THE DEVELOPER'S CONFERENCE

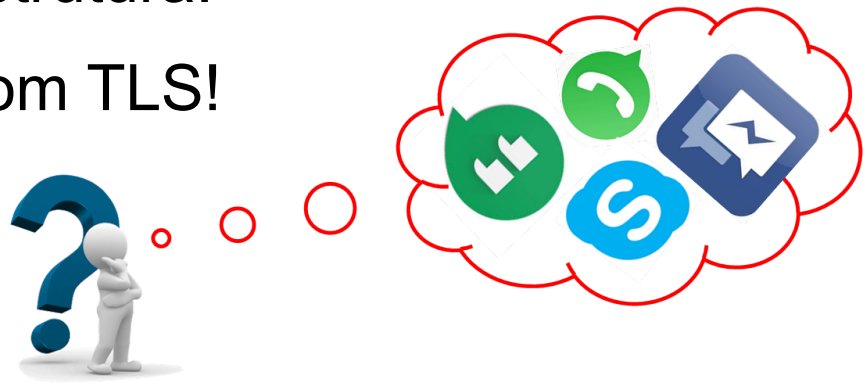
Como *não* usar uma API criptográfica

Dr. Alexandre Braga
PMP, CISSP, CSSLP

Você sabia?

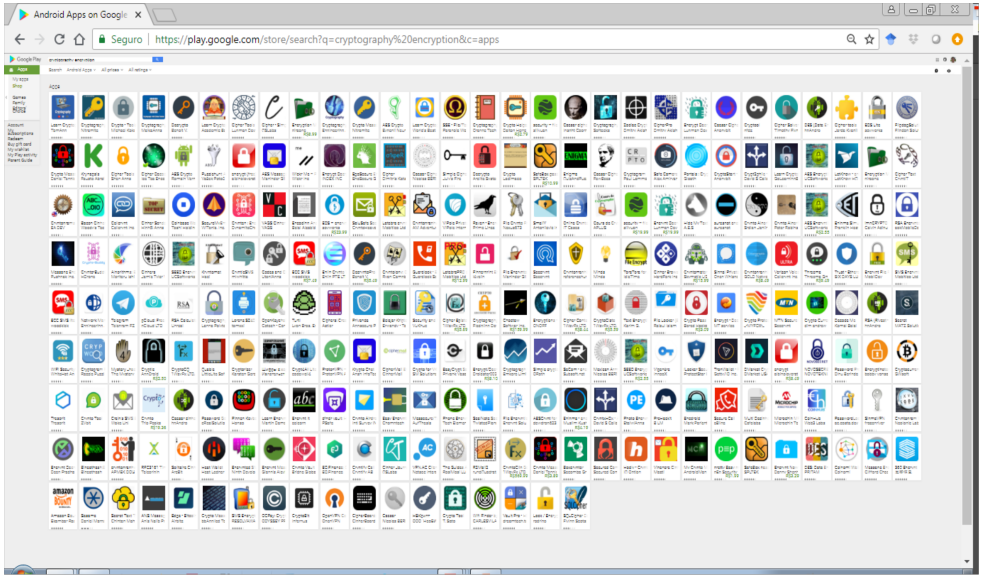
THE DEVELOPER'S CONFERENCE

- Os softwares criptográficos mais populares são os aplicativos com **criptografia na lógica de negócios**
- O uso correto da criptografia na aplicação
  - Não é problema de infraestrutura!
  - Não é resolvido apenas com TLS!
  - Não é só https!



Você tem, usa ou faz

- Software criptográfico é cada vez mais comum

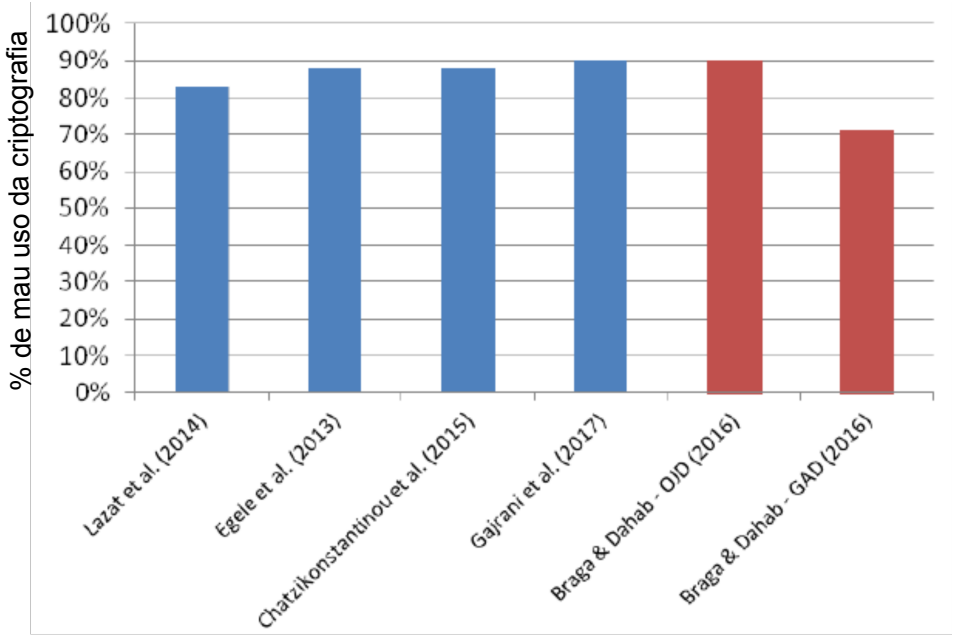


Fonte:
https://play.google.com/store/search?q=cryptography&c=apps

Mau uso frequente

THE DEVELOPER'S CONFERENCE

- O mau uso da criptografia é muito frequente
  - Lojas de aplicativos
  - Fóruns de programação
  - Etc..

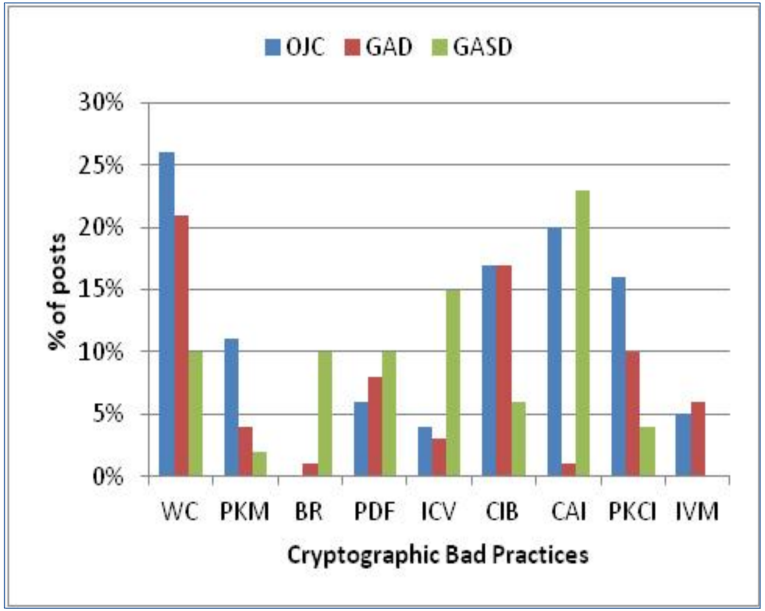


Tipos de maus usos

THE DEVELOPER'S CONFERENCE

- OJC → Oracle Java Cryptography
- GAD, GASD → Google Android Developers / Security Discussions

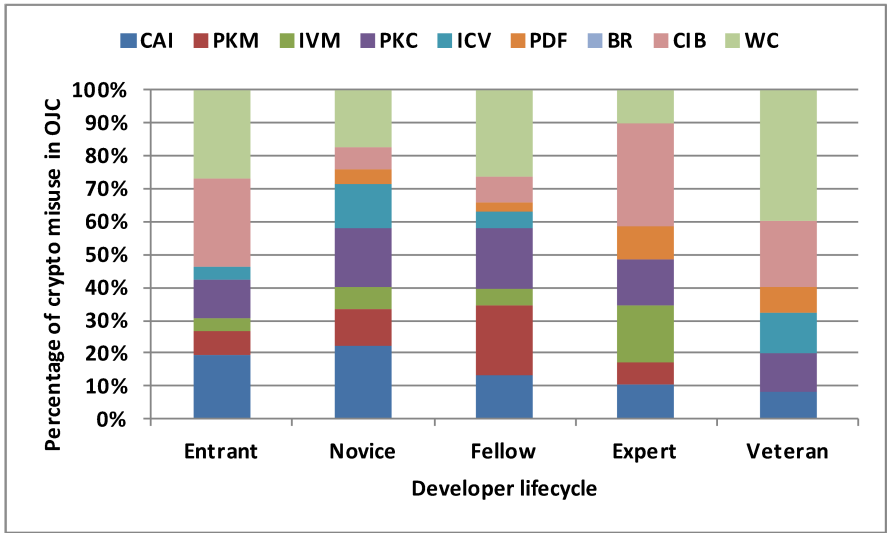
Categories of Cryptography Misuse	OJC	GAD	GASD
Weak Cryptography (WC)	26%	21%	10%
Poor Key Management (PKM)	11%	4%	2%
Crypto Architecture and Infra (CAI)	20%	1%	23%
Bad Randomness (BR)	0%	1%	10%
Program Design Flaws (PDF)	6%	8%	10%
Improper Certificate Validation (ICV)	4%	3%	15%
Public-Key Cryptography (PKC)	16%	10%	4%
Coding and Implementation Bugs (CIB)	17%	17%	6%
IV/Nonce Management (IVM)	5%	6%	0%
Env. and Platform Specific Issues (EPSI)	60%	32%	48%



O que vimos em OJC

THE DEVELOPER'S CONFERENCE

- Mau uso da criptografia *versus* maturidade do programador
  - Novice: CAI, PKC e WC
  - Novice e Fellow: PKM e PKC
  - WC e CIB sempre ocorrem

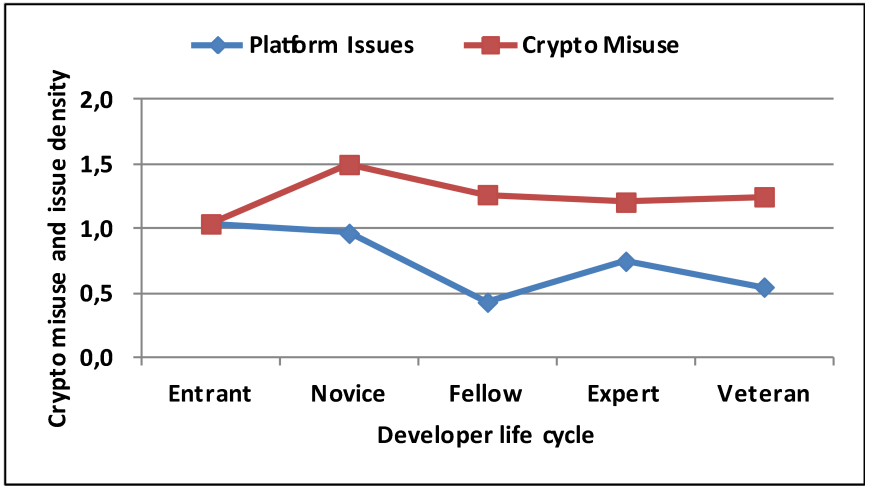


- Ciclo de vida otimista
- Entrant → Novice → Fellow → Expert → Veteran

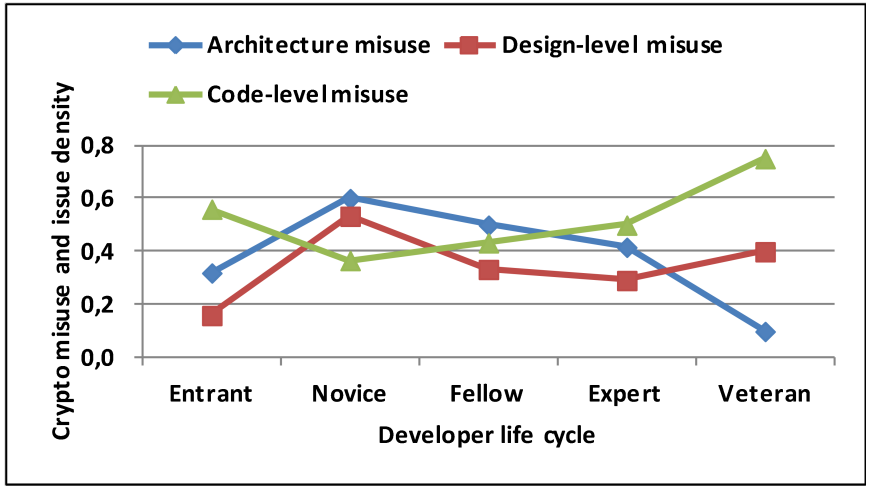
Densidade de mau uso

THE DEVELOPER'S CONFERENCE

- Densidade de mau uso de criptografia em OJC
- Densidade = # maus usos / # de posts
- Densidade é estável mesmo com diminuição de problemas da plataforma



- Densidade de maus usos em código (WC, BR, CIB) aumenta com o tempo
- Densidade de maus usos em projeto (PDF, PKC, ICV) e arquitetura (CAI, PKM, ICV) diminuem com o tempo



O Arroz com feijão ...

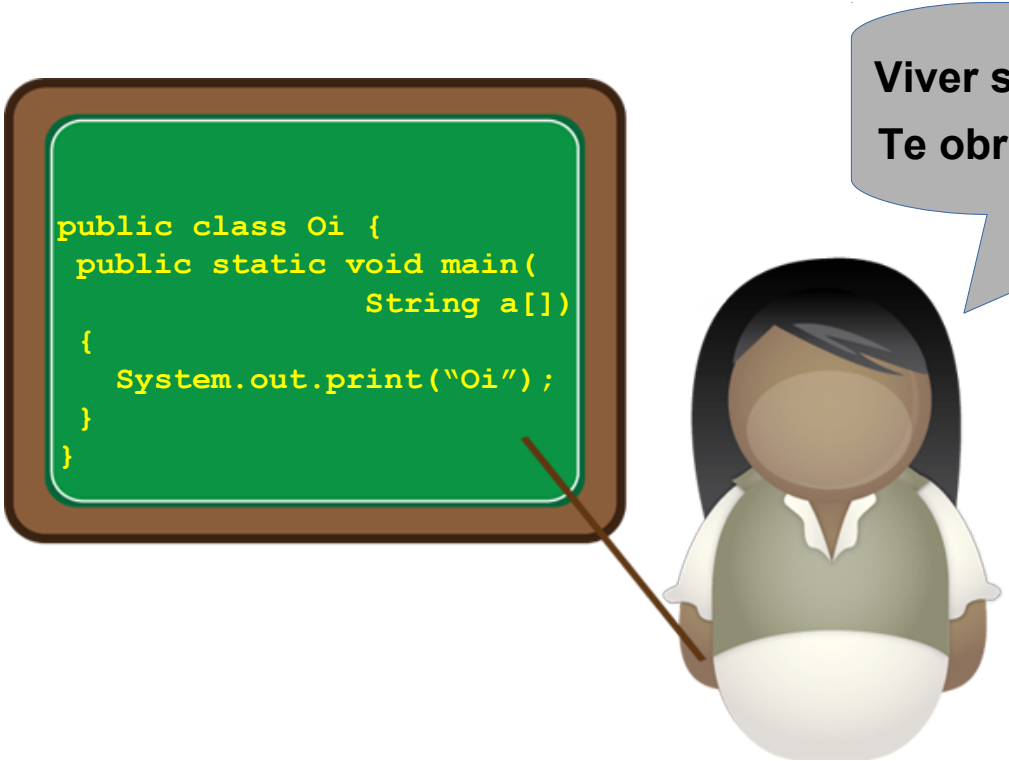
THE DEVELOPER'S CONFERENCE

- Evitar a criptografia fraca ou obsoleta
  - DES, 3DES, RC4, MD5, MD2, SHA-1

https://commons.wikimedia.org/wiki/File:Rice_and_beans,_Hotel_in_Itatiaia.jpeg

- Evitar tamanhos de chave inseguros
  - < 2048 para RSA
  - < 2048 para DH
  - < 128 para encriptação simétrica
  - < 256 para *hashes (tamanho da saída)*
  - *< 256 para ECC*

AES	RSA	DH (k)	DH (p)	ECC	hash
80	1024	160	1024	160–163	SHA1 (160)
112	2048	224	2048	224–233	SHA-224/512 SHA3-224
128	3072	256	3072	256–283	SHA-256/512 SHA3-256
192	7680	384	7680	384–409	SHA-384 SHA3-384
256	15360	512	15360	512–571	SHA-512 SHA3-512


public class Oi {
public static void main(
String a[])
{
System.out.print("Oi");
}
}
Viver sem ler código fonte é perigoso.
Te obriga a acreditar no que te dizem.

O pior caso que eu já vi (1)

THE DEVELOPER'S CONFERENCE

Em Java: combina 3 maus usos …

1) Criptografia fraca
2) Bug de *char encoding*
3) Projeto inseguro

```
01  byte[] pt = ("OJC worst case..").getBytes();
02  KeyGenerator g=KeyGenerator.getInstance("AES");
03  g.init(128); Key k = g.generateKey();
04  // encryption
05  Cipher e = Cipher.getInstance("AES");
06  e.init(Cipher.ENCRYPT_MODE, k);
07  byte[] ct = e.doFinal(pt);
08  String s = new String(ct);
09  // decryption in other machine or platform
10  Cipher d = Cipher.getInstance("AES");
11  d.init(Cipher.DECRYPT_MODE, k);
12  byte[] pt2 = d.doFinal(s.getBytes());
```

- Texto claro obtido de Strings
- Weak crypto usada por "acidente"
- Modo default do AES é ECB
- Criptograma salvo em String
- Dependente do modo de Codificação
- Com resultado inesperado

O pior caso que eu já vi (2)

THE DEVELOPER'S CONFERENCE

Em Android: combina 3 maus usos …

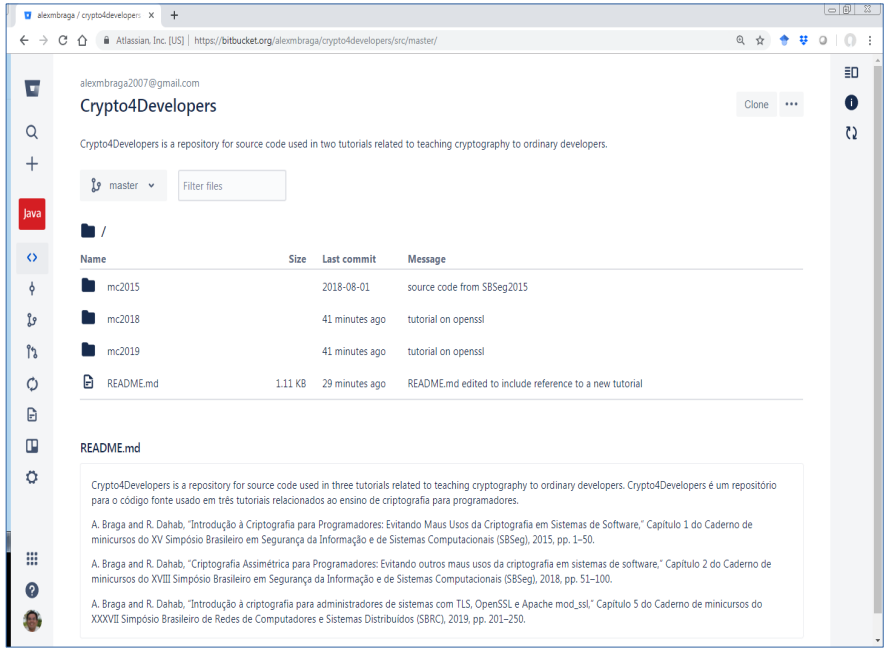
1) Criptografia fraca
2) Bug de *char encoding*
3) Defeitos na plataforma

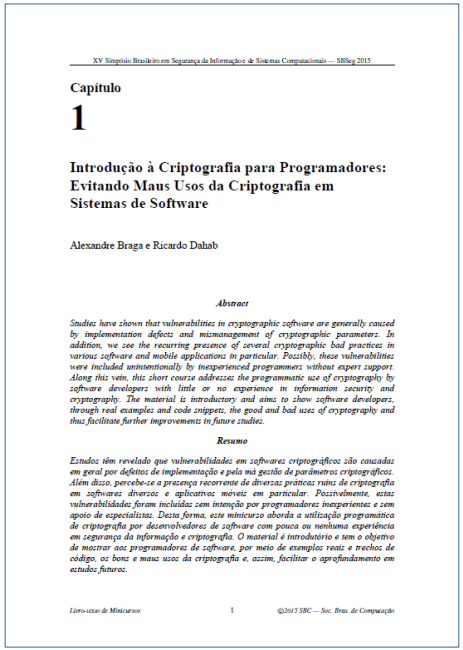
```
01  // weak hash of user's password
02  md = MessageDigest.getInstance("MD5");
03  byte[] hash = md.digest(password.getBytes());
04  // weak PRNG with fixed seed
05  sr = SecureRandom.getInstance("SHA1PRNG");
06  sr.setSeed(hash.getBytes());
07  byte[] keyBytes = new byte[16];
08  sr.nextBytes(keyBytes);
09  // crypto key derived from password
10  ks = new SecretKeySpec(keyBytes,"AES");
```

- MD5 usado no cálculo do hash de senha
- PRNG usa hash de semente fixa
- PRNG gera sempre a mesma chave
- Bug em versões diferentes do Android
  - PRNG se comportava diferente
- Chaves diferentes eram geradas

bitbucket.org/alexmbraga/crypto4developers

THE DEVELOPER'S CONFERENCE





Encriptação determinística

THE DEVELOPER'S CONFERENCE

```
Security.addProvider(new BouncyCastleProvider()); // provedo
byte[] textoClaroAna = ("Deterministica..").getBytes();
KeyGenerator g = KeyGenerator.getInstance("AES", "BC");
g.init(128);
Key k = g.generateKey();
String[] aes = {
  "AES",
  "AES/ECB/NoPadding",
  "AES/ECB/PKCS5Padding",
  "AES/ECB/PKCS7Padding"};

U.println("Texto claro    : " + new String(textoClaroAna));
U.println("Chave AES      : " + U.b2x(k.getEncoded()));
for (int a = 0; a < aes.length; a++) {
  Cipher enc = Cipher.getInstance(aes[a], "BC");
  enc.init(Cipher.ENCRYPT_MODE, k);
  Cipher dec = Cipher.getInstance(aes[a], "BC");
  dec.init(Cipher.DECRYPT_MODE, k);
```

- Criptografia determinística com AES no modo ECB

```
run:
Texto claro    : Deterministica..
Chave AES      : CFBDF66FDAB6A629DB72E5F4CF3E34AE
Encriptado com: AES
Criptograma    : BD32D181CA1B04CD9688F940AF420F96A23294282FA391B56A8E820B913D7BB2
Criptograma    : BD32D181CA1B04CD9688F940AF420F96A23294282FA391B56A8E820B913D7BB2
Encriptado com: AES/ECB/NoPadding
Criptograma    : BD32D181CA1B04CD9688F940AF420F96
Criptograma    : BD32D181CA1B04CD9688F940AF420F96
Encriptado com: AES/ECB/PKCS5Padding
Criptograma    : BD32D181CA1B04CD9688F940AF420F96A23294282FA391B56A8E820B913D7BB2
Criptograma    : BD32D181CA1B04CD9688F940AF420F96A23294282FA391B56A8E820B913D7BB2
Encriptado com: AES/ECB/PKCS7Padding
Criptograma    : BD32D181CA1B04CD9688F940AF420F96A23294282FA391B56A8E820B913D7BB2
Criptograma    : BD32D181CA1B04CD9688F940AF420F96A23294282FA391B56A8E820B913D7BB2
CONSTRUÍDO COM SUCESSO (tempo total: 11 segundos)
```

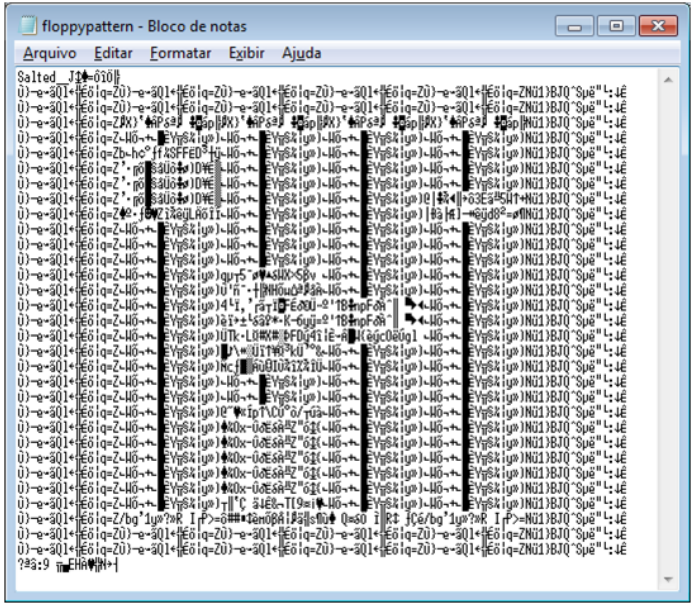
Encriptação determinística

THE DEVELOPER'S CONFERENCE

- Padrões do texto claro geram padrões no criptograma com modo ECB

C:\>openssl enc -e -AES-128-ECB -in floppy.txt -out floppypattern.txt

C:\>type floppy.txt



Encriptação determinística

THE DEVELOPER'S CONFERENCE

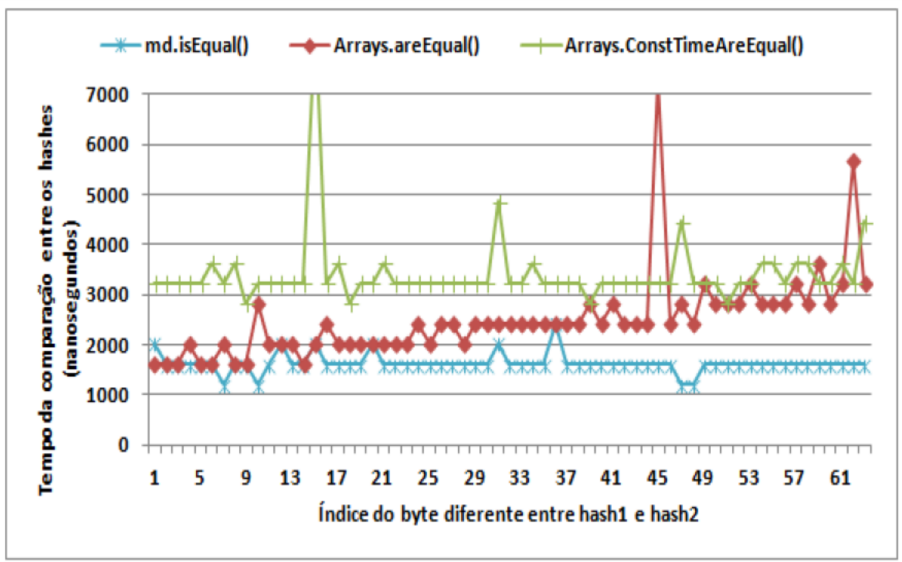
```
Security.addProvider(new BouncyCastleProvider()); // provedor BC
byte[] textoClaroAna = ("Cripto deterministica").getBytes();
//byte[] textoClaroAna = ("Deterministica").getBytes();
KeyPairGenerator g = KeyPairGenerator.getInstance("RSA", "BC");
g.initialize(512);
KeyPair kp = g.generateKeyPair();
String[] rsa = { "RSA",                          // determinístico
  "RSA/ECB/NoPadding",                           // determinístico
  "RSA/None/NoPadding",                          // determinístico
  "RSA/None/PKCS1Padding",                       // pseudo-aleatório
  "RSA/None/OAEPWithSHA1AndMGF1Padding"}; // pseudo-aleatório
U.println("Texto claro    : " + new String(textoClaroAna));
for (int a = 0; a < rsa.length; a++) {
  Cipher enc = Cipher.getInstance(rsa[a], "BC");
  enc.init(Cipher.ENCRYPT_MODE, kp.getPublic());
  Cipher dec = Cipher.getInstance(rsa[a], "BC");
  dec.init(Cipher.DECRYPT_MODE, kp.getPrivate());

  U.println("Encriptado com: " + enc.getAlgorithm());
  byte[][] criptograma = new byte[2][];
  for (int i = 0; i < 2; i++) {
    criptograma[i] = enc.doFinal(textoClaroAna);
    byte[] textoClaroBeto = dec.doFinal(criptograma[i]);
    U.println("Criptograma    : " + U.b2x(criptograma[i]));
    //U.println("Texto claro    : " + new String(textoClaroBeto));
  }
```

- Criptografia determinística assimétrica com RSA

```
run:
Texto claro    : Cripto deterministica
Encriptado com: RSA
Criptograma    : 391C2CB1A0F5132CCF4D111104F7174B76A47757E8B3E36F42AD199BF
Criptograma    : 391C2CB1A0F5132CCF4D111104F7174B76A47757E8B3E36F42AD199BF
Encriptado com: RSA/ECB/NoPadding
Criptograma    : 391C2CB1A0F5132CCF4D111104F7174B76A47757E8B3E36F42AD199BF
Criptograma    : 391C2CB1A0F5132CCF4D111104F7174B76A47757E8B3E36F42AD199BF
Encriptado com: RSA/None/NoPadding
Criptograma    : 391C2CB1A0F5132CCF4D111104F7174B76A47757E8B3E36F42AD199BF
Criptograma    : 391C2CB1A0F5132CCF4D111104F7174B76A47757E8B3E36F42AD199BF
Encriptado com: RSA/None/PKCS1Padding
Criptograma    : 64567FB16598D965FC7FA674B0BFF84B41041EBC29CAD13F21ECED1CD
Criptograma    : A552DB23DE922FEEBE3C2F01ED9571AA4C5BBAEAB87EEC7C99FB8C4F7
Encriptado com: RSA/None/OAEPWithSHA1AndMGF1Padding
Criptograma    : A645D6F5FB183BB4CB28168A2311A906A31DEA5D1449FB73DB0929309
Criptograma    : 1E1626B885D813FE1D18D4AEC7499C4F40BD0786D87A4AA3A0E11BC63
CONSTRUÍDO COM SUCESSO (tempo total: 2 segundos)
```

Canal lateral de tempo

- Arrays.areEqual() revela posição de *mismatch do hash*

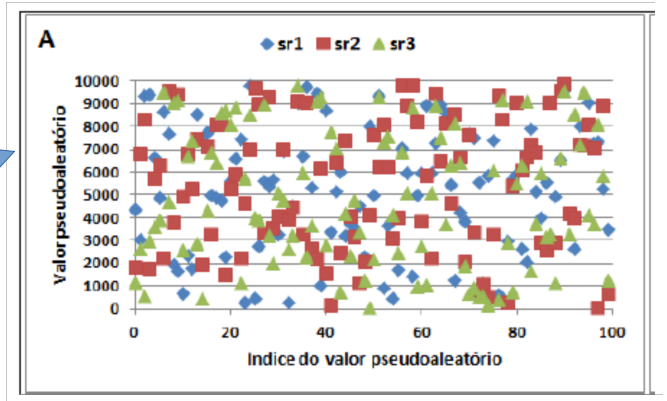


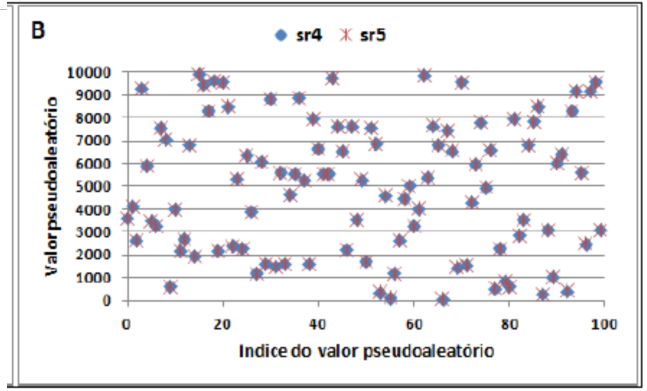

```
for (int i = 0; i < t.length; i++) { // 64 bytes
  md.reset();
  byte[] hash2 = md.digest(cancaoDoExilio.getBytes());
  hash2[i] = (byte) (hash2[i] ^ 0x01);
  t1 = System.nanoTime();
  ok = Arrays.areEqual(hash2, hash1);
  t2 = System.nanoTime();
  tttt[i] = t2 - t1;
}
```

Números pseudoaleatórios

- Mesma semente com o mesmo PRNG = mesma sequência pseudoaleatória
- Isto não é sempre bom!

```
System.out.println("Teste 1: Dispersão estatística - SecureRandom - SUN");
SecureRandom sr1 = SecureRandom.getInstance("SHA1PRNG", "SUN");
SecureRandom sr2 = SecureRandom.getInstance("SHA1PRNG", "SUN");
SecureRandom sr3 = SecureRandom.getInstance("SHA1PRNG", "SUN");
System.out.println("i , sr1 , sr2, sr3");
for (int i = 0; i < 100; i++) {
  U.println(i + ","+
      sr1.nextInt(10000)+","+sr2.nextInt(10000)+","+sr3.nextInt(10000));
}
System.out.println("Teste 2: Imprevisibilidade - SecureRandom - SUN");
SecureRandom sr4 = SecureRandom.getInstance("SHA1PRNG", "SUN");
SecureRandom sr5 = SecureRandom.getInstance("SHA1PRNG", "SUN");
byte[] seed = sr4.generateSeed(32);
sr4.setSeed(seed); sr5.setSeed(seed); // mesma semente
System.out.println("i , sr4 , sr5");
for (int i = 0; i < 100; i++) {
  U.println(i + "," + sr4.nextInt(10000) + ","+ sr5.nextInt(10000));
}
```





Mau uso de Streamcipher

THE DEVELOPER'S CONFERENCE

- Uso incorrecto do RC4 pelo WhatsApp facilitou a revelação de mensagens (out. 2013)
  - https://blog.thijsalkema.de/blog/2013/10/08/piercing-through-whatsapp-s-encryption/
- Permitia a decriptação de mensagens trocadas entre usuários
- Dois problemas de mau uso da criptografia
  - Uso incorreto do RC4
  - Reuso de chaves criptográficas nos dois canais de comunicação
- Dois desentendimentos sobre streamciphers
  - RC4 (streamcipher) não pode subistituir AES/CBC (blockcipher)
  - Reuso de chaves em streamciphers

Trocando bloco por fluxo

THE DEVELOPER'S CONFERENCE

```
Security.addProvider(new BouncyCastleProvider()); // provedor BC
byte[][] M = {("Troca a cifra de").getBytes(),
              ("bloco por fluxo.").getBytes()};
byte[][] C = new byte[2][], iv = new byte[2][];

byte[] k = U.x2b("00112233445566778899AABBCCDDEEFF");
byte[] seed = U.x2b("0123456789ABCDEF0123456789ABCDEF");

SecretKeySpec ks = new SecretKeySpec(k, "AES");
Cipher enc = Cipher.getInstance("AES/CBC/NoPadding", "BC");
//Cipher enc = Cipher.getInstance("AES/OFB/NoPadding", "BC");

SecureRandom sr = new SecureRandom();
sr.setSeed(seed);
enc.init(Cipher.ENCRYPT_MODE, ks, sr);
C[0] = enc.doFinal(M[0]);
iv[0] = enc.getIV();

sr = new SecureRandom();
sr.setSeed(seed);
enc.init(Cipher.ENCRYPT_MODE, ks, sr);
C[1] = enc.doFinal(M[1]);
iv[1] = enc.getIV();

byte[] M0xorM1 = U.xor(C[0], C[1]);
byte[] M1 = U.xor(M[0], M0xorM1);
```

AES/CBC/NoPadding

```
M0 = Troca a cifra de; M1 = bloco por fluxo.
Encriptado com: AES/CBC/NoPadding
C[0] =8C61DC63FF9682588910E6FF77866E58
iv[0]=BB3E5CFEDD9A5E6F666AC3ACB3D56D1C
C[1] =00D0625EC980B1FD965931601CE279C3
iv[1]=BB3E5CFEDD9A5E6F666AC3ACB3D56D1C
C0^C1 = k^M0^K^M1 = M0^M1 = 8CB1BE3D361633A51F49D79F6B64179B
M0^M1^M0 = M1 = □□□^W6R□| □□
Ds□
CONSTRUÍDO COM SUCESSO (tempo total: 7 segundos)
```

Trocando bloco por fluxo

THE DEVELOPER'S CONFERENCE

```
Security.addProvider(new BouncyCastleProvider()); // provedor BC
byte[][] M = {("Troca a cifra de").getBytes(),
              ("bloco por fluxo.").getBytes()};
byte[][] C = new byte[2][], iv = new byte[2][];

byte[] k = U.x2b("00112233445566778899AABBCCDDEEFF");
byte[] seed = U.x2b("0123456789ABCDEF0123456789ABCDEF");

SecretKeySpec ks = new SecretKeySpec(k, "AES");
//Cipher enc = Cipher.getInstance("AES/CBC/NoPadding", "BC");
Cipher enc = Cipher.getInstance("AES/OFB/NoPadding", "BC");

SecureRandom sr = new SecureRandom();
sr.setSeed(seed);
enc.init(Cipher.ENCRYPT_MODE, ks, sr);
C[0] = enc.doFinal(M[0]);
iv[0] = enc.getIV();

sr = new SecureRandom();
sr.setSeed(seed);
enc.init(Cipher.ENCRYPT_MODE, ks, sr);
C[1] = enc.doFinal(M[1]);
iv[1] = enc.getIV();

byte[] M0xorM1 = U.xor(C[0], C[1]);
byte[] M1 = U.xor(M[0], M0xorM1);
```

AES/CBC/NoPadding

```
M0 = Troca a cifra de; M1 = bloco por fluxo.
Encriptado com: AES/CBC/NoPadding
C[0] =8C61DC63FF9682588910E6FF77866E58
iv[0]=BB3E5CFEDD9A5E6F666AC3ACB3D56D1C
C[1] =00D0625EC980B1FD965931601CE279C3
iv[1]=BB3E5CFEDD9A5E6F666AC3ACB3D56D1C
C0^C1 = k^M0^K^M1 = M0^M1 = 8CB1BE3D361633A51F49D79F6B64179B
M0^M1^M0 = M1 = □□□^W6R□| □□
Ds□
CONSTRUÍDO COM SUCESSO (tempo total: 7 segundos)
```

AES/OFB/NoPadding

```
M0 = Troca a cifra de; M1 = bloco por fluxo.
Encriptado com: AES/OFB/NoPadding
C[0] =29CF058FFD0543D255888BC1A12F33DD
iv[0]=BB3E5CFEDD9A5E6F666AC3ACB3D56D1C
C[1] =1FD1058FF305529D44C18BDFB5773896
iv[1]=BB3E5CFEDD9A5E6F666AC3ACB3D56D1C
C0^C1 = k^M0^K^M1 = M0^M1 = 361E00000E00114F1149001E14580B4B
M0^M1^M0 = M1 = bloco por fluxo.
CONSTRUÍDO COM SUCESSO (tempo total: 4 segundos)
```

Streamcipher maleável

THE DEVELOPER'S CONFERENCE

```
    Security.addProvider(new BouncyCastleProvider()); // provedor BC
  byte[][]M={("Ana   para Carlo").getBytes(),("Valor:010.000,00").getBytes()};
  byte[][] iv = {U.x2b("0123456789ABCDEF0123456789ABCDEF"),
                 U.x2b("0123456789ABCDEF0123456789ABCDEF")};
  byte[][] iv2 ={iv[0].clone(),iv[0].clone()};
  byte[] k = U.x2b("00112233445566778899AABBCCDDEEFF"), X = null;
  byte[][] C = new byte[2][], N = new byte[2][];

  SecretKeySpec ks = new SecretKeySpec(k, "AES");
  Cipher c = Cipher.getInstance("AES/CTR/NoPadding", "BC");

  for (int i= 0; i < M.length; i++){
    c.init(Cipher.ENCRYPT_MODE, ks, new IvParameterSpec(iv[i]));
    C[i] = c.doFinal(M[i]);
    if (i < M.length -1) U.stdIncIV(iv[i+1],iv[i+1].length/2);
  }

  //Ivo passa a ser o novo recebedor de um valor muito mais alto
  //X = U.xor("Ana   para Carlo".getBytes(),"Ana   para   Ivo".getBytes());
  //C[0] = U.xor(C[0], X);
  //X = U.xor("Valor:010.000,00".getBytes(),"Valor:100.998,54".getBytes());
  //C[1] = U.xor(C[1], X);

  for (int i = 0; i < C.length; i++){
    c.init(Cipher.DECRYPT_MODE, ks, new IvParameterSpec(iv2[i]));
    N[i] = c.doFinal(C[i]);
    if (i < C.length -1) U.stdIncIV(iv2[i+1],iv2[i+1].length/2);
    U.println("N[" + i + "] =" + U.b2s(N[i]));
  }
```

Sem manipulação maliciosa

```
run:
N[0] =Ana   para Carlo
N[1] =Valor:010.000,00
CONSTRUÍDO COM SUCESSO (tempo total: 2 segundos)
```

Streamcipher maleável

THE DEVELOPER'S CONFERENCE

```
  Security.addProvider(new BouncyCastleProvider()); // provedor BC
byte[][]M={("Ana   para Carlo").getBytes(),("Valor:010.000,00").getBytes()};
byte[][] iv = {U.x2b("0123456789ABCDEF0123456789ABCDEF"),
               U.x2b("0123456789ABCDEF0123456789ABCDEF")};
byte[][] iv2 ={iv[0].clone(),iv[0].clone()};
byte[] k = U.x2b("00112233445566778899AABBCCDDEEFF"), X = null;
byte[][] C = new byte[2][], N = new byte[2][];

SecretKeySpec ks = new SecretKeySpec(k, "AES");
Cipher c = Cipher.getInstance("AES/CTR/NoPadding", "BC");

for (int i= 0; i < M.length; i++){
  c.init(Cipher.ENCRYPT_MODE, ks, new IvParameterSpec(iv[i]));
  C[i] = c.doFinal(M[i]);
  if (i < M.length -1) U.stdIncIV(iv[i+1],iv[i+1].length/2);
}

 //Ivo passa a ser o novo recebedor de um valor muito mais alto
 X = U.xor("Ana   para Carlo".getBytes(),"Ana   para   Ivo".getBytes());
 C[0] = U.xor(C[0], X);
 X = U.xor("Valor:010.000,00".getBytes(),"Valor:100.998,54".getBytes());
 C[1] = U.xor(C[1], X);

for (int i = 0; i < C.length; i++){
  c.init(Cipher.DECRYPT_MODE, ks, new IvParameterSpec(iv2[i]));
  N[i] = c.doFinal(C[i]);
  if (i < C.length -1) U.stdIncIV(iv2[i+1],iv2[i+1].length/2);
  U.println("N[" + i + "] =" + U.b2s(N[i]));
}
```

Sem manipulação maliciosa

```
run:
N[0] =Ana   para Carlo
N[1] =Valor:010.000,00
CONSTRUÍDO COM SUCESSO (tempo total: 2 segundos)
```

Manipulação do 1º. Bloco: Ivo e não Carlo!
Manipulação do 2º. Bloco: valor abusivo!

```
run:
N[0] =Ana   para   Ivo
N[1] =Valor:100.998,54
CONSTRUÍDO COM SUCESSO (tempo total: 4 segundos)
```

Combo hash + encriptação

THE DEVELOPER'S CONFERENCE

```
// configurações do sistema criptográfico para Ana e Beto
byte[] k =  U.x2b("0123456789ABCDEF0123456789ABCDEF");// 128 bits
byte[] iv = U.x2b("0123456789ABCDEF0123456789ABCDEF");
SecretKeySpec sks1 = new SecretKeySpec(k, "AES");
Cipher c = Cipher.getInstance("AES/CTR/NoPadding", "BC");
MessageDigest md = MessageDigest.getInstance("SHA256", "BC");
byte[] textoclaroAna = "De Ana para Beto".getBytes(), X, Y;
boolean ok, ivo = true;

// encripta entao hash: Encrypt-then-Hash (EtH)
String s = "Encrypt-then-Hash(EtH): calcula o hash do criptograma";
md.reset(); c.init(Cipher.ENCRYPT_MODE, sks1, new IvParameterSpec(iv));
byte[] criptograma = c.doFinal(textoclaroAna);
byte[] hash = md.digest(criptograma); // calcula a hash do criptograma

if (ivo){
  X = U.xor("De Ana para Beto".getBytes(), "De Ana para  Ivo".getBytes());
  criptograma = U.xor(criptograma, X);
  hash = md.digest(criptograma);
}

// decriptação pelo Beto com verificação da hash
md.reset(); c.init(Cipher.DECRYPT_MODE, sks1, new IvParameterSpec(iv));
byte[] textoclaroBeto = c.doFinal(criptograma); // decripta
ok = MessageDigest.isEqual(md.digest(criptograma), hash); // verifica hash
p(s,hash,textoclaroBeto,md,c,ok);
```

```
Encrypt-then-Hash(EtH): calcula o hash do criptograma
Verificado com SHA-256: true
Encriptado com: AES/CTR/NoPadding
Valor da   tag: AB28E1848F3384EC66E5D2CC16B41A12050260A6CF3C2B8C89A81B5B4A6FA27F
Texto claro do Beto: De Ana para  Ivo

Encrypt-and-Hash (E&H): calcula o hash do texto claro
Verificado com SHA-256: true
Encriptado com: AES/CTR/NoPadding
Valor da   tag: 3971C3ED5E50D6FE7F2F637AC184BAF3EB48344546639104D69A35FA6644BF23
Texto claro do Beto: De Ana para  Ivo

Hash-then-Encrypt(HtE):calcula o hash do texto claro e encripta a tag
Verificado com SHA-256: true
Encriptado com: AES/CTR/NoPadding
Valor da   tag: 3971C3ED5E50D6FE7F2F637AC184BAF3EB48344546639104D69A35FA6644BF23
Texto claro do Beto: De Ana para  Ivo

CONSTRUÍDO COM SUCESSO (tempo total: 5 segundos)
```

Combo MAC + encriptação

THE DEVELOPER'S CONFERENCE

```
// configurações do sistema criptográfico para Ana e Beto
byte[] k =  U.x2b("0123456789ABCDEF0123456789ABCDEF");// 128 bits
byte[] iv = U.x2b("0123456789ABCDEF0123456789ABCDEF");
SecretKeySpec sks1 = new SecretKeySpec(k, "AES");
SecretKeySpec sks2 = new SecretKeySpec(k, "HMACSHA256");
Cipher c = Cipher.getInstance("AES/CTR/NoPadding", "BC");
Mac m = Mac.getInstance("HMACSHA256", "BC");
byte[] textoclaroAna = "De Ana para Beto".getBytes(), X;
boolean ok, ivo = true;

// encripta entao autentica: Encrypt-then-MAC (EtM)
String s = "Encrypt-then-MAC(EtM): calcula a tag do criptograma";
m.init(sks2); c.init(Cipher.ENCRYPT_MODE, sks1, new IvParameterSpec(iv));
byte[] criptograma = c.doFinal(textoclaroAna);
byte[] tag = m.doFinal(criptograma); // calcula a tag do criptograma

if (ivo){
  X = U.xor("De Ana para Beto".getBytes(), "De Ana para  Ivo".getBytes());
  criptograma = U.xor(criptograma, X);
}

// decriptação pelo Beto com verificação da tag
m.init(sks2); c.init(Cipher.DECRYPT_MODE, sks1, new IvParameterSpec(iv));
byte[] textoclaroBeto = c.doFinal(criptograma); // decripta
ok = MessageDigest.isEqual(m.doFinal(criptograma), tag); // verifica tag
p(s,tag,textoclaroBeto,m,c,ok);
```

```
Encrypt-then-MAC(EtM): calcula a tag do criptograma
Verificado com HMACSHA256: false
Encriptado com: AES/CTR/NoPadding
Valor da   tag: 43E79D846FB301A4D1ED90C3CBDC62E309E4B1B86787109788C63DEFF86B7F4F
Texto claro do Beto: De Ana para  Ivo

Encrypt-and-MAC (E&M): calcula a tag do texto claro
Verificado com HMACSHA256: false
Encriptado com: AES/CTR/NoPadding
Valor da   tag: 06D97392F2A8F00D24EEC570CB25F0D4A50464709614CFAD1237D403E0BA61F6
Texto claro do Beto: De Ana para  Ivo

MAC-then-Encrypt (MtE): calcula a tag do texto claro
Verificado com HMACSHA256: false
Encriptado com: AES/CTR/NoPadding
Valor da   tag: 06D97392F2A8F00D24EEC570CB25F0D4A50464709614CFAD1237D403E0BA61F6
Texto claro do Beto: De Ana para  Ivo

CONSTRUÍDO COM SUCESSO (tempo total: 4 segundos)
```

Encriptação autenticada

THE DEVELOPER'S CONFERENCE

```
SecretKeySpec ks = new SecretKeySpec(k, "AES");
GCMParameterSpec gps = new GCMParameterSpec(128, iv);//tag size + iv
Cipher c = Cipher.getInstance("AES/GCM/NoPadding", "BC");

// Encriptação pela Ana
c.init(Cipher.ENCRYPT_MODE, ks, gps); //inicializa o AES para encriptacao
byte[] textoclaroAna = "Testando o GCM..".getBytes();
c.updateAAD("AAD nao estah cifrado...".getBytes());
byte[] criptograma = c.doFinal(textoclaroAna);

//criptograma[0] = (byte)(criptograma[0]^(byte)0x01);
// decriptação pelo Beto
c.init(Cipher.DECRYPT_MODE, ks, gps); //inicializa o AES para decriptacao
c.updateAAD("AAD nao estah cifrado...".getBytes());
boolean ok = true;
byte[] textoclaroBeto = null;
try {
  textoclaroBeto = c.doFinal(criptograma); // Decriptando
} catch (AEADBadTagException e) { ok = false; }
if (ok) {
  U.println("Algoritmo "+c.getAlgorithm()+",tag de "+gps.getTLen()+" bits");
  U.println("Criptograma: " + U.b2x(criptograma)+", "+criptograma.length);
  U.println("Texto claro: " + new String(textoclaroBeto));
} else {
  U.println("Tag não verificada!");
}
```

```
Algoritmo AES/GCM/NoPadding,tag de 128 bits
Criptograma: E34E8BF4D79711FA0243F4E30B978EDE773722A87646C6B1E48CF0A99AE8577C, 32
Texto claro: Testando o GCM..
```

```
run:
Tag não verificada!
```

Concluindo

THE DEVELOPER'S CONFERENCE

- É muito fácil usa errado uma API criptográfica
- Muitas vezes é difícil ver os problemas (falta conhecimento)
- O nível de abstração das APIs é inadequado
- As ferramentas SAST não ajudam muito

- Muitas oportunidades em
  - Novas APIs criptográficas com foco em casos de uso de programação
  - Ferramentas de análise estática de código fonte para criptografia

Onde achar ajuda?

THE DEVELOPER'S CONFERENCE

ATMOSPHERE
Adaptive, Trustworthy, Manageable, Orchestrated, Secure Privacy-assuring Hybrid, Ecosystem for REsilient Cloud Computing

LASCA
LABORATÓRIO DE SEGURANÇA E CRIPTOGRAFIA

UNICAMP

Dr. Alexandre Braga

Postdoctoral Researcher at UNICAMP

(ISC)² Instructor, Consultant, PMP, CSSLP, CISSP

alexbraga@ic.unicamp.br

br.linkedin.com/in/alexmbraga

bitbucket.org/alexmbraga/crypto4developers

THE DEVELOPER'S CONFERENCE

Este trabalho foi realizado no Laboratório de Segurança e Criptografia (LASCA) do Instituto de Computação, Universidade Estadual de Campinas. O instrutor agradece o apoio financeiro do projeto ATMOSPHERE, RNP e MCTIC, no âmbito to acordo de cooperação Número 51119 (*ATMOSPHERE is funded by the European Commission under the Cooperation Programme, Horizon 2020 grant agreement No 777154*) e da Fapesp, por meio do projeto temático Segurança e Confiabilidade da Informação: Teoria e Aplicações, no. 2013/25.977-7.