מבוא לקריפטוגרפיה / תרגיל בית 3#

אריאל סטולרמן
ודים סטוטלנד

(1)

(a)

כדי למצוא את מפתח ה-EKE נפעל לפי האלגוריתם הבא: נבצע את התהליך הבא m פעמים:

- נקבל מאליס g^{a_i}; נזרוק אותו ונעביר לבוב $g^{a_i'}$ עבור a' ראנדומלי שאנחנו בוחרים.
- בוב מגריל b_i ונקבל ממנו את $E_p(g^b)$. בוב מחשב את $k_i = g^{a_i' b_i}$
- נקבל מבוב את $E_{k_i}(z_{b_i})$, נשמור אצלנו ונעצור את התקשורת כך שאליס תיזום את התקשורת שוב.

עבור הנתונים שברשותינו נעבור על כל המפתחות $p_j \in D$ ונחשב:

- עבור כל מפתח נחשב לכל לכל $1 \le i \le m$:
  - נפענח: $D_{p_j}\left(E_p(g^{b_i})\right)$ ונקבל מועמד עבור g^{b_i}, נסמנו $g^{b_i'}$.
  - נחשב מועמד למפתח $k' = g^{a_i' b_i'}$ ונפענח: $D_{k'}\left(E_{k_i}(z_{b_i})\right)$, ונקבל מועמד עבור z_{b_i}.
- אם כל $pref(z_{b_i}) = 'b'$ לכל $1 \le i \le m$, נחזיר את p_j, אחרת נבדוק את הבא.

תחת הנחה שתו בודד בתחילת בלוק תופס 8 ביטים, כיוון שבכל שלב z_{b_i} נבחרים ראנדומלית, הסיכוי שנמצא מפתח שמקיים את הדרישה והוא אינו המפתח הנכון שווה לסיכוי ש-m מחרוזות אקראיות (שכן כל תהליך הפענוח מתבסס על פוני הצפנה וערכים אקראיים) יתחילו בתו $'b'$, וסיכוי זה הוא $\frac{1}{2^{8m}}$. עבור $m = 16$ למשל, כלומר 16 ניסיונות תקשורת מצד אליס, הסיכוי לטעות הוא $\frac{1}{2^{128}}$, וסיכוי זה זניח. כיוון שמרחב המפתחות האפשריים p הוא 2^{30}, וב-worst case נעבור על כולם, סה"כ נבצע $2^{30} \cdot 16 = 2^{34}$ פעולות.

(b)

כדי למצוא את מפתח ה-EKE במקרה זה נפעל לפי האלגוריתם הבא: נבצע m פעמים את התהליך הבא:

- מקבלים מאליס $E_p(g^{a_i})$.
- מגרילים b_i'.
- מגרילים p_j ומפענחים: $g^{a_i'} = D_{p_j}\left(E_p(g^{a_i})\right)$ ומקבלים מועמד עבור g^{a_i}.
- מגרילים $z_{b_i'}$ בלוק המתחיל ב-$'b'$, מצפינים: $E_{g^{a_i' b_i'}}\left(z_{b_i'}\right)$ ושולחים לאליס.
- אם קיבלנו מאליס תשובה, נשמור את התשובה שלה $E_{g^{a_i b_i'}}\left(z_{a_i}, z_{b_i''}\right)$ כאשר $z_{b_i''}$ מתחיל ב-$'b'$.
- אם לא קיבלנו תשובה, חוזרים על התהליך (אליס שולחת מפתח חדש).

לאחר שקיבלנו m הצפנות מהצורה $E_{g^{a_i b_i'}}\left(z_{a_i}, z_{b_i''}\right)$ נעצור ונעבור לפענוח אופליין: עבור על המפתחות $p_j \in D$ שכל אחד מהם קובע את g^{a_i} (ע"י פענוח של $E_p(g^{a_i})$) נפענח את $E_{g^{a_i b_i'}}\left(z_{a_i}, z_{b_i''}\right)$ עם המועמד עבור g^{a_i}. אם בכל ה-m קיבלנו $prefix\left(z_{b_i''}\right) = 'b'$ ו-$prefix(z_{a_i}) = 'a'$ בהתאמה, וסיכוי זה בהנחה שגודל תו הוא 8 ביטים הוא $\frac{1}{2^{16m}}$, ועבור $m = 4$ נקבל סיכוי טעות קטן של $\frac{1}{2^{64}}$. בכל שלב מ-m השלבים, הסיכוי שנשלח לאליס בלוק $E_{g^{a_i' b_i'}}\left(z_{b_i'}\right)$ שיפוענח על ידה עם המפתח $g^{a_i b_i'}$ לבלוק המתחיל ב-$'b'$, שווה לסיכוי של בחירה ראנדומלית של z_b המתחיל ב-$'b'$, וסיכוי זה הוא $\frac{1}{256} = \frac{1}{2^8}$. מספר הפעמים שנצטרך לבצע זאת עד שזה יקרה הוא בתוחלת 256 פעמים. סה"כ נזדקק ל-$256 \cdot 4 = 1024$ יצירות קשר, שמתוכן 1020 יסתיימו בניתוק מצד אליס, וזהו מספר מתקבל הדעת.

(c)

הרעיון של אליס ובוב טפשי שכן כך ניתן לשחזר (עם הסתברות נמוכה לטעות) את המפתח p באופן הבא: יוצרים תקשורת עם אליס m פעמים ונשמור את $E_p(g^a)$ שקיבלנו. נעבור על כל ה-p האפשריים (2^{30}) ונפענח את כל ההצפנות שברשותינו תחת המפתח הנוכחי; אם כל הפענוחים **אינם** שאריות ריבועיות עבור p כלשהו שבודקים, נצהיר עליו כמפתח הסודי. אם נשתמש גם ב-$E_p(g^b)$, נצמצם פי שניים את מספר הניתוקים כלפי כל צד.

בפרוטוקול הנוכחי a הוא אי זוגי ($lsb = 1$) ולכן g^a אינו שארית ריבועית, והרי חצי ממרחב המפתחות הוא שאריות ריבועיות. לכן עבור כל הצפנה שברשותינו מ-m ההצפנות שקיבלנו נוכל לפסול מחצית מהמפתחות. כך, הסיכוי הכולל לטעות יהיה (בדומה לסעיפים קודמים) $\frac{1}{2^m}$, ועבור $m = 128$ (או $m = 64$ כלפי כל אחד מהצדדים) נקבל סיכוי טעות גילוי p קטן מאוד: $\frac{1}{2^{128}}$.

(2)

```
# initialize p, q
##################
tmp = next_prime(randint(10^72,10^82))
p = 2*tmp + 1
tmp1082 = 10^82
while (not(is_prime(p)) or (p <= tmp1082)):
    tmp = next_prime(tmp)
    p = 2*tmp + 1

tmp = next_prime(randint(10^70,10^80))
q = 2*tmp + 1
tmp1077 = 10^77
while (not(is_prime(q)) or (q <= tmp1077)):
    tmp = next_prime(tmp)
    q = 2*tmp + 1

# calculate N, e and d
######################

N = p*q

# pick e at random so that e an phi_N are relatively prime
phi_N = (p-1)*(q-1)
e = randint(10^6,10^10)
while (not(gcd(e,phi_N) == 1)):
    e = randint(10^6,10^10)

# calculate d such that e*d = 1 (mod phi_N)
a,d,b = xgcd(e, phi_N)
d = mod(d,phi_N)        # normalization of d

# print variables
##################
print "N is:",N
print "10^82 is:",tmp1082
print "p is:",p
print "10^77 is:",tmp1077
print "q is:",q
print "e is:",e
print "d is:",d

OUTPUT:
=======
> N is: 338056257403431546310400256155630842571654409766513824002177330010203325\
  095560932154213198123306486274078389776263258773774613758705258366163648\
  6201028910945705249
> 10^82 is:
  100000000000000000000000000000000000000000000000000000000000000000000000\
  00000000000
> p is:
  197586277204891969530905686466522103166968687159138296367802126706359624\
  86121694303
> 10^77 is:
  100000000000000000000000000000000000000000000000000000000000000000000000\
  000000
> q is:
  171092983878063435505412728397963047421660032180701257026019075759213484\
  914822783
```

```
> e is: 681688023
> d is:
  237865131074949404279850288259358719161071679178248547928109306973422179\
  224603552631260567816647731628940667011050271884857388398162394677492941\
  8654726570340731599
```

(a)

בחירת p נעשתה כך: הגרלנו מספר בטווח $10^{72}, 10^{82}$ (פשוט שיהיה גדול מ-10^{72}) ולקחנו את הראשוני הראשון אחריו. הגדרנו את p להיות אותו מספר בכפולה של $2 + 1$, ובדקנו האם p ראשוני וגדול מ-10^{82}. המשכנו כך בלולאה עד שמצאנו p העונה על התנאים. עבור q בצענו את אותו מהלך עם הנתונים המתאימים לו (הגרלת המספר הראשון היתה בטווח $10^{80}, 10^{70}$ ו-q צריך להיות גדול מ-10^{77}).

(b)

```
# translation
##############
def get_int_from_char(c):
    if (c == " "): return 0
    if (c == "A" or c == "a"): return 1
    ...
    if (c == "Z" or c == "z"): return 26

def get_char_from_int(n):
    chars = list(" ABCDEFGHIJKLMNOPQRSTUVWXYZ")
    return chars[n]

# create plaintext and encrypt it
plaintext = list("KALEV ALPERNAS AND KIRIL SOLOVEY ARE SCUMBAGS")

ascii_text = 0
for i in range(len(plaintext)):
    ascii_text = ascii_text*100
    ascii_text = ascii_text + get_int_from_char(plaintext[i])

ciphertext = mod(ascii_text,N)^e

# decrypt the ciphertext
dec_ascii_text = int(mod(ciphertext,N)^d)              # already mod N, can be replaced with ciphertext^d

tmp = dec_ascii_text
dec_plaintext = list()
while (tmp > 0):
    r = int(mod(tmp,100))
    dec_plaintext = list(get_char_from_int(r)) + dec_plaintext
    tmp = int(tmp/100)

# prints
########
print "plaintext is:",plaintext
print "ascii_text is:",ascii_text
print "ciphertext is:",ciphertext
print "dec_ascii_text is:",dec_ascii_text
print "dec_plaintext is:",dec_plaintext

OUTPUT:
=======
plaintext is: ['K', 'A', 'L', 'E', 'V', ' ', 'A', 'L', 'P', 'E', 'R',
'N', 'A', 'S', ' ', 'A', 'N', 'D', ' ', 'K', 'I', 'R', 'I', 'L', ' ',
'S', 'O', 'L', 'O', 'V', 'E', 'Y', ' ', 'A', 'R', 'E', ' ', 'S', 'C',
'U', 'M', 'B', 'A', 'G', 'S']
ascii_text is:
110112052200011216051814011900011404001109180912001915121522052500011805\
001903211302010719
ciphertext is:
178441152568455877275290453692582937299174164587058861633743528685474339\
052737225811067349187842411928724687265336242558271594374654834561799606\
7074351689333458311
dec_ascii_text is:
110112052200011216051814011900011404001109180912001915121522052500011805\
001903211302010719
dec_plaintext is: ['K', 'A', 'L', 'E', 'V', ' ', 'A', 'L', 'P', 'E',
'R', 'N', 'A', 'S', ' ', 'A', 'N', 'D', ' ', 'K', 'I', 'R', 'I', 'L', '
', 'S', 'O', 'L', 'O', 'V', 'E', 'Y', ' ', 'A', 'R', 'E', ' ', 'S', 'C',
'U', 'M', 'B', 'A', 'G', 'S']
```

(c) הצטוותנו עם הדר וייס ויוסי שאשו.

```
# teaming up with another team: Hadar Weiss and Yossi Shasho
############################################################

# giving them our encrypted message:
enc_message = mod(ascii_text,their_N)^their_e
# print our sent message encrypted
print "our sent encrypted message is:",enc_message

# their N is:
their_N=150547124280806773146799617088635465589548929623654296596644010161018014320523689210022682633722762034206043144667061759897514665622725244181856064000679157725570
# their e is:
their_e=667

# decrypting their message:
their_enc_msg=17933471622709900999111345440726290534647648313192400668365281096396967498095809584493803794039147228884408596095260683725361428881958091360282231040383464259601 3

their_msg_ascii = int(mod(their_enc_msg,N)^d)
print "their_msg_ascii is:",their_msg_ascii

tmp = their_msg_ascii
their_plaintext = list()
while (tmp > 0):
    r = int(mod(tmp,100))
    their_plaintext = list(get_char_from_int(r)) + their_plaintext
    tmp = int(tmp/100)
# print their message
print "their message is:",their_plaintext

OUTPUT:
=======
our sent encrypted message is:
260125953903577710824840160220059717653874321426368621324367976811075725\
870032161796395859361859836978747697087775776304992964881925012616572305\
8707738127251669
their_msg_ascii is: 230805180500091900132500130914 04
their message is: ['W', 'H', 'E', 'R', 'E', ' ', 'I', 'S', ' ', 'M',
'Y', ' ', 'M', 'I', 'N', 'D']
```

ההודעה המפוענחת שלהם היא: WHERE IS MY MIND

(d)

```
# CRT test
##########
p = next_prime(10^300)
print "p is",p
q = next_prime(10^200)
print "q is",q
N = p*q
phi_N = (p-1)*(q-1)
e = randint(10^6,10^10)
print "e is",e
a,d,b = xgcd(e, phi_N)
d = mod(d,phi_N)
print "d is",d
ascii_text=110112052200011216051814011900011404001109180912001915121522052500011805\
001903211302010719
ciphertext = int(mod(ascii_text,N)^e)

# straight forward decryption:
timeit('mod(c,N)^d')

# the CRT speed-up:
timeit('\
c_mod_d_p = int(mod(ciphertext,p)**d);\
c_mod_d_q = int(mod(ciphertext,q)**d);\
CRT(c_mod_d_p,c_mod_d_q,p,q)')

# check outputs
print mod(ciphertext,N)^d
```

```
c_mod_d_p = int(mod(ciphertext,p)**d)
c_mod_d_q = int(mod(ciphertext,q)**d)
print mod(CRT(c_mod_d_p,c_mod_d_q,p,q),N)

OUTPUT:
=======
p is
10000000000000000000000000000000000000000000000000000000000000000000000000000000\
00000000000000000000000000000000000000000000000000000000000000000000000000000000\
00000000000000000000000000000000000000000000000000000000000000000000000000000000\
00000000000000000000000000000000000000000000000000000000000000000000000000000000\
0000000000331
q is
10000000000000000000000000000000000000000000000000000000000000000000000000000000\
00000000000000000000000000000000000000000000000000000000000000000000000000000000\
000000000000000000000000000000000000000000000000000000000000000357
e is 4401123674
d is
78454817286736396310593656814380172312331162180379119243991506156434357 9\
04390032371537487496653337626703557160707045379884046403191350082474414 9\
19156848033623333257869272046273335421839363653383215794910220278440646 2\
46652145317559644655420787432296091391318625326137563113614970866187933 4\
40926977257659650743320602264902406375776842121069647542015425763288832 3\
14690368776035444806271081397473130858444492782503889255623767322472201 9\
02499865992177524071094758333755471739556483092822080963862502993513
125 loops, best of 3: 4.17 ms per loop
125 loops, best of 3: 2.57 ms per loop
12124664039697994884965644891850772090530478857952273233966586759623568 1\
92711393639950785107979068445506127576824727080803120588889767583934919 8\
34583957686335691336247892390896961
12124664039697994884965644891850772090530478857952273233966586759623568 1\
92711393639950785107979068445506127576824727080803120588889767583934919 8\
34583957686335691336247892390896961
```

ניתן לראות כי אכן יש שיפור אך הוא יותר קרוב לפקטור 2 מאשר לפקטור 4.

(3)

נניח כי היינו משתמשים בפונקציה $f(x) = x^2$ באלגוריתם פולארד, ונסמן את פלטי הפונקציה: $y_1 = y_0^2, y_2 = y_1^2 = y_0^4, \dots, y_k = y_0^{2^k}$, ונסמן את הגורם אותו מחפשים ב-p וננית כי $p = 2q + 1$ כאשר q גורם ראשוני גדול. אם y_0 הוא שורש פרמיטיבי של $\mathbb{Z}_p^*$ אז מתקיים: $y_0^{2^i} \equiv y_0^{2^j} (\mathrm{mod}\, p) \Leftrightarrow 2^i \equiv 2^j (\mathrm{mod}\, p-1)$. מכאן מתקיים: $2^i \equiv 2^j (\mathrm{mod}\, 2q) \Rightarrow 2^{i-1} \equiv 2^{j-1} (\mathrm{mod}\, q)$.

אנו יודעים שהסדר של 2 ב-$\mathbb{Z}_q^*$ הוא $q-1$ ולכן אורך המחזור באלגוריתם פולארד הוא $q \cong \frac{1}{2}p$, גדול בהרבה מהסיבוכיות הידועה $\sqrt{p}$. ניתן להסביר זאת גם אינטואיטיבית בכך שבחירת הפונ' f מבטלת את הראנדומליות הנדרשת לפרדוקס היומולדת עליו מתבססת הוכחת סיבוכיות אלג' פולארד.

(4)

(a)

(ii+i

```
# build B
B = []
for i in range(150):
    if is_prime(i):
        B = B + [i]
Blen = len(B)

# returns the powers vector for given y
def get_pow_vec(y):
    e = []
    facy = factor(y)
    j = 0
    for i in range(Blen):
        if (facy[j][0] == B[i]):
            e = e + [facy[j][1]]
            j = j + 1
            if (j == len(facy)):
                j = j - 1
        else:
            e = e + [0]
```

```
    return e

# returns true iff x is B-smooth
def is_Bsmooth(x):
    facx = factor(x)
    biggest_fac_x = facx[len(facx)-1][0]   # get x's biggest factor
    if biggest_fac_x > B[Blen-1]:
        return false
    else:
        return true

x = 1001       # bigger than 1000
mat = Matrix(Blen,Blen)

# build matrix of Blen rows, where each row is e(x+1)-e(x) (x, x+1 are B-smooth)
for i in range(Blen - 1):
    # get a B-smooth x
    while not(is_Bsmooth(x) and is_Bsmooth(x + 1)):
        x = x + 1
    # calculate power vectors for x, x+1
    ex = get_pow_vec(x)
    ex1 = get_pow_vec(x + 1)
    # calculate e(x+1) - e(x)
    e = []
    for j in range(Blen):
        e = e + [ex1[j] - ex[j]]
    evec = vector(e)
    # update matrix
    mat[i] = evec
    # increment x to get a different x for each for iteration
    x = x + randint(1000,10000)

# define last row of matrix to hold "log_2(2)"
last_row = [1]
for i in range(Blen - 1):
    last_row = last_row + [0]

last_row_vec = vector(last_row)
mat[Blen - 1] = last_row_vec

# solve equations
Y_list = []
for i in range(Blen - 1):
    Y_list = Y_list + [0]
Y_list = Y_list + [1]
Y = vector(Y_list)
print mat.solve_right(Y)

OUTPUT (in Q):
==============
(1, 9546/6023, 13985/6023, 16901/6023, 20832/6023, 22291/6023,
24621/6023, 25578/6023, 27233/6023, 29262/6023, 29829/6023, 31386/6023,
32278/6023, 32678/6023, 33457/6023, 34507/6023, 35434/6023, 1879/317,
36524/6023, 37068/6023, 37287/6023, 37983/6023, 38403/6023, 38999/6023,
39775/6023, 40110/6023, 40276/6023, 40707/6023, 40699/6023, 41083/6023,
42108/6023, 42269/6023, 42756/6023, 42857/6023, 43401/6023)

Qsol = mat.solve_right(Y)
m = Matrix(RR,1,Blen)
m[0] = Qsol
print m

OUTPUT (in R):
==============
[1.00000000000000 1.58492445625104 2.32193259173170 2.80607670596048
3.45874149095135 3.70097957828325 4.08782998505728 4.24672090320438
4.52150091316620 4.85837622447285 4.95251535779512 5.21102440644197
5.35912336045160 5.42553544745144 5.55487298688361 5.72920471525818
5.88311472688029 5.92744479495268 6.06408766395484 6.15440810227461
6.19076871990702 6.30632575128673 6.37605844263656 6.47501245226631
6.60385190104599 6.65947202390835 6.68703304001328 6.75859206375560
6.75726382201561 6.82101942553545 6.99120039847252 7.01793126348996
7.09878797941225 7.11555703137971 7.20587746969949]
```

iii) באמצעות פונקציית factor של sage מצאנו כי מתקיים:

$$\log_2 99999999 = \log_2(3^2 \cdot 11 \cdot 73 \cdot 101 \cdot 137) = 2\log_2 3 + \log_2 11 + \log_2 73 + \log_2 101 + \log_2 137 \cong$$

כעת נקח את ערכי ה-$\log_2 x$ הרלוונטיים שקיבלנו בסעיפים הקודמים, ונציב:

$$\cong \frac{1}{6023}(2 \cdot 9546 + 20832 + 37287 + 40110 + 42756) = 26.57761913$$

בחישוב ישיר של $\log_2 99999999$ נקבל את התוצאה 26.57542474, וכפי שניתן לראות ההפרש קטן מ-$3 \cdot 10^{-3}$.

(b) כעת נחזור על האלגוריתם עם $M = 10^6$ (נבחר את x להתחיל מ-1000001):

```
OUTPUT:
=======
(1, 2823043/1781141, 8271365/3562282, 10000603/3562282, 6161734/1781141,
13181993/3562282, 7280349/1781141, 15132315/3562282, 16114201/3562282,
8652757/1781141, 8824115/1781141, 18557565/3562282, 9542555/1781141,
9664941/1781141, 19787025/3562282, 20404467/3562282, 10477830/1781141,
10563492/1781141, 10804567/1781141, 21907133/3562282, 22049905/3562282,
11227925/1781141, 11354860/1781141, 23068403/3562282, 23510737/3562282,
11859213/1781141, 23819165/3562282, 24015037/3562282, 12055109/1781141,
24295387/3562282, 24895649/3562282, 25055033/3562282, 25285165/3562282,
25359689/3562282, 25716709/3562282)
```

כעת נחשב את $\log_2 99999999$ לפי הפירוק החדש:

```
h = mat.solve_right(Y)
z = 2*h[1]+h[4]+h[20]+h[25]+h[32]
print z + 0.0
```

$$2 \cdot \frac{2823043}{1781141} + \frac{6161734}{1781141} + \frac{22049905}{3562282} + \frac{11859213}{1781141} + \frac{25285165}{3562282} = 26.5754187905393$$

ניתן לראות כי הדיוק השתפר (כעת קטן מ-$6 \cdot 10^{-6}$), ואם נקבע ע"פ הנתונים נראה כי חסם תחתון לדיוק של התוצאה עבור M נתון הוא בערך $1 - \frac{1}{M}$.

(6)

להלן אלגוריתם המשתמש ב-A לפירוק $N = pq$:

מגרילים $a \in \mathbb{Z}_N^* = \mathbb{Z}_{pq}^*$ ומחשבים $a^2 (\mathrm{mod}\, pq)$. נסמן $b = A(a^2)$, וכיוון של-a^2 יש 4 שורשים מעל $\mathbb{Z}_{pq}^*$ נקבל בסיכוי $\frac{1}{2}$ $b \neq \pm a$. נשים לב כי מתקיים: $b^2 - a^2 \equiv 0 (\mathrm{mod}\, pq)$ ולכן $(b-a)(b+a) \equiv 0 (\mathrm{mod}\, pq)$. מכאן ש-$pq$ מחלק את $(b-a)(b+a)$. תחת הנחה שקיבלנו $b \neq \pm a$, נובע כי pq אינו מחלק את $(b-a)$, אחרת $b - a \equiv 0 (\mathrm{mod}\, pq)$ כלומר $b = a$, ומשיקול דומה pq אינו מחלק את $b + a$ אחרת $b = -a$. מכאן, p מחלק את $b - a$ ו-q מחלק את $b + a$ או הפוך. מכאן ש-$\gcd(b-a, N) = p$ ו-$\gcd(b+a, N) = q$ או הפוך.

ניתן לבצע את האלגוריתם הנ"ל מספר פעמים קבוע, כאשר כל פעם מקטינה את הסיכוי לטעות פי $\frac{1}{2}$ (). עבור 3 איטרציות, שזמן הריצה הוא $3t(n)$ ($O(t(n))$ כנדרש), נקבל סיכוי הצלחה של $1 - \frac{1}{2^3} = \frac{7}{8}$.

(7)

יישום חתימה דיגיטלית כמתואר בשאלה על הפרדיגמה של Diffie-Hellman להלן: תהי H פונ' hash כנדרש, ו-N, d, e מפתחות RSA (כאשר N, e פומביים ו-d המפתח הפרטי). כדי לייצר חתימה על הודעה m, נגריל $k \in \{0,1\}^{128}$ ונשלח את השלישיה $\left(m, AES_k\left(H(m)\right), k^d\right)$. זיוף החתימה לאחר צפיה בהודעה אחת m שנשלחה ע"י בוב לכל הודעה חדשה שהיא m' פשוטה: בהינתן $\left(m, AES_k\left(H(m)\right), k^d\right)$ נחשב את $(k^d)^e = k$, ועבור הודעה m' נייצר את $\left(m', AES_k\left(H(m')\right), k^d\right)$ כאשר k^d פשוט נתון לנו מבוב – לא היה צורך כלל למצוא את d במפורש.