Segurança em Sistemas de Computação

Desafio 06 - Parte 1
(AES - 128 bits Com Chave de Sessão)

INF01045 – Comunicação – Turma U – Prof. Raul Weber

Luiz Gustavo Frozi de Castro e Souza - Cartão 96957
Mário César Gasparoni Jr. - Cartão 151480

Setembro de 2013

Texto Cifrado (notação hexadecimal):

62876D72F63FA4416232323FCD174AA5
24D6CCC8FEA90D80BB0D760484FF250D
55E1379F1A9AEA1CCB4F78A493A52D3B
366DF3E0D8FE243C678EA5684CA93BA9
2223F0319C61F988F8458629DC40F012
D4CA5B4F9EB764280CACF7D94BB6AD72
45BF11ACEEE09B345EBB3C7FEE4DE538
A88A7581C03489671C39E3E8D7BBB583
E5AC55D051D25AC0D3037894F84448EC
FD2BA8F131FDCDE263350EDB2FF05788
406B31C60D98C82FCE5691048D941297
AEA2A59EFE1F0F302B772E3FA71ADC2B
F41F2A9366D23A6466F0189FAA64DBD2
A2886B04E44ADD5DA017750ABE3B9640
64830D476141C52ADF1A1F6CEF9B251F
0B32B0939F0D39649E922D9079D5F40D
37223AEBDA6B8A9CE90C6AF7377314CF
BB8536DA945BACF0A49806E203AF9360
AB99EE8C6B039DAFF19B98F9B022DF31
45A9BEED64D31BAC0E0D4C123CF75414
63C4E0B54D9EA3672C8A01278D76D8AB
51AF84E6AA724D88ACBDF38C5A97536B
0663A73267A1B844B6C7964EB37F5DC6
00359B2A6B2A5F84746DB2AD3C673E96
0DFE64AA45DC88551DDC719097394D81
64B85E0D31E0E0F9101E7B81CBAE9A10
87352B318B6A922F0F13C55D03ED3616
EBEC0742A7609730860BB63FED4D87AB
2A38410548B923D84E963E76434C4390
EDBB171C2CAAFE26CB15ED5397CD6072
8326D2B0688AF789977F11A2F6568F94
E97D50D3B537600C8BDAA679DA9DF008
40A369C1F8D33AC3270DE9A930E27199
2C046043401CBDF5056F072B89ACCA9E
B679E8EE601A2319D23D71DC16A069A3
BB8D80A332F807F4F530D48643234A70
17F3691B907822EC6C1D7F2291AC2840
F64640F53B639E2FBDE92AAC2D9C7001
7195DF585525B1074D055574A6BC539E
00B68BAED5E9D42944B17F276BE5C177
3D73DF93029D323C9C3727F28212A034
71EAF2195578CEC509926EB9B7F03D5B
58242C0FF430AC3EE612D4E23C260A38
5E6ADE558D68B0E6D426B653BAF1D5ED
846157BB8A7BFD37BC26546C6566A4DF

537A2B1074A990143DFED27F974FA118
E1D346FE32CB9E510802CE1AEDAC800F
F6AA7193C300A28881AC2139E900E332
1CD23686429A1B96BF08F1EAC2DA03A2
392FA0CC520732602B0908BBC6E1FAC5
B43621C47BD4B691FD23BB4C8C72C7F3
B9DCBCE039FEA805A2A3C2461D7F6D20
0EC3BA63FE34B07E64ED88B1381BA6F5
38D9D9868FA77585846604CD68004DE1
F3496484D8805A464845180A372A783A
EC7A5A18E5CB5F85C07E0DB47B3A6176
45DDB95EDF2310445952F8D8F7891919
53C2EAC56834F10C28C1C6BC0C211BBE
2923F64FCB596779BD306ADD5198C834
434B417A75DACE44170C57873B7A1017
CFF67B5091BB2E174E16E89E79EE48B8
FA02E504ED5F95282F72936076B4977A
F50BED746F702684676E83008A6BF332
A377160D0A5787573A31B2F7496CD9FE
3B7F8211933A2BA58ED49EC8E1AA1F44
0577005A5EDD846716546217F48F0D11
ADAF225B3D710D5F91DC0B644C93069B
AEF1DD06B4884B5C5E04343FB444E8B1
C4D4032FA3F1130C6FED663EB76E8F88
FDA87820B78E26CB2547EC0FEA40AB17
117B451382052780E7430D2BD6A3662D
A24A4182E5627AD9BFE5BF24D437CE96
7CE3DF19F8F87F6928C0FE25DD0E2F72
ADA79F05D18516E3FA46023DAD49690E
53FB73BC1152061613349E09D22A703B
71A1A5CAE851CE8981A72555EAC8C0CC
DBCCD65C9F156BDD8037AB7AB57C42B9
B42A2402D14A8EB94170644F37396803
C7A0ED7B3CDFB45FED182B6D1C21575A
3910150CA7C9A6623C3706DD913A9887
8164ACF4C1F60D8CB5C1F9DB846911EC
DF774ABB054F12D4433322FFDF5D122F
F98AC0A783A5DC3D3A4C5A54320C02FD
CDC4101626FE13CA1606C03EB83DCB3D
9045EABC2B8C3B608E270653076912B3
EE234B5FA10D3944B7A35E7E84FA0FEC
CB7D6A691F941A86CB715B4F03FC2202
FC8B3E0210CCB974128A92DF746B9529
FC7E72127701F6C3E264D094EC16632C
A30F49DF6DD1BC992D42082E8AD79E9A
48DA0A20817FD75A8A121AB126138E9D
F4212EC1CE368350D85FB7091D53D61E
FF2ADEE6B06319864F705A7792539931
DE6BFC90AE5118EF06A11198513466BE
D4FD68518A56D5E9BB7B1A3F48B41CEC
24972F4C1738475B8580F8AE02F6FB87
9119E1EBF6AF543C11CB3D539CFCB8C9

Texto Decifrado:

Sabe, estive pensando, e o melhor é usar o RSA no lugar do Diffie Hellman. Andei lendo um pouco a respeito, e não é tão complicado assim. Os cálculos são bem parecidos, e com o RSA dá até para fazer

assinatura digital. Quer ver?

Vamos precisar de dois primos. Um deles eu já gerei: 57007341816453784345611883741305290721948860641 69

Vou enviar o segundo primo no segundo arquivo.

Se por acaso você não conseguir decifrar o segundo arquivo, então faça o seguinte:

1 - Gere um segundo primo (maior que o primo dado acima, por favor) - para isto, é bom usar uma função de teste de primalidade, ou então implementar um teste determinístico.

2 - Calcule o "n" do RSA multiplicando estes dois primos (o que está neste arquivo e o que você gerou).

3 - Calcule (n,e) e (n,d), as tuas chaves pública e privada, de acordo com o indicado pelo RSA.

4 - Assine o "desafio" abaixo com a tua chave secreta:
508963806460217157251376378453850802085

Isso é bem seguro. Tão seguro que eu até lanço um desafio aqui: se algum aluno estiver lendo isto, então siga as mesmas instruções e coloque no Moodle a chave pública (o par (n,e)) e o desafio assinado (ou seja, o resultado de (desafio^d mod n))

E sabe de uma coisa? Descobri um ótimo gerador de randômicos. Daqui para a frente, vou usar somente ele:

2^x . 3^y . 5^w . 7^z

onde x, y, w e z são inteiros positivos maiores que zero.

Assinado: BOB

ATENÇÃO: para ser dispensado da prova de criptografia, você tem que decifrar o segundo arquivo!!

Chave de sessão (Notação Hexadecimal):
B3E8C0D7F9C724822B7FE70FB34DD198.

Passo a Passo da Análise efetuada:
Quase todo o ataque foi planejado com base nas mensagens capturadas durante a conversa. Os seguintes dados serviram de base para o protocolo de Diffie-Hellman:

1. Número Primo **n = 340282366920938463463374607431768211297**
2. Segundo número **g = 339661812359158752487805590648382723989**
3. Número **X = 234399727922910008434346336719740475791** passado pela Alice para Bob, resultante do cálculo X = x * g mod n, onde **x** é o número escolhido aleatoriamente por Alice.
4. Número **Y = 317353208171493985828526472609655218772** passado por Bob para Alice, resultante do cálculo Y = y * g mod n, onde **y** é o número escolhido aleatoriamente por Bob.
5. Variante **multiplicativa** do protocolo Diffie-Hellman.
6. A mensagem será cifrada com base no algoritmo AES de 128 bits.
7. A chave para cifragem do AES será composta por 128 bits extraídos da chave de sessão conforme o algoritmo:
   - for i := 0 to 15
   - begin
   - bytechave[i] := n mod 256;

- n := n div 256;
- end;
- n é o número da chave de sessão e bytechave[i] são os bytes da chave AES

Sabemos que o protocolo não requer que essas informações estejam ocultas, podem ser públicas e/ou transmitidas em texto claro.
A força criptográfica do protocolo baseia-se que o atacante deve realizar resolver um problema NP-Difícil para encontrar a chave, não deve apenas considerar que os números escolhidos sejam grandes. Com base nisso, podemos ver que o problema aqui foi a escolha do algoritmo do protocolo, que usa uma variante multiplicativa. Esse problema não é NP-Difícil.
Para encontrar a chave de sessão, apenas foi suficiente calcular o inverso multiplicativo de **g** em módulo **n**, ou seja, g^{-1}, e multiplicar pelos números aleatórios trocados entre Alice e Bob, ou seja, $\mathbf{K = X * Y * g^{-1} \bmod n}$. De posse da chave de sessão, basta aplicar o algoritmo trocado e obter os 128 bits da chave do AES.

Códigos-fonte utilizados:

Programa de Ataque (desafio6-p1.py):

```
1.  ## Universidade Federal do Rio Grande do Sul - Instituto de Informatica
2.  ## Departamento de Informatica Aplicada
3.  ## Seguranca em Sistemas de Computacao - 2013/2
4.  ## Professor: Raul Fernando Weber
5.  ## Alunos: Luiz Gustavo Frozi e  Mario Gasparoni Junior
6.  ##
7.  ## Python 3.2
8.  ##
9.  ## Desafio 6 - Parte 1
10. ##
11. ## Script que executa encontra a chave de sessao de um algoritmo Diffie-Hellman
12. ## multiplicativo.
13. ##
14. from Crypto.Cipher import AES
15. from Crypto import Random
16. import array
17. #
18. # Infomacoes Interceptadas
19. #
20.
21. # Numero Primo
22. n = 340282366920938463463374607431768211297
23.
24. # Segundo Numero
25. g = 33966181235915875248780559064838272398 9
26.
27. # Alice -> Bob : X = x * g mod n
28. X = 234399727922910008434346336719740475791
29.
30. # Bob -> Alice : Y = y * g mod n
31. Y = 317353208171493985828526472609655218772
32.
33. # Mensagem Cifrada
34. cyphered = [] #cada bloco eh um elemento da lista
35. cyphered.append("62876D72F63FA4416232323FCD174AA5".decode("hex"))
36. cyphered.append("24D6CCC8FEA90D80BB0D760484FF250D".decode("hex"))
37. cyphered.append("55E1379F1A9AEA1CCB4F78A493A52D3B".decode("hex"))
38. cyphered.append("366DF3E0D8FE243C678EA5684CA93BA9".decode("hex"))
39. cyphered.append("2223F0319C61F988F8458629DC40F012".decode("hex"))
40. cyphered.append("D4CA5B4F9EB764280CACF7D94BB6AD72".decode("hex"))
41. cyphered.append("45BF11ACEEE09B345EBB3C7FEE4DE538".decode("hex"))
42. cyphered.append("A88A7581C03489671C39E3E8D7BBB583".decode("hex"))
43. cyphered.append("E5AC55D051D25AC0D3037894F84448EC".decode("hex"))
44. cyphered.append("FD2BA8F131FDCDE263350EDB2FF05788".decode("hex"))
45. cyphered.append("406B31C60D98C82FCE5691048D941297".decode("hex"))
46. cyphered.append("AEA2A59EFE1F0F302B772E3FA71ADC2B".decode("hex"))
```

```
cyphered.append("F41F2A9366D23A6466F0189FAA64DBD2".decode("hex"))
cyphered.append("A2886B04E44ADD5DA017750ABE3B9640".decode("hex"))
cyphered.append("64830D476141C52ADF1A1F6CEF9B251F".decode("hex"))
cyphered.append("0B32B0939F0D39649E922D9079D5F40D".decode("hex"))
cyphered.append("37223AEBDA6B8A9CE90C6AF7377314CF".decode("hex"))
cyphered.append("BB8536DA945BACF0A49806E203AF9360".decode("hex"))
cyphered.append("AB99EE8C6B039DAFF19B98F9B022DF31".decode("hex"))
cyphered.append("45A9BEED64D31BAC0E0D4C123CF75414".decode("hex"))
cyphered.append("63C4E0B54D9EA3672C8A01278D76D8AB".decode("hex"))
cyphered.append("51AF84E6AA724D88ACBDF38C5A97536B".decode("hex"))
cyphered.append("0663A73267A1B844B6C7964EB37F5DC6".decode("hex"))
cyphered.append("00359B2A6B2A5F84746DB2AD3C673E96".decode("hex"))
cyphered.append("0DFE64AA45DC88551DDC719097394D81".decode("hex"))
cyphered.append("64B85E0D31E0E0F9101E7B81CBAE9A10".decode("hex"))
cyphered.append("87352B318B6A922F0F13C55D03ED3616".decode("hex"))
cyphered.append("EBEC0742A7609730860BB63FED4D87AB".decode("hex"))
cyphered.append("2A38410548B923D84E963E76434C4390".decode("hex"))
cyphered.append("EDBB171C2CAAFE26CB15ED5397CD6072".decode("hex"))
cyphered.append("8326D2B0688AF789977F11A2F6568F94".decode("hex"))
cyphered.append("E97D50D3B537600C8BDAA679DA9DF008".decode("hex"))
cyphered.append("40A369C1F8D33AC3270DE9A930E27199".decode("hex"))
cyphered.append("2C046043401CBDF5056F072B89ACCA9E".decode("hex"))
cyphered.append("B679E8EE601A2319D23D71DC16A069A3".decode("hex"))
cyphered.append("BB8D80A332F807F4F530D48643234A70".decode("hex"))
cyphered.append("17F3691B907822EC6C1D7F2291AC2840".decode("hex"))
cyphered.append("F64640F53B639E2FBDE92AAC2D9C7001".decode("hex"))
cyphered.append("7195DF585525B1074D055574A6BC539E".decode("hex"))
cyphered.append("00B68BAED5E9D42944B17F276BE5C177".decode("hex"))
cyphered.append("3D73DF93029D323C9C3727F28212A034".decode("hex"))
cyphered.append("71EAF2195578CEC509926EB9B7F03D5B".decode("hex"))
cyphered.append("58242C0FF430AC3EE612D4E23C260A38".decode("hex"))
cyphered.append("5E6ADE558D68B0E6D426B653BAF1D5ED".decode("hex"))
cyphered.append("846157BB8A7BFD37BC26546C6566A4DF".decode("hex"))
cyphered.append("537A2B1074A990143DFED27F974FA118".decode("hex"))
cyphered.append("E1D346FE32CB9E510802CE1AEDAC800F".decode("hex"))
cyphered.append("F6AA7193C300A28881AC2139E900E332".decode("hex"))
cyphered.append("1CD23686429A1B96BF08F1EAC2DA03A2".decode("hex"))
cyphered.append("392FA0CC520732602B0908BBC6E1FAC5".decode("hex"))
cyphered.append("B43621C47BD4B691FD23BB4C8C72C7F3".decode("hex"))
cyphered.append("B9DCBCE039FEA805A2A3C2461D7F6D20".decode("hex"))
cyphered.append("0EC3BA63FE34B07E64ED88B1381BA6F5".decode("hex"))
cyphered.append("38D9D9868FA77585846604CD68004DE1".decode("hex"))
cyphered.append("F3496484D8805A464845180A372A783A".decode("hex"))
cyphered.append("EC7A5A18E5CB5F85C07E0DB47B3A6176".decode("hex"))
cyphered.append("45DDB95EDF2310445952F8D8F7891919".decode("hex"))
cyphered.append("53C2EAC56834F10C28C1C6BC0C211BBE".decode("hex"))
cyphered.append("2923F64FCB596779BD306ADD5198C834".decode("hex"))
cyphered.append("434B417A75DACE44170C57873B7A1017".decode("hex"))
cyphered.append("CFF67B5091BB2E174E16E89E79EE48B8".decode("hex"))
cyphered.append("FA02E504ED5F95282F72936076B4977A".decode("hex"))
cyphered.append("F50BED746F702684676E83008A6BF332".decode("hex"))
cyphered.append("A377160D0A5787573A31B2F7496CD9FE".decode("hex"))
cyphered.append("3B7F8211933A2BA58ED49EC8E1AA1F44".decode("hex"))
cyphered.append("0577005A5EDD846716546217F48F0D11".decode("hex"))
cyphered.append("ADAF225B3D710D5F91DC0B644C93069B".decode("hex"))
cyphered.append("AEF1DD06B4884B5C5E04343FB444E8B1".decode("hex"))
cyphered.append("C4D4032FA3F1130C6FED663EB76E8F88".decode("hex"))
cyphered.append("FDA87820B78E26CB2547EC0FEA40AB17".decode("hex"))
cyphered.append("117B451382052780E7430D2BD6A3662D".decode("hex"))
cyphered.append("A24A4182E5627AD9BFE5BF24D437CE96".decode("hex"))
cyphered.append("7CE3DF19F8F87F6928C0FE25DD0E2F72".decode("hex"))
cyphered.append("ADA79F05D18516E3FA46023DAD49690E".decode("hex"))
cyphered.append("53FB73BC1152061613349E09D22A703B".decode("hex"))
cyphered.append("71A1A5CAE851CE8981A72555EAC8C0CC".decode("hex"))
cyphered.append("DBCCD65C9F156BDD8037AB7AB57C42B9".decode("hex"))
cyphered.append("B42A2402D14A8EB94170644F37396803".decode("hex"))
cyphered.append("C7A0ED7B3CDFB45FED182B6D1C21575A".decode("hex"))
cyphered.append("3910150CA7C9A6623C3706DD913A9887".decode("hex"))
cyphered.append("8164ACF4C1F60D8CB5C1F9DB846911EC".decode("hex"))
cyphered.append("DF774ABB054F12D4433322FFDF5D122F".decode("hex"))
cyphered.append("F98AC0A783A5DC3D3A4C5A54320C02FD".decode("hex"))
cyphered.append("CDC4101626FE13CA1606C03EB83DCB3D".decode("hex"))
cyphered.append("9045EABC2B8C3B608E270653076912B3".decode("hex"))
```

```
cyphered.append("EE234B5FA10D3944B7A35E7E84FA0FEC".decode("hex"))
cyphered.append("CB7D6A691F941A86CB715B4F03FC2202".decode("hex"))
cyphered.append("FC8B3E0210CCB974128A92DF746B9529".decode("hex"))
cyphered.append("FC7E72127701F6C3E264D094EC16632C".decode("hex"))
cyphered.append("A30F49DF6DD1BC992D42082E8AD79E9A".decode("hex"))
cyphered.append("48DA0A20817FD75A8A121AB126138E9D".decode("hex"))
cyphered.append("F4212EC1CE368350D85FB7091D53D61E".decode("hex"))
cyphered.append("FF2ADEE6B06319864F705A7792539931".decode("hex"))
cyphered.append("DE6BFC90AE5118EF06A11198513466BE".decode("hex"))
cyphered.append("D4FD68518A56D5E9BB7B1A3F48B41CEC".decode("hex"))
cyphered.append("24972F4C1738475B8580F8AE02F6FB87".decode("hex"))
cyphered.append("9119E1EBF6AF543C11CB3D539CFCB8C9".decode("hex"))

#
# Funcoes de Biblioteca
#
# Algoritmo para gerar a chave citado nas mensagens
def makeKey128(key):
    bytechave = [None]*16
    for i in range(0, 16):
        bytechave[i] = key % 256
        key = key // 256
    return bytechave

# Versao do gcd/mdc modificada entre "a" e "b"
def egcd(a, b):
    if a == 0:
        return (b, 0, 1)
    else:
        g, y, x = egcd(b % a, a)
        return (g, x - (b // a) * y, y)

# Calcula o inverso multiplicativo de "a" em modulo "m"
def modinverse(a, m):
    g, x, y = egcd(a, m)
    if g != 1:
        raise Exception('modular inverse does not exist')
    else:
        return x % m

#
# Inicio do Ataque
#
g_inv = modinverse(g, n)

K = (X * Y * g_inv) % n

print("Key:")
key = makeKey128(K)
print(key)

strkey = "".join(chr(x) for x in key) #key eh uma lista de inteiros,
#converte cada int em char e depois da um join nessa lista de string com a string ""

print("Texto:")
cipher = AES.new(strkey, AES.MODE_ECB)
for block in cyphered:
    plain = cipher.decrypt(block) #decifra bloco
    print(plain)
```