Mestrado em Engenharia Informática (MEI)
Mestrado Integrado em Engenharia Informática (MiEI)

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

Engenharia de Segurança

jose.miranda@devisefutures.com

Tópicos

- Criptografia – revisões dos conceitos básicos
- Criptografia Aplicada
  - Gerador de número aleatórios / pseudo-aleatórios
  - Partilha/Divisão de segredo (Secret Sharing/Splitting)
  - Authenticated encryption
  - Algoritmos e tamanho de chaves - Legacy, Futuro

Criptografia – Revisões dos conceitos básicos

- Para que necessitamos da criptografia?

Criptografia

- Criptografia Moderna
  - Não se centra apenas na confidencialidade
  - Integridade
    - Funções de Hash
    - MAC
    - Assinaturas digitais
  - Troca Justa
    - Assinatura de Contratos
  - Anonimicidade
    - Dinheiro Electrónico
    - Votação Electrónica
  - Não Repúdio
    - Criptografia assimétrica
  - Autenticidade
    - MAC
    - Assinatura Digital
  - ...

Criptografia

- Breve História da Criptografia
  - Há > 2000 anos: Cifras de substituição
    - Ex.: Cifra de César

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	

Criptografia

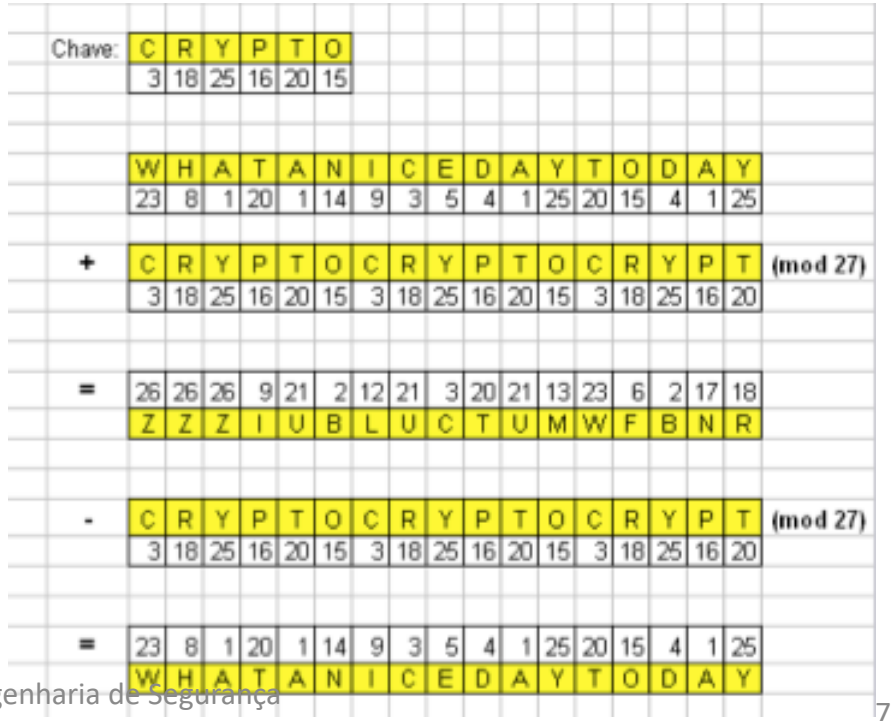
- Breve História da Criptografia
  - Há > 2000 anos: Cifras de substituição
  - Alguns séculos depois: Cifras de permutação

HELLOWORLD → LOHLERDLWO

$$\begin{pmatrix} 1 & 2 & 3 & 4 & 5 \\ 3 & 5 & 4 & 1 & 2 \end{pmatrix}$$

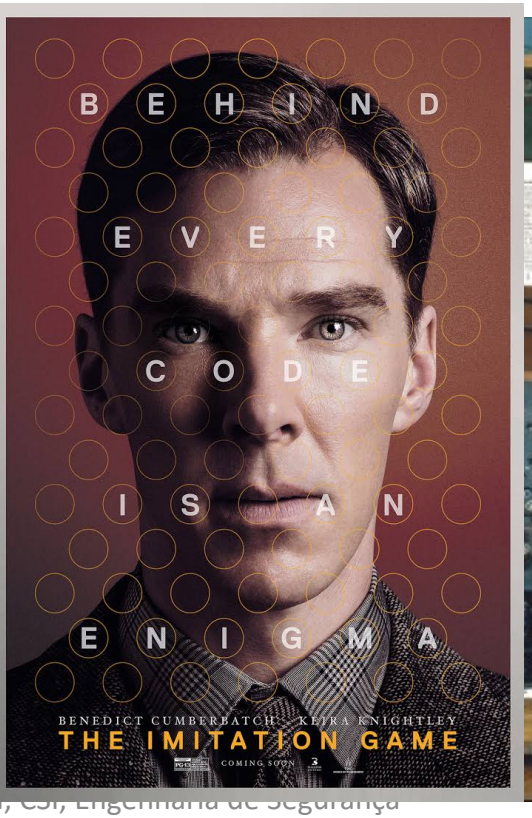
Criptografia

- Breve História da Criptografia
  - Há > 2000 anos: Cifras de substituição
  - Alguns séculos depois: Cifras de permutação
  - Renascença: Cifras Poli-alfabética



Criptografia

- Breve História da Criptografia
  - Há > 2000 anos: Cifras de substituição
  - Algu
  - Rena
  - 1844
    - E



Criptografia

- Breve História da Criptografia
  - Há > 2000 anos: Cifras de substituição
  - Alguns séculos depois: Cifras de permutação
  - Renascença: Cifras Poli-alfabética
  - 1844: Mecanização
  - 1976: Public Key Criptography (PKI)

Criptografia

- Sistemas criptográficos, são genericamente classificados segundo três dimensões independentes:
  - **Tipo de operação** utilizada para transformar *plaintext* em *ciphertext*. Todos os algoritmos de cifragem estão baseados em dois principios genéricos:
    - substituição - cada elemento no *plaintext* (bit, letra, grupo de bits ou letras) é mapeado noutro elemento
    - transposição - elementos do *plaintext* são reorganizados

    Requisito fundamental é que nenhuma informação seja perdida, i.e., que todas as operações sejam reversíveis.

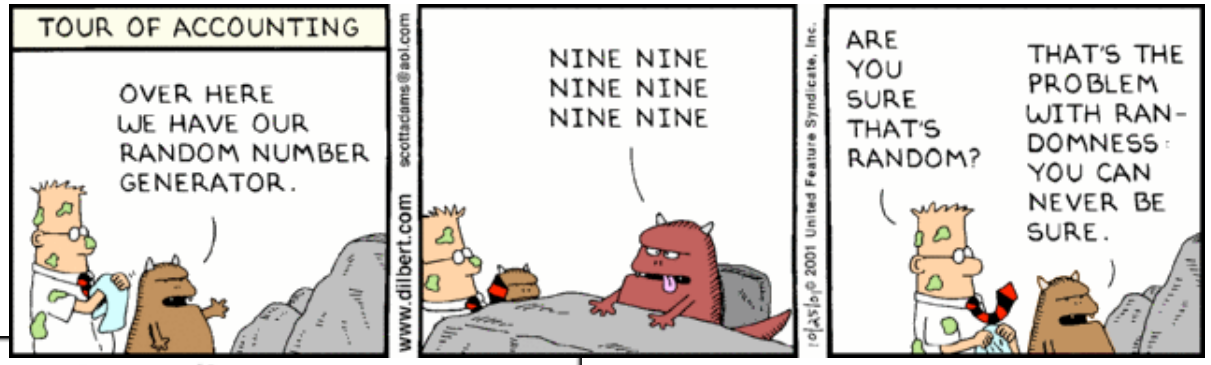
    A maior parte dos algoritmos de cifragem envolvem múltiplas fases de substituições e transposições.
  - **Número de chaves** usada:
    - simétrico, chave secreta ou cifragem convencional, se emissor e receptor usam a mesma chave
    - assimétrico ou chave pública, se emissor e recetor usam chaves distintas
  - **Modo** como o *plaintext* é **processado**:
    - cifra de bloco - processa o input, um bloco de elementos de cada vez, produzindo um bloco de output por cada bloco de input
    - cifra de fluxo (stream) - processa os elementos de input de uma forma continua, produzindo o output, à medida que passa o fluxo de input

Gerador de número aleatórios / pseudoaleatórios

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 fenómenos 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

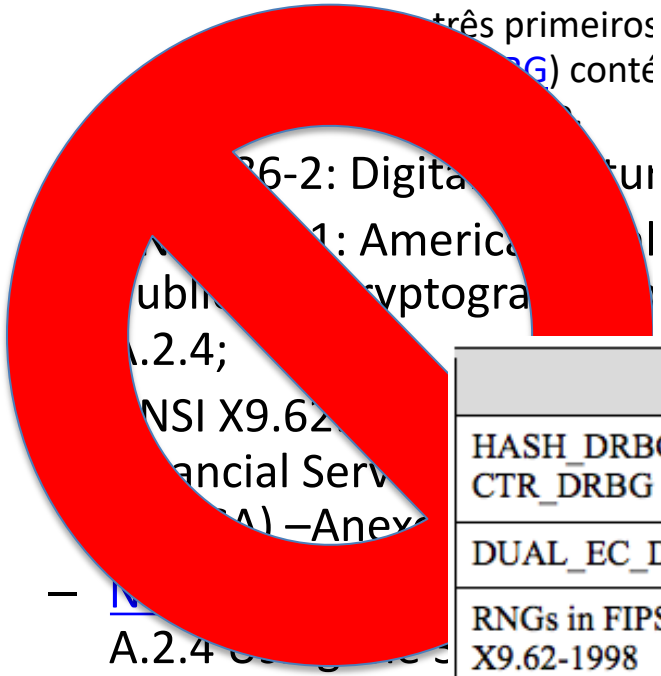
- **Geradores de números pseudoaleatórios criptograficamente seguros** são geradores de números pseudoaleatórios com **propriedades** adequadas para poderem ser utilizados em criptografia:
  - Utiliza entropia obtida de uma fonte de alta qualidade, como um gerador de números aleatórios em hardware ou 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);
  - N… três primeiros … G) conté… … 6-2: Digita… …tur… … 1: Americ… …l … ubli… …yptogra… … .2.4; … NSI X9.62… … ancial Serv… … A) –Anex…
  - N… A.2.4 …



Desde 1 de Janeiro de 2016, de acordo com o 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

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)

https://beacon.nist.gov/beacon/2.0/pulse/last

JSON Raw Data Headers

Save Copy Pretty Print


```
{
  "pulse" : {
    "uri" : "https://beacon.nist.gov/beacon/2.0/chain/1/pulse/759848",
    "version" : "Version 2.0",
    "cipherSuite" : 0,
    "period" : 60000,
    "certificateId" : "02288edbdc04bdeeb1c8140a41a1ad40ab2ea7ad27fdd316946f3ec5aff1a7129f1fb5078202d75d42c201878b06d79c45bf37adb55f83aa213200834792b1da",
    "chainIndex" : 1,
    "pulseIndex" : 759848,
    "timeStamp" : "2020-02-15T17:15:00.000Z",
    "localRandomValue" : "CB7D21C0C1E47D197AAF17B507BE1D8040462CF1E6211BD8A1422D33CAE0F0E782BA0A974A70EA31BFD0FB7800A64957585B378F375CDC5393918A335C52C4D3",
    "external" : {
      "sourceId" : "0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000",
      "statusCode" : 0,
      "value" : "0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000"
    },
    "listValues" : [ {
      "uri" : "https://beacon.nist.gov/beacon/2.0/chain/1/pulse/759847",
      "type" : "previous",
      "value" : "426CDD8FA8755FBDDA5BBF918B3987CE51F5D95A4E29ED0A4A3C14308BB5C14B0C2743A5FBB5C92408DBB9A862D39558DB627670D00BF4872FEF94ADDD6B0E26"
    }, {
      "uri" : "https://beacon.nist.gov/beacon/2.0/chain/1/pulse/759833",
      "type" : "hour",
      "value" : "ABF8C92DF1137B4903D92460DC186F9150EE4D6C673EB2907AD545EE79E5510EE54FF54FCB642EE4300BF6DEC635D78BF54F9840C6B46E0B8F24C305B96941D5"
    }, {
      "uri" : "https://beacon.nist.gov/beacon/2.0/chain/1/pulse/758828",
      "type" : "day",
      "value" : "89E7221F10C765175356B89014D3683A0380DB28ECD0AF7532F5D17B4E6607466C24C957FFD568932A02AAE88B6983F9E273D68AFA40AD259104D354DF80EB7F"
    }, {
      "uri" : "https://beacon.nist.gov/beacon/2.0/chain/1/pulse/738668",
      "type" : "month",
      "value" : "F9BC6B469CD47442BEFBFB3AFC5E8A99035D379FD42807ACA414A5BF504925342265AC9069FC00040B0602EBEDDAF0C3133DE94BE6711FDB609B3595440C6140"
    }, {
      "uri" : "https://beacon.nist.gov/beacon/2.0/chain/1/pulse/694662",
      "type" : "year",
      "value" : "CBC9AA97CDD5954218C585C89B061F356EF5F4158622C7CB38FBC317CA69C7ABE9E4379D4738B1076F7671C916C78AD0167A9ADB5A53E0CB20CC7F3D38736857"
    } ],
    "precommitmentValue" : "A7B521BBFD56FDDE5B88E40FF58CCA511557F724DC61101B634244F2B998636AE92430F1272F7E276DD7C4155059A6B1B87964C6EA70DC9DAC4A816760B65504",
    "statusCode" : 0,
    "signatureValue" :
"7EAFD096F9624B990BB7E298C674C5D9B0D3F67FD956DEDDCB60E05D889C38799A869F6FAF992D2C038D6767171A3110B0F786C69488E997FF6938A66DFEBA0E4513EEB639B1E2889A2621B4EDE0BAC4
    "outputValue" : "07DB5114B3287FE1FAB218F08458509C5E4292F0D7E4E62247D73C682B777F552D88C2BA7C06CA2A667427C383FE58C3696BF2D0EA2646F6B13130FD070EC10D"
  }
}
```

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
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
vhzSE0K1STKB1QhrY3zmPoGGcAs5zJs07VjcrtbnU5M7EXPyJpQfr607e3k0IVv/
8F5IFCRG0tkXDm720lNgnKszCYThtXYBUd0xsM4PVIyhGP/yct4G7f4AxCWbMgSA
/Tw58sJh3VrJ64QEKF1divB95xZ2dqVhdJ50kUrfe60j+JaRlujByhAJRn1jujEv
153yk89pJvy1qd3cBe2JYu47aIqWB8DmMz2BKieP176CgArnbh2RoBOrgfi33+4/
E5HoAk4+5weMNgxMkj4qaVy7TTh+fpupCsdyMLr5KalZRJf8275Vl2FAMyaX+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+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==
```

Geradores de números pseudoaleatórios criptograficamente seguros

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

}
```

Partilha/Divisão de segredo (Secret Sharing/Splitting)

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 XOR p_1 XOR p_2 XOR ... 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 XOR p_2 XOR ... 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: 00100101001010111101101000110110011111011100111010000110010000100100110010100000 10
Codigo 1: 00100011111110010001000000010110100010010000010100100100111001111111001101010110
Codigo 2: 10101101111011110010110110000011101000110010000101011011101000001111100110011000
Codigo 3: 00011011111100100011001110111011111000100000001010110000111100010110100111010001
Codigo 5: 11110011100111001001110100111000100111101001101110000111011110111010111100010000
```

```
[jepm@ProOne SecretSharing]$ php reconstroiSecret.php 10101101111011110010110110000011101000110010000101011011101000001111100110011000 00100101001010111101101000110110011111011100111010000110010000100100110010100000 10 11110011100111001001110100111000100111101001101110000111011110111010111100010000 00011011111100100011001110111011111000100000001010110000111100010110100111010001 10110100111010001 00100011111111001000100000001011010001001000000101001001001110011111110011010101 10
Segredo: CSI - DIUM
```

Nota: código php na directoria 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 $\{P_{m1}(P_{m2}(...(P_{mN}(S)))), Q_m, m1, m2, ... mN\}$,

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 $a_1, ..., a_{N-1}$, com $a_i < P$ e $a_0 = S$, construindo deste modo a função polinomial $f(x) = (a_0 + a_1 x + a_2 x^2 + a_3 x^3 + ... + a_{N-1} x^{N-1}) \bmod P$
- De seguida, calcule M pontos da função polinomial (i = 1, ... , 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:

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

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

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 $f(x) = 1234 + 166\,x + 94\,x^2 \bmod 1613$
- Calculamos 6 pontos/códigos (i, f(i) mod P):
  - $C_1 = (1, 1494)$, $C_2 = (2, 329)$, $C_3 = (3, 965)$, $C_4 = (4, 176)$, $C_5 = (5, 1188)$, $C_6 = (6, 775)$
- 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

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

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

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

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

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

$$\equiv 1234$$

Note que:

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

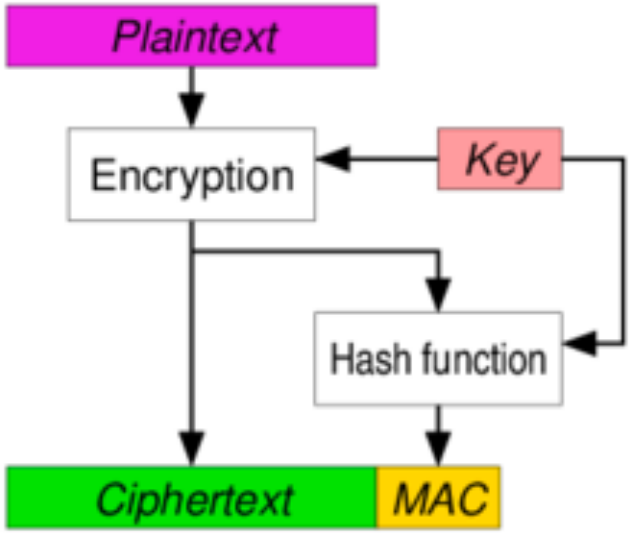
Authenticated Encryption

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.

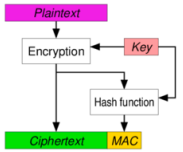
Authenticated Encryption

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

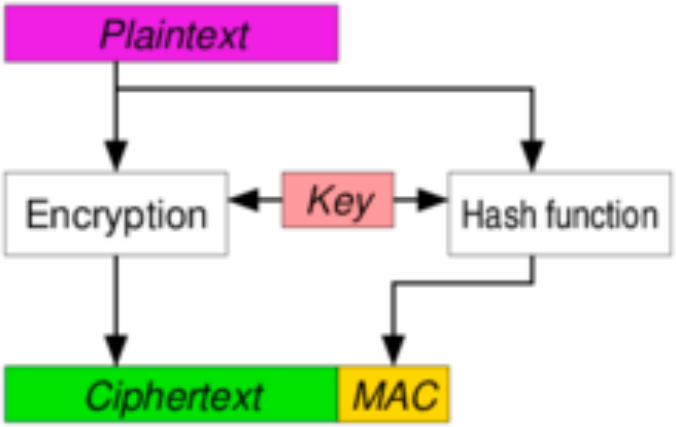


Authenticated Encryption

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

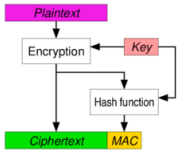


  - E&M (Encrypt-and-MAC)
    - Utilizado no ssh, entre outros.

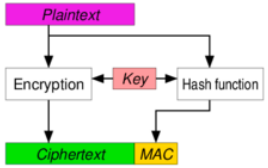


Authenticated Encryption

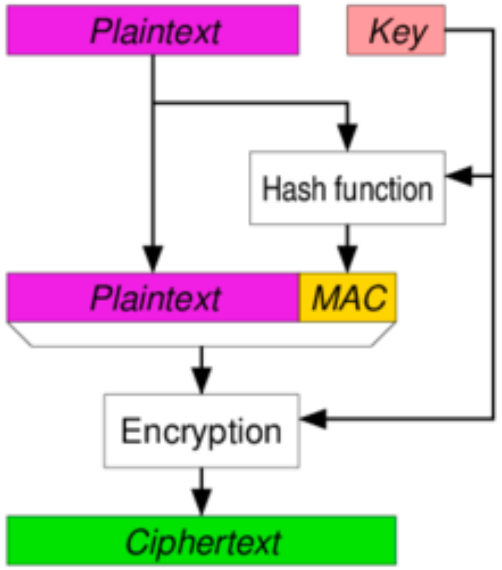
- Esquemas de Authenticated Encryption:
  - EtM (Encrypt-then-MAC)
    - Utilizado no IPsec, entre outros.



  - E&M (Encrypt-and-MAC)
    - Utilizado no ssh, entre outros.

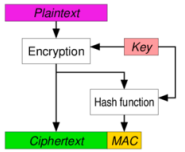


  - MtE (MAC-then-Encrypt)
    - Utilizado no SSL/TLS, entre outros

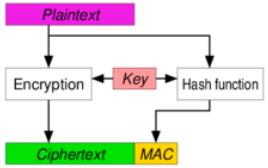


Authenticated Encryption

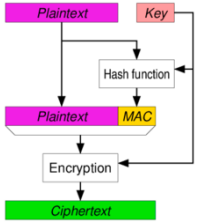
- Esquemas de Authenticated Encryption:
  - EtM (Encrypt-then-MAC)
    - Utilizado no IPsec, entre outros.



  - E&M (Encrypt-and-MAC)
    - Utilizado no ssh, entre outros.



  - MtE (MAC-then-Encrypt)
    - Utilizado no SSL/TLS, entre outros



- Adicionalmente a Authenticated Encryption pode introduzir segurança contra ataques de *ciphertext* escolhidos. Como?

Algoritmos e tamanho de chaves - Legacy, Futuro

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} 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 caracteres (permutação)	$26! = 4{,}03 \times 10^{26}$	2×10^{26} µs = $6{,}4 \times 10^{12}$ anos	$6{,}4 \times 10^6$ anos

[Tempo necessário para procura exaustiva no espaço de chaves]

Criptografia simétrica

- Algoritmos de Criptografia simétrica – Classificação 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.

Funções de hash criptográficas

- Algoritmos de Funções de hash criptográficas – Classificação 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.

Criptografia assimétrica (de chave pública)

- Algoritmos de Funções de hash criptográficas – Classificação 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.

Algoritmos e Tamanhos de chaves

- Tamanho de chaves
  - 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

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	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

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	Discrete Logarithm Group	Elliptic Curve	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	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

Algoritmos e Tamanhos de chaves

- Tamanho de chaves (https://www.keylength.com)
  - ANSSI (França) 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

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	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.

Algoritmos e Tamanhos de chaves

- Tamanho de chaves (https://www.keylength.com)
  - BSI (Alemanha) 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