f8fbbe3595e6af21bbfaa82f8aa02276acce29ee6200ca56b699.ed59ff6e354944d0c8b7d9f93fd3
f287abc56222d92a49994d600a1ccf3744ab
pRDash components: 7cb3aebdeaa9723f21df988a3b09fe9475f1e2253011d4cbfda09d62baa1ffe4.9e7f7b8503308a7565e7650f164
8b61b0b45ad2227928e770520884311d8a87d
Output
Signature: f5efcba14b35e3f6c04a19772fa6af3f95550733ce15e84c687f14adc82b927d
```

Assinaturas cegas (*Blind signatures*)

Assinaturas cegas baseadas no ECDLP

- Verificação

```
user@CSI:~/Aulas/Aula3/BlindSignature$ python verifySignature-app.py key.crt
Input
Original data: Vamos assinar esta mensagem sem o Bob a ver
Signature: f5efcba14b35e3f6c04a19772fa6af3f95550733ce15e84c687f14adc82b927d
Blind components: 2934575c9dd4f8fbcc3595e6af21bbfaa82f8aa02276acce29ee6200ca56b699.ed59ff6e354944d0c8b7d9f93fd3
f287abc56222d92a49994d600a1ccf3744ab
pR components: ed59ff6e354944d0c8b7d9f93fd3f287abc56222d92a49994d600a1ccf3744ab.a6157cffafe5fbbb7a41f5cda2aeb85
82568d61a1374f3c2d9cd9ddb37deb611
Output
Valid signature
```



Tópicos

- **Parte IX: Criptografia Aplicada**

- Gerador de número aleatórios / pseudo-aleatórios
- Partilha/Divisão de segredo (Secret Sharing/Splitting)
- Authenticated encryption
- Algoritmos e tamanho de chaves - Legacy, Futuro
- Assinaturas cegas (blind signatures)
- **Criptografia homomórfica**

Criptografia homomórfica

- Técnica criptográfica proposta em 1978 por Ronald Rivest, Adleman e Dertouzos (*On Data Banks and Privacy Homomorphisms*, 1978, <https://evervault.com/papers/rivest-adleman-dertouzos.pdf>), mas só em 2009 é que foram estabelecidas, por Craig Gentry, as bases teóricas para a construção de tal técnica (*A fully homomorphic encryption scheme*, 2009, <https://evervault.com/papers/gentry.pdf>).
- A criptografia homomórfica permite trabalhar com dados cifrados sem a necessidade de os decifrar previamente ou de dispor da chave com que foram cifrados.
- Deste modo, é possível reduzir o número de vezes que os dados devem ser decifrados,
 - Minimizando a possibilidade de exposição dos dados;
 - Controlo mais rigoroso sobre a disponibilidade dos dados, garantindo a sua integridade e confidencialidade;
 - Permitindo anonimizar o processamento de dados.

Criptografia homomórfica - Conceito

- Ideia base: homomorfismo matemático.

Considerando:

- dois Grupos (A, \cdot) e $(B, *)$ e
- uma função $F: A \rightarrow B$ que toma elementos do conjunto A e retorna elementos do conjunto B .

F é um homomorfismo se e somente se $F(x \cdot y) = F(x) * F(y)$, para qualquer par de elementos “ x ” e “ y ” pertencentes ao conjunto A .

- Em termos de técnica criptográfica, o conjunto A é o texto simples, o conjunto B é o texto cifrado, “ \cdot ” é uma operação que pode ser executadas sobre o texto simples, “ $*$ ” é uma operação que pode ser executado sobre o texto cifrado, e F é a função criptográfica. As operações “ \cdot ” e “ $*$ ” são (usualmente) adição, subtração e multiplicação.
- Note que a maioria das funções criptográficas que foram estudadas até agora, são concebidas para não terem propriedades de homomorfismo para que os dados cifrados não tenham uma estrutura relacionada com os dados originais.
 - No caso da criptografia homomórfica, embora seja segura, dependendo da função criptográfica utilizada e da sua implementação, podem existir ataques que explorem essa característica.

Criptografia homomórfica - Tipos

- Alguns tipos de criptografia homomórfica:
 - *Partially homomorphic encryption*
 - Quando a função criptográfica (F) é um homomorfismo e permite que uma operação possa ser efetuada um número infinito de vezes. Por exemplo no caso da adição, significaria que permitia adicionar dois *ciphertext* (o que seria equivalente a cifrar o resultado da soma dos dois correspondentes *plaintext*) um número infinito de vezes;
 - São funções criptográficas relativamente “fáceis” de desenhar;
 - Alguns algoritmos de cifra são parcialmente homomórficos (por acaso), como o algoritmo RSA que é homomórfico para a operação de multiplicação. Como é baseado na exponenciação ($c = m^k \text{ mod } n$), sabemos pelas regras da exponenciação que $m_1^k \times m_2^k = (m_1 \times m_2)^k$, sendo k a chave e m_1, m_2 o *plaintext*. Logo, multiplicar dois *ciphertext* cifrados com a mesma chave é equivalente ao produto dos dois *plaintext* elevado à potência da chave utilizada na cifra.

Criptografia homomórfica - Tipos

- Alguns tipos de criptografia homomórfica:
 - *Somewhat Homomorphic Encryption*
 - Quando a função criptográfica (F) é um homomorfismo e permite um número finito de qualquer operação (em vez de um número infinito de uma operação). Por exemplo, pode ser capaz de processar uma combinação de 8 adições e multiplicações, mas a nona operação (de qualquer tipo) produz um resultado inválido.
 - São funções criptográficas mais “complexas” de desenhar. É mais difícil desenhar uma função homomórfica que permita adição e multiplicação (mesmo para um número finito de operações) de *ciphertext*, do que uma que permita uma operação infinita de adição ou multiplicação de *ciphertext*.
 - *Fully Homomorphic Encryption*
 - Quando a função criptográfica (F) é um homomorfismo e permite um número infinito de qualquer operação, produzindo sempre resultados válidos.
 - Primeira função de *fully homomorphic encryption* criada em 2009 por Craig Gentry.
 - Algumas bibliotecas *open-source*:
 - Helib, IBM, <https://homenc.github.io/HElib/>
 - Microsoft SEAL, Microsoft, <https://github.com/Microsoft/SEAL>
 - PALISADE, DARPA-funded defense contractors and academics, <https://gitlab.com/palisade/palisade-release>.
 - *Framework open-source*:
 - E3 (*Encrypt-Everything-Everywhere*), <https://github.com/momalab/e3>



Criptografia homomórfica - Vantagens

- Utilização interessante para serviços na cloud que têm que processar dados sem os exporem (decifrarem), para cumprir requisitos do Regulamento Geral de Proteção de Dados (RGPD) ou outros.

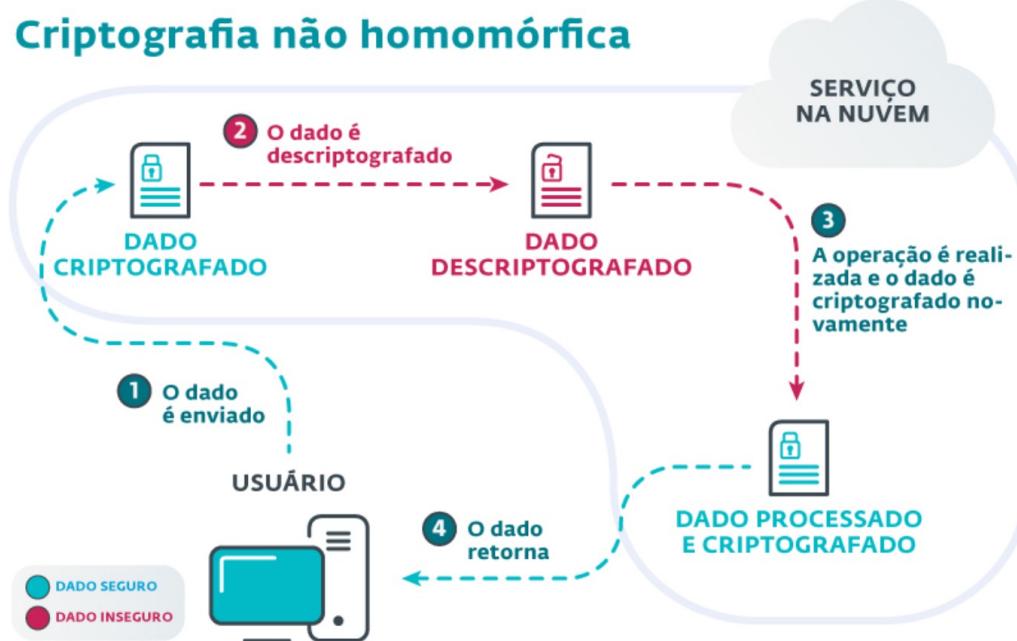


Imagen: <https://www.welivesecurity.com.br/2019/09/06/criptografia-homomorfica-um-esquema-de-criptografia-cada-vez-mais-usado/>

Criptografia homomórfica - Vantagens

- Utilização interessante para serviços na cloud que têm que processar dados sem os exporem (decifrarem), para cumprir requisitos do Regulamento Geral de Proteção de Dados (RGPD) ou outros.



Imagens: <https://www.welivesecurity.com.br/2019/09/06/criptografia-homomorfica-um-esquema-de-criptografia-cada-vez-mais-usado/>

Criptografia homomórfica - Exemplo

Exemplo utilizando a biblioteca phe em python da **criptografia homomórfica de Pallier** (*Partially homomorphic encryption* para a operação de adição).

(ver <https://coderzcolumn.com/tutorials/python/paillier-homomorphic-encryption-phe>)

```
import phe
from phe import paillier
# Gera o par de chaves
pub_key,priv_key = paillier.generate_paillier_keypair(n_length=3072)
# Cifra os dados, por exemplo antes de enviar para o serviço cloud
data1 = pub_key.encrypt(20)
data2 = pub_key.encrypt(32)
data3 = pub_key.encrypt(21)
# Efetua operação sobre os dados cifrados, por exemplo no serviço cloud
soma = data1 + data2 + data3
# Vê o que é a soma, por exemplo no serviço cloud
print(soma)
# Obtém o valor da soma, por exemplo depois de a descarregar do serviço cloud
print(priv_key.decrypt(soma))
```

