European Journal of Sustainable Development (2014), **3**, 2, 73-88 ISSN: 2239-5938
Doi: 10.14207/ejsd.2014.v3n2p73

Some Cryptographic Weakness and the Ways to Avoid Them in ICT Services of Developing Countries.

By
Dolantina HYKA[1] and Ardi BENUSI[2],

Abstract: One of the best ways to learn cryptography is to exercise their skills on encrypting and decrypting algorithms through different examples on corresponding cryptosystems (RSA [5], Massey-Omura [8], EMO1 [6], EMO2 [6],etc). Everyone have to try solving them by doing computations. Sometimes even following a wrong method one may get the right answer.
Following the idea presented in [1], [2], [3], we have presented here a maple algorithm to generate strong or diagnostic[7] cryptographic examples over Finite Field and Elliptic Curves.
The Maple algorithm presented, generate diagnostic exercises, in which all error paths lead to different answers. Using it one can easily generate exercises that are sound so a student who makes errors in algorithm steps computing will get the wrong answer.
This conclusion can also be used even in constructing algorithms not only in cryptography but also in other disciplines for example generating matrix different types matrix equations and being sure about the number of solutions without solving them first, or generating economic examples ect…

Keywords: ICT, strong example, sound example, diagnostic example, EMO1, EMO2, Fiat – Shamir, ect...

1. Introduction

Information Comunication Technology has advanced rapidly in the recent years, transforming not only the lyfe style of the human society but also economical and financial relations. Parallell to the benefits provided by the expansion of ICT, multiple risks related to security of this systems have been raised. The security issue has atracted important investments in industrialized countries, meanwhile in less developed countries these investments have been more modest, although the threats are presented uniformally in the globalized network. Our study concerned some services of e-banking

[1]Faculty of Natural Sciences, University of Tirana, ALBANIA.
[2]Faculty of Natural Sciences, University of Tirana, ALBANIA.

and e-commerce in the Balkan Region and in the Southern Mediterranean Region. It is very important that corporates that apply Information Computer Technology tools to have the ability to implement some software in PC, telephones, credit / debit cards, and in many electronic objects which can be used as surveillance equipment or objects on which depends the personal security (privacy), many business security or banks to the security of the state, etc...

In this paper are presented some algorithms examples that generate strong or diagnostic cryptographic examples. Initially are presented the analytical algorithms for encryption and decoding protocols reviewed and cryptosystems, mistakes that can be made during the generation of examples and ways to avoid each of these mistakes. Once submitted the problem and ways are needed to build the examples that are avoiding these mistakes, some algorithms appear in Maple, Java or C + + for each case. Through the execution of these algorithms, one can generate strong examples or diagnostic examples depending on the program algorithm build.

The goal of these paper is to generate not only strong examples or diagnostic examples for students but also to generate cryptosystems and protocols parameters such that the cryptosystem or the protocoll will be more secure.So the Maple and JAVA algorithms presented here, generates parameters for a secure cryptosystem and protocol.

2. Massey – Omura Cryptosystem

Encoding And Decoding Algorithm [5]

- We suppose that everyone has agreed upon a finite field Z_p which is fixed and publicly known.
- Each user of the system secretly selects a random integer u, r between 0 and p - 1 such that g.c.d. (u, p - 1) = 1 and (r, p-1) =1.
- Using the Euclidean algorithm, computes its inverse $v=u^{-1}\text{mod}(p-1)$ and $s=r^{-1}\text{mod}(p-1)$
- If user A (Alice) wants to send a message M to user B (Bob), first she need to convert the alphanumeric message to numeric message M, then computes $M_1=M^u \bmod p$ and sends it to Bob
- Without attempting to make sense of it, he raises it to his r and sends $M_2 = M_1^{\,r} \bmod p = M^{ur} \bmod p$ back to Alice.
- The third step is for Alice to unravel the message part of the way by rising to the v-th power $M_3 = M_2^{\,v} \bmod p = M^{urv} \bmod p = M^{r} \bmod p$ sending it back again to Bob.
- He can read the message just by raising this to the s-th power $M_4 = M_3^{\,s} \bmod p = M^{rs} \bmod p = M$.

Every cryptosystsem example has to be generatet in such way that the student can not arrive at the solution without passing throwght all correct algorithm steps.

To make it possible (to build strong cryptographic examples or diagnostic samples) during encryption should be noted that:

I) Encryption algorithm is performed over a finite field of characteristic p relatively large

II) As private keys should be chosen:

1. (u,p-1)=1 so the key *u* should be respectively prime to p-1 because of the chosen value key u there exist the inverse element v.
2. u≠v If the encrypting key u will be equal to the key v there is the possibility that the algorithm will be executed even if one will not find the inverse $v=u^{-1} \bmod (p-1)$ and the result can be correct. In these way we cannot be sure if there are followed all the right steps of the algorithm.
3. u≠r encryption keys of communicating parties must be different because otherwise the decoding keys would be the same as the computation are made in the same area. Random selection of the same keys increases the chance of transmitting the plaintext rather than encrypted one. If the receiver incorrectly encrypts the message with his key s instead of r then he transmits the plaintext instead of the code. With this condition is also avoided the situation where performs actions by sender or recipient only.
4. u≠s reasoning similar to the case of point 3.
5. (r,p-1)=1 r – key must also be respectively prime to φ(p) so one can compute the inverse s.
6. r≠s reasoning is similar to the case of point 2.
7. s≠v reasoning is similar to the case of point 3 too.
8. p≠1 because if p=1 we transmit the plain text
9. p≠p-1 because of the Fermat Theorem, we lose the plain text, It will be transmitted a code composed only by 1,1,1…

III) Mistakes commonly made during calculating the keys

0. p inverse is used instead of u inverse;
1. p inverse is used instead of v inverse;
2. p-1 inverse is used instead of u inverse;
3. p-1 inverse is used instead of v inverse;
4. Computations for finding the inverse element must be performed by module p – 1 and not module p.

2.1. Some Mistakes Commonly Made During the Algorithm Steps Computations are:

(a) Computations for M_1 , M_2, M_3, M_4 mod (p – 1) instead of mod p;
(b) One can get the inverses v , s mod p instead of mod p – 1;
(c) One can get the opposite of u , r instead of the inverse v, s respectively;
(d) The field characteristic can be confused with the private key during the coresponding inverse evaluations (ie instead found v mod (p-1) [s mod (p-1)], one can evaluate $(p-1)^{-1}$ mod u $[(p-1)^{-1} \bmod r]$;
(e) Is used u instead of v (s instead of r);
The following examples, are examples that are generated from maple and Java algorithms that avoids the errors listed above in Massey – Omura cryptosystem.

Example 1: Computations are made over a field by characteristic p=2464621 which, before displaying, has passed in all control steps. After that generate the privat key u=1361143 and r=1105913 that fulfill all the conditions set out above. One time generated the keys the system computes their inverse ensuring that calculations were made by p-1 module to avoid all errors listed above. When the algorithm have calcuated the inverse keys respectively, it compare them to avoid all the possible errors listed. If any error is presentet the algorithm indicates the type of the error and the way how to avoid it. So we obtain v=748867 and s=2165417. After that it begins to calculate the steps of encrypting and decrypting.
One firstly convert the plaintext into numeric plaintext so alfanumeric plaintext:
[DA,TA,_S,EC,UR,IT,Y_]
We obtain the numerical plaintext :
M=[3027,4627,7845,3129,4744,3546,5178]
0. At the firs step of the algorithm is calculated
$M_1=M^u \bmod p$ = [1447684, 2092759, 544105, 1184396, 2127421, 664622, 938321]
1. Then is calculated $M_2 = M_1^r \bmod p = M^{ur} \bmod p$
M2= [1672507, 1440192, 1486122, 1425848, 1664796, 2087290, 2261580]
2. The third step is for Alice to unravel the message part of the way by raising to the v-th power $M_3 = M_2^v \bmod p = M^{urv} \bmod p = M^r \bmod p$
M3= [1638815, 2056064, 1100039, 2159749, 1495619, 2014962, 2261580]
3. At the end the algorithm computes $M_4 = M_3^s \bmod p = M^{rs} \bmod p = M$
M4 = [3027, 4627, 7845, 3129, 4744, 3546, 5178]
Converting M4 back again in text message we obtain [DA,TA,_S,EC,UR,IT,Y_]
NOTE: We have divided the message in block of two letters for the parameters used in this example. If one need to encrypt three or more letters at the same time he may just increase the parameter p.)

Example 2: At the same way we created the example 1 we have:
The algorithm generates: p = 1617943, u = 222307, r = 487655
Computes the respectively inverse v = 514501, s = 1284941
Convert the message from text into numeric:
[PU,BL,IC,_K,EY,_S,YS,TE,M_] = [4247, 2838, 3529, 7837, 3151, 7845, 5145, 4631, 3978]
1) M1 = [995317, 1604512, 1054199, 1381626, 73343, 244070, 1554382, 456257, 1310889]
2) M2 = [662247, 262222, 705307, 1020310, 142724, 225411, 1386640, 75397, 16148]
3) M3 = [1127249, 977334, 80554, 133410, 1146582, 778, 949612, 192630, 481083]
4) M4 = [4247, 2838, 3529, 7837, 3151, 7845, 5145, 4631, 3978]

M4 = [PU, BL, IC, _K, EY, _S, YS, TE, M_]

Both of them are strong examples because to generate diagnostic examples, every two error algorithms have to be different from each other.
First of all, notice that it is absolutely necessary to use a good signature scheme along with the Massey-Omura cryptosystem. Otherwise, any eavedroper E (Evi) who is not supposed to know the message M could pretend to be Bob at the first step of the

cryptosystem algorithm. Not knowing that an intruder was using his own r', she would proceed to raise to the v and make it possible for Eve to read the message.
This is one of the reasons for requiring that communicating through chipertext throwght this kriptosistem become in real time. So both parties that participate have to be online during communication. Without this assumption, the scheme is not secure.
An other way to secure the comunication is that, the message $M_2 = M_1^{\,r} \bmod p = M^{ur} \bmod p$ from Bob to Alice must be accompanied by some authentification, i.e., some message in some signature scheme which only Bob could have sent.
To avoid interference of eavedroper Evi at the first step of communication between Alice and Bob, could also use an enhancement of Massey – Omura cryptosystem like EMO1 or EMO2. Theese one rather than an enhancement can be seen as a simultaneous action of EMO1 and RSA cryptosystem which may be presented in another paper. Basically there is a selection of a not prime public key for which is difficult to factorize in two prime numbers. And then are followed the Massey – Omura cryptosystem steps.

3. EMO-2 Cryptosystem

The EMO-2 cryptosystem is an enhancement of the EMO-1 cryptosystem described above. The enhancement is achieved by adding a second layer of encryption to each transmission, making cryptanalysis more difficult for those who would intercept the message without authorization. Furthermore, the system employs a digital signature which enables the recipient of a message to verify the identity of the sender, providing still another dimension of security.
The result is a partially private key, partially public key cryptosystem. More specifically, the EMO-2 cryptosystem is a combination of an EMO-1 private key system and an RSA public key system **[3,5,6].** The details of the system construction are described below. Establishing the Communication System. In order to establish an EMO-2 cryptosystem for a network of correspondents, the key center first performs the following functions.
For generating strong examples or diagnostically examples and exercises, we have created a maple algorithm that avoids generating the wrong keys or parameters for wich the cryptosystem is at risc from possible eavesdropper and also avoid coming at the wright answer following the wrong method for some mistakes that students usually do. In princip the algorithm avoid the steps described below. should be noted that remains all the cases reviewed in the algorithm of EMO1 cryptosystem adding the below conditions too.

A. Encrypting and Decrypting algorithm steps have to be evaluated on module n=pq. Where p and q are prime factors.
B. Private keys have to be chosen :
1. $(u, \varphi(n)) = 1$
2. u≠v
3. u≠r
4. u≠s

5. $(r,\varphi(n))=1$
6. r≠s
7. s≠v

For the same reason as in M-O

8. $v = u^{-1} \bmod \varphi(n) \neq u^{-1} \bmod (n-1)$
9. $s = r^{-1} \bmod \varphi(n) \neq r^{-1} \bmod (n-1)$

It is worth mentioning for EMO2 (8,9) is that one have to be careful that the value of v and s in module φ(n) should d not be the same as the value calculated by module (n-1) so cannot control when calculated φ(n)=φ(p)φ(q) but it is calculated at the same way as when n is a prime number violatin the cryptographic algorithm conditions.

10. $v = u^{-1} \bmod \varphi(n) \neq u^{-1} \bmod n$
11. $s = r^{-1} \bmod \varphi(n) \neq r^{-1} \bmod n$

Should also avoid case (10,11) when the computations to find the inverse of elements can be performed by the module n in stead of the module φ(n).

The following is an algorithm presented in Maple which avoids these errors in Massey – Omura enhancement cryptosystem (EMO1). This is an algorithm to build strong or diagnostic cryptographic examples too. Because some conditions are executed in the same step of the algorithm that does not tell us exactly which of the error message errors are avoided. If we just separate the "if conditions" adding them the corresponding error message than we can generate diagnostic examples that tells us even what type of error has occurred.

C. The public keys e, t, and private keys d,z have to be chosen in such way that:
1.(e,φ(n))=1 The private key e have to be respectively prime to φ(n) because of the chosen value key u exists the inverse element v.

2. d≠e If the public key would be the same as the private key d then there is the possibility that even without foundin the inverse element d, the cryptosystem algorithm executed and the output is correct. So if you can not control is implemented correctly or not the algorithm. **In this case we do not only generate diagnostic or strong examples but alse avoid choosing such public keys to generate a weak cryptosystem.**

3. e≠t public keys of communicating parties must be different because otherwise the decoding keys would be the same as the actions carried out over the same field. Random selection of the same keys increases the chance of transmitting plaintext instead of RSA encrypted. If the sender encrypts the message by mistake with hes private key z instead of the public key e then he transmits a text where the second encryption key can be "deactivated" instantly since the second key t which is the inverse of z is public in the system. This will turn EMO2 cryptosystem into EMO1 cryptosystem. **These is an other circumstance where we do not only generate diagnostic or strong examples but alse avoid choosing such public keys to generate a weak cryptosystem.**

4. d≠u otherwise e would be equal to v which directly affects on breaking EMO2 cryptosystem.

5.d≠v for the same reason as 4 because u and v are the inverse elements of each- other as long as the other key will then automatically recognized the inverse v well known that he is breaking the EMO2 and turning it in RSA .
6.d≠e because actions can be computed without the private key d.
The following is an algorithm presented in Maple which avoids these errors in Massey – Omura enhancement cryptosystem (EMO1). This is an algorithm to build strong or diagnostic cryptographic examplestoo. Because some conditions are executed in the same step of the algorithm that does not tell us exactly which of the error message errors are avoided. If we just separate the "if conditions" adding them the corresponding error message than we can generate diagnostic examples that tells us even what type of error has occurred.

```
> restart;
`crypt/alphabet` := `abcdefghijklmnopqrstuvwxyz`
||`ABCDEFGHIJKLMNOPQRSTUVWXYZ`
||```1234567890-=~!@#$£%^&*()_+`
||` ,./<>?;':"[]{}|    `:
> ne_numra := proc(st, string)
local ll, nn, ss, ii;
ll := length(st);
if ll = 0 then RETURN(0) fi;
nn := 1;
for ii to ll do
ss := SearchText(substring(st, ii .. ii),`crypt/alphabet`);
nn := 100*nn + ss
od;
nn - 10^(2*ll)
end:
> ne_germa := proc(nn, integer)
local ss, mm, ll, pp, ii, ans; mm := nn;
ll := floor(1/2*trunc(evalf(log10(mm))))+1;
ans := ``; for ii to ll do mm := mm/100;
pp := 100*frac(mm);
ss := substring(`crypt/alphabet`, pp..pp);
ans := cat(ss, ans); mm := trunc(mm)
od; ans end:
> with(numtheory): with(RandomTools):with(linalg):
> TekstiHapur:=Array([_PU,BLI,C_K,EY_]);
 TekstiHapur := [_PU, BLI, C_K, EY_]

> Mesazhi:=map(ne_numra,TekstiHapur);
 Mesazhi := [784247, 283835, 297837, 315178]

> m0:=ArrayNumElems(TekstiHapur);
 m0 := 4

> rg:=2^15:phi(rg);
 16384

> p:=Generate(posint(range=rg)): if (type(p,prime))  then  p  else Change the private key
end if;
```

```
 32611
> q:=Generate(posint(range=rg)): if (type(q,prime))  then  q  else Change the private key
end if;
 14563
> n:=p*q; #celesi privat
 n := 474913993
> n1:=phi(n);
 n1 := 474866820
>  u:=Generate(posint(range=n1));if  (gcd(u,n1)=1)  then  v:=modp((1/u),n1)  else
u1:=change the key_u end if;
 u := 188902883
 v := 240331847
>  e:=Generate(posint(range=n1));if  (gcd(e,n1)=1)  then  d:=modp((1/e),n1)  else
e1:=change the key_e end if;
 e := 158087113
 d := 351566977
>  if  (gcd(u,n1)<>1  or  gcd(u,n)<>1  or  gcd(u,n-1)<>1  or  v=modp((1/u),n-1)  or
v=modp((1/u),n) or u=v) then u1 end if;
>  if(gcd(e,n1)<>1  or  gcd(e,n)<>1  or  gcd(e,n-1)<>1  or  d=modp((1/e),n-1)  or
d=modp((1/e),n) or e=d) then e1 end if;
> r:=Generate(posint(range=n1)); if (gcd(r,n1)=1) then s:=modp((1/r),n1) else r1:=change
the key_r end if;
 r := 255394819
 s := 4498759
>  if  (gcd(u,n1)<>1  or  gcd(u,n)<>1  or  gcd(u,n-1)<>1  or  v=modp((1/u),n-1)  or
v=modp((1/u),n)  or  u=v  or  gcd(e,n1)<>1  or  gcd(e,n)<>1  or  gcd(e,n-1)<>1  or
d=modp((1/e),n-1) or d=modp((1/e),n) or e=d) then u1 end if;
> if (r=u or r=v) then r1 end if;
> if (gcd(r,n1)=1) then s:=modp((1/r),n1)  else r1 end if;
 s := 4498759
>  if  (gcd(r,n1)<>1  or  gcd(r,n)<>1  or  gcd(r,n-1)<>1   or  s=modp((1/r),n)  or
s=modp((1/r),n-1)) then r1 end if;
> if (s=u or s=v or s=r) then r1 end if;
> if (v=(1/u) mod p) then u1 end if;
> for i from 1 to m0 do
C1[i]:=Mesazhi[i]&^u mod n
end do:
> for i from 1 to m0 do
if (C1[i]= C1[i]&^u mod (n1))then u1 end if
end do;
> M1:=array([C1[1],C1[2],C1[3],C1[4]]);
```

M1 := [302312480, 215628639, 266550473, 235441710]

```
> for i from 1 to m0 do
C2[i]:=C1[i]&^r mod n
end do:
> for i from 1 to m0 do
if (C2[i]=C2[i]&^u mod n1)then u1 end if
end do;
> M2:=array([C2[1],C2[2],C2[3],C2[4]]);
```

M2 := [379741741, 177444087, 4533581, 143006112]

```
> for i from 1 to m0 do
C3[i]:=C2[i]&^v mod n
end do:
> for i from 1 to m0 do
if (C3[i]= C3[i]&^u mod (n1))then u1 end if
end do;
> M3:=array([C3[1],C3[2],C3[3],C3[4]]);
```

M3 := [37322677, 85188548, 1715573, 144852328]

```
> for i from 1 to m0 do
C4[i]:=C3[i]&^s mod n
end do:
> for i from 1 to m0 do
if (C4[i]= C4[i]&^u mod (n1))then u1 end if
end do;
> M4:=array([C4[1],C4[2],C4[3],C4[4]]);
```

M4 := [784247, 283835, 297837, 315178]

```
> M4_:=map(ne_germa,M4);
```

*M4_* := [*_PU, BLI, C_K, EY_*]

4. M-O IN ELLIPTIC CURVE CRYPTOGRAHY

• We suppose that everyone has agreed upon a finite field Zp which is fixed and publicly known.
• Each user of the system secretly selects a random integer u, r between 0 and p - 1 such that g.c.d. (u, p - 1) = 1 and (r, p-1) =1.
• Using the Euclidean algorithm, computes its inverse v=u-1mod(p-1) and s=r-1mod(p-1)
• If user A (Alice) wants to send a message M to user B (Bob), first she need to convert the alphanumeric message to an elliptic curve point M, then computes M1=uM mod p and sends it to Bob
• Without attempting to make sense of it, he multiplyes the point M1 to his key r and sends the message M2 to Alice.
• The third step is for Alice to unravel the message part of the way by multiplying with his other key v.
•He can read the message just by multiplying this by the key s back to Alice.

To make it possible (to build strong cryptographic examples or diagnostic samples) during encryption should be noted that:
I) Encryption algorithm is performed over a finite field of characteristic p relatively
II) As private keys should be chosen:
1. (u,p-1)=1 so the key u should be respectively prime to p-1 because of the chosen value key u exists the inverse element v.
2. u≠v If the encrypting key u will be equal to the key v there is the possibility that the algorithm will be executed even if one will not find the inverse v=u-1mod(p-1) and the result can be correct. In these way we cannot be sure if there are followed all the right steps of the algorithm.
3. u≠r encryption keys of communicating parties must be different because otherwise the decoding keys would be the same as the computation are made in the same area. Random selection of the same keys increases the chance of transmitting the plaintext rather than encrypted one. If the receiver incorrectly encrypts the message with his key s instead of r then he transmits the plaintext instead of the code. With this condition is also avoided thesituation where performs actions by seder or recipient only.
4. u≠s reasoning similar to the case of point 3.
5. (r,p-1)=1 r – key must also be respectively prime to so one can compute the inverse s.
6. r≠s reasoning is similar to the case of point 2.
7. s≠v reasoning is similar to the case of point 3 too.
8.Computations for finding the inverse element must be performed by module p – 1 and not module p.

2.2 Some of the mistakes commonly made during the algorithm steps computations are:
(a) Computations for M1;
(b) One can get the inverses v, s mod p instead of mod p – 1;
(c) One can get the opposite of u;
(d) The field characteristic can be confused with the private key during the corresponding inverse evaluations (i.e. instead found v mod (p-1) [s mod (p-1)], one can evaluate (p-1)-1 [(p-1)-1 mod r] ;
(e) Is used u instead of v (s instead of r);
III Other mistakes one can make are:

1. Computing M_i to power u,r,v,s respectively as a commune plain text in Massey Omura cryptosystem not as an elliptic curve.
2. M1,M2, M3, M4 mod (p – 1) instead of mod p;

The following is an algorithm presented in Maple which avoids these errors in Massey – Omura cryptosystem. This is an algorithm to build strong or diagnostic cryptographic examples. Because some conditions are executed in the same step of the algorithm that does not tell us exactly which of the error message errors are avoided. If we just separate the "if conditions" adding them the corresponding error message than we can generate diagnostic examples that tells us even what type of error has occurred.

```
a:=0:b:=1:c:=2: d:=3: e:=4: f:=5: g:=6: h:=7: i:=8: j:=9: k:=10: l:=11: m:=12: n:=13: o:=14: p:=15:
q:=16: r:=17: s:=18: t:=19: u:=20: v:=21: w:=22: x:=23: y:=24: z:=25:
 with(numtheory): with(RandomTools):with(linalg):
rg := 2^6;
p := Generate(posint(range = rg)); if `and`(type(p, prime), evalb(`mod`(p, 4) = 3) = true) then p else
Change the private key end if;
p := 31;
TekstiHapur := Array([s, o, t]);
x1 := TekstiHapur[1]*26^2+26*TekstiHapur[2]+TekstiHapur[3];
f := x^3+2*x-1;
`mod`((`mod`(x1^3+2*x1-1, p))^((p-1)*(1/2)), p);                1
u := Generate(posint(range = p-1)); u1 := change the key_u; u := 17;               17
if gcd(u, p-1) = 1 then v := modp(1/u, p-1) else u1 end if;                23
if `or`(`or`(`or`(gcd(u, p-1) <> 1, gcd(u, p) <> 1), v = modp(1/u, n-1)), u = v) then u1 end if;
r := Generate(posint(range = p-1)); r1 := change the key_r; r := 7;               7
if `or`(r = u, r = v) then r1 end if;
if gcd(r, p-1) = 1 then s := modp(1/r, p-1) else r1 end if;                13
if `or`(`or`(gcd(r, p-1) <> 1, gcd(r, p) <> 1), s = modp(1/r, p)) then r1 end if;
if `or`(`or`(s = u, s = v), s = r) then r1 end if;
if v = `mod`(1/u, p) then u1 end if;
convert(u, binary);                10001
convert(v, binary);                10111
convert(r, binary);                111
convert(s, binary);                1101
P0 := array([`mod`(x1, p), `mod`((x1^3+2*x1-1)^((p+1)*(1/4)), p)]);
P00 := array([`mod`(x1, p), `mod`(-(x1^3+2*x1-1)^((p+1)*(1/4)), p)]);
a0 := 1; k1 := `mod`((3*P0[1]^2+a0)*(2*P0[2])^(-1), p); k2 := `mod`((3*P0[1]^2+a0)*(2*P00[2])^(-
1), p);
x3 := `mod`(k1^2-2*P0[1], p); y3 := `mod`(-P0[2]+k1*(P0[1]-x3), p); y4 := `mod`(-
P00[2]+k2*(P0[1]-x3), p);
P0 := Array([x3, y3]); P00 := Array([x3, y4]);
a0 := 1; k1 := `mod`((3*P0[1]^2+a0)*(2*P0[2])^(-1), p); k2 := `mod`((3*P0[1]^2+a0)*(2*P00[2])^(-
1), p);
x3 := `mod`(k1^2-2*P0[1], p); y3 := `mod`(-P0[2]+k1*(P0[1]-x3), p); y4 := `mod`(-
P00[2]+k2*(P0[1]-x3), p);
P0 := Array([x3, y3]); P00 := Array([x3, y4]);
a0 := 1; k1 := `mod`((3*P0[1]^2+a0)*(2*P0[2])^(-1), p); k2 := `mod`((3*P0[1]^2+a0)*(2*P00[2])^(-
1), p);
x3 := `mod`(k1^2-2*P0[1], p); y3 := `mod`(-P0[2]+k1*(P0[1]-x3), p); y4 := `mod`(-
P00[2]+k2*(P0[1]-x3), p);
P0 := Array([x3, y3]); P00 := Array([x3, y4]);
a0 := 1; k1 := `mod`((3*P0[1]^2+a0)*(2*P0[2])^(-1), p); k2 := `mod`((3*P0[1]^2+a0)*(2*P00[2])^(-
1), p);
x3 := `mod`(k1^2-2*P0[1], p); y3 := `mod`(-P0[2]+k1*(P0[1]-x3), p); y4 := `mod`(-
P00[2]+k2*(P0[1]-x3), p);
P01 := Array([x3, y3]); P001 := Array([x3, y4]);
a1 := `mod`((P01[2]-P0[2])/(P01[1]-P0[1]), p); a11 := `mod`((P001[2]-P0[2])/(P01[1]-P0[1]), p);
x4 := `mod`(a1^2-P01[1]+P0[1], p); y5 := `mod`(-P01[2]+a1(P0[1]-x4), p); y6 := `mod`(-
P001[2]+a11(P0[1]-x4), p);
P1 := Array([x4, y5]); P11 := Array([x4, y6]);
              Array(%id = 4494486850)
              Array(%id = 4494486914)
k1 := `mod`((3*P1[1]^2+a0)*(2*P1[2])^(-1), p); k2 := `mod`((3*P1[1]^2+a0)*(2*P11[2])^(-1), p);
```

```
x3 := `mod`(k1^2-2*P1[1], p); y3 := `mod`(-P11[2]+k2*(P1[1]-x3), p); y4 := `mod`(-
P11[2]+k2*(P1[1]-x3), p);
P10 := Array([x3, y3]); P110 := Array([x3, y4]);
k1 := `mod`((3*P1[1]^2+a0)*(2*P1[2])^(-1), p); k2 := `mod`((3*P1[1]^2+a0)*(2*P11[2])^(-1), p);
x3 := `mod`(k1^2-2*P1[1], p); y3 := `mod`(-P11[2]+k2*(P1[1]-x3), p); y4 := `mod`(-
P11[2]+k2*(P1[1]-x3), p);
P10 := Array([x3, y3]); P110 := Array([x3, y4]);
a1 := `mod`((P10[2]-P1[2])/(P10[1]-P1[1]), p); a11 := `mod`((P110[2]-P11[2])/(P10[1]-P1[1]), p);
x4 := `mod`(a1^2-P10[1]+P1[1], p); y5 := `mod`(-P10[2]+a1(P1[1]-x4), p); y6 := `mod`(-
P110[2]+a11(P1[1]-x4), p);
P1 := Array([x4, y5]); P11 := Array([x4, y6]);
k1 := `mod`((3*P1[1]^2+a0)*(2*P1[2])^(-1), p); k2 := `mod`((3*P1[1]^2+a0)*(2*P11[2])^(-1), p);
x3 := `mod`(k1^2-2*P1[1], p); y3 := `mod`(-P11[2]+k2*(P1[1]-x3), p); y4 := `mod`(-
P11[2]+k2*(P1[1]-x3), p);
P10 := Array([x3, y3]); P110 := Array([x3, y4]);
a1 := `mod`((P10[2]-P1[2])/(P10[1]-P1[1]), p); a11 := `mod`((P110[2]-P11[2])/(P10[1]-P1[1]), p);
x4 := `mod`(a1^2-P10[1]+P1[1], p); y5 := `mod`(-P10[2]+a1(P1[1]-x4), p); y6 := `mod`(-
P110[2]+a11(P1[1]-x4), p);
P1 := Array([x4, y5]); P11 := Array([x4, y6]);
k1 := `mod`((3*P1[1]^2+a0)*(2*P1[2])^(-1), p); k2 := `mod`((3*P1[1]^2+a0)*(2*P11[2])^(-1), p);
x3 := `mod`(k1^2-2*P1[1], p); y3 := `mod`(-P11[2]+k2*(P1[1]-x3), p); y4 := `mod`(-
P11[2]+k2*(P1[1]-x3), p);
P10 := Array([x3, y3]); P110 := Array([x3, y4]);
a1 := `mod`((P10[2]-P1[2])/(P10[1]-P1[1]), p); a11 := `mod`((P110[2]-P11[2])/(P10[1]-P1[1]), p);
x4 := `mod`(a1^2-P10[1]+P1[1], p); y5 := `mod`(-P10[2]+a1(P1[1]-x4), p); y6 := `mod`(-
P110[2]+a11(P1[1]-x4), p);
P2 := Array([x4, y5]); P22 := Array([x4, y6]);
                Array(%id = 4494479554)
                Array(%id = 4494479618)
k1 := `mod`((3*P2[1]^2+a0)*(2*P2[2])^(-1), p); k2 := `mod`((3*P2[1]^2+a0)*(2*P22[2])^(-1), p);
x3 := `mod`(k1^2-2*P2[1], p); y3 := `mod`(-P22[2]+k2*(P2[1]-x3), p); y4 := `mod`(-
P22[2]+k2*(P2[1]-x3), p);
P20 := Array([x3, y3]); P220 := Array([x3, y4]);
a1 := `mod`((P20[2]-P2[2])/(P20[1]-P2[1]), p); a11 := `mod`((P220[2]-P22[2])/(P20[1]-P2[1]), p);
x4 := `mod`(a1^2-P20[1]+P2[1], p); y5 := `mod`(-P20[2]+a1(P2[1]-x4), p); y6 := `mod`(-
P220[2]+a11(P2[1]-x4), p);
P2 := Array([x4, y5]); P22 := Array([x4, y6]);
k1 := `mod`((3*P2[1]^2+a0)*(2*P2[2])^(-1), p); k2 := `mod`((3*P2[1]^2+a0)*(2*P22[2])^(-1), p);
x3 := `mod`(k1^2-2*P2[1], p); y3 := `mod`(-P22[2]+k2*(P2[1]-x3), p); y4 := `mod`(-
P22[2]+k2*(P2[1]-x3), p);
P20 := Array([x3, y3]); P220 := Array([x3, y4]);
a1 := `mod`((P20[2]-P2[2])/(P20[1]-P2[1]), p); a11 := `mod`((P220[2]-P22[2])/(P20[1]-P2[1]), p);
x4 := `mod`(a1^2-P20[1]+P2[1], p); y5 := `mod`(-P20[2]+a1(P2[1]-x4), p); y6 := `mod`(-
P220[2]+a11(P2[1]-x4), p);
P3 := Array([x4, y5]); P33 := Array([x4, y6]);
                Array(%id = 4494480066)
                Array(%id = 4494480130)
k1 := `mod`((3*P3[1]^2+a0)*(2*P3[2])^(-1), p); k2 := `mod`((3*P3[1]^2+a0)*(2*P33[2])^(-1), p);
x3 := `mod`(k1^2-2*P3[1], p); y3 := `mod`(-P33[2]+k2*(P3[1]-x3), p); y4 := `mod`(-
P33[2]+k2*(P3[1]-x3), p);
P30 := Array([x3, y3]); P330 := Array([x3, y4]);
a1 := `mod`((P30[2]-P3[2])/(P20[1]-P2[1]), p); a11 := `mod`((P330[2]-P33[2])/(P30[1]-P3[1]), p);
x4 := `mod`(a1^2-P30[1]+P3[1], p); y5 := `mod`(-P30[2]+a1(P3[1]-x4), p); y6 := `mod`(-
P330[2]+a11(P3[1]-x4), p);
P3 := Array([x4, y5]); P33 := Array([x4, y6]);
```

```
k1 := `mod`((3*P3[1]^2+a0)*(2*P3[2])^(-1), p); k2 := `mod`((3*P3[1]^2+a0)*(2*P33[2])^(-1), p);
x3 := `mod`(k1^2-2*P3[1], p); y3 := `mod`(-P33[2]+k2*(P3[1]-x3), p); y4 := `mod`(-P33[2]+k2*(P3[1]-x3), p);
P30 := Array([x3, y3]); P330 := Array([x3, y4]);
k1 := `mod`((3*P3[1]^2+a0)*(2*P3[2])^(-1), p); k2 := `mod`((3*P3[1]^2+a0)*(2*P33[2])^(-1), p);
x3 := `mod`(k1^2-2*P3[1], p); y3 := `mod`(-P33[2]+k2*(P3[1]-x3), p); y4 := `mod`(-P33[2]+k2*(P3[1]-x3), p);
P30 := Array([x3, y3]); P330 := Array([x3, y4]);
a1 := `mod`((P30[2]-P3[2])/(P20[1]-P2[1]), p); a11 := `mod`((P330[2]-P33[2])/(P30[1]-P3[1]), p);
x4 := `mod`(a1^2-P30[1]+P3[1], p); y5 := `mod`(-P30[2]+a1(P3[1]-x4), p); y6 := `mod`(-P330[2]+a11(P3[1]-x4), p);
P4 := Array([x4, y5]); P44 := Array([x4, y6]);
                Array(%id = 4494471298)
                Array(%id = 4494471362)
y8 = `mod`(P4[1], 26);                    y8 = 15
y9 := `mod`(P4[1]-y8, 26);                  15 + 25 y8
y10 := `mod`(`mod`((P4[1]-y8)*26^(-1), p)-y9, 26);                  13
```

5. RSA

5.1. Public Key Generation

Generate two prime intigers p and q. Compute $\varphi(n) = (p-1)(q-1)$.
Choose an integer e, $1 < e < \varphi$, such that gcd(e, φ) = 1.

Encoding Algorithm[5]

Given: modulus $n = pq$, public exponent $e \in \mathbb{Z}^*_{\varphi(n)}$,and message $M \in \mathbb{Z}_n$.

Find: cyphertext $C = M^e \bmod n$.

Method: find $C = M^e \bmod n$, preferably by fast modular exponentiation.

Decoding Algorithm[5]

Given: modulus $n = pq$, public exponent $e \in \mathbb{Z}^*_{\varphi(n)}$, and ciphertext $C = M^e \bmod n$.

Find: $M = C^d \bmod n$, where $d = e^{-1} \bmod \varphi(n)$.

Method: factorise n, to find p and q; calculate $\varphi(n) = (p-1)(q-1)$; find $d = e^{-1} \bmod \varphi(n)$ using the Extended Euklidian Algorithm; and find $M = C^d \bmod n$, preferably by fast modular exponentiation.

Errors that the Maple algoritme have to nutralize:

1. Compute $\varphi(n) = n - 1$ instead of $\varphi(n) = (p-1)(q-1)$
2. Chose the integer e, $1 < e < n$, such that gcd(e,n) = 1.[4]
3. use $\varphi(n)$ instead of n as the modulus for Message or Code exponentiation;
4. use $n - 1$ instead of n as the modulus for Message or Code exponentiation;
5. When representing a message as digits sometimes is taken A-Z=1-26 and other times A-Z=0-25 on representing the same message.
6. use $n - 1$ instead of $\varphi(n)$ as the modulus when finding e^{-1};
7. use n instead of $\varphi(n)$ as the modulus when finding e^{-1};
8. use e instead of d;[4]
9. $e \neq d$ one can compute the code C as $C=M^e \bmod n$ or can compute the message as $M=C^e \bmod n$
10. negate the inverse e^{-1};
11. swap the modulus and d when computing d^{-1}(so, with modulus $\varphi(n)$, the student finds $\varphi(n)^{-1} \bmod d$ instead of $d^{-1} \bmod \varphi(n)$;
12. use $\varphi(n)$ instead of n as the modulus for exponentiation;

13. use n - 1 instead of n as the modulus for exponentiation.
14. $e \neq \varphi(n)$ because prom Euler Theorem $M^{\varphi(n)} \equiv 1 \bmod n$ so we risk to transmit an 111… code.

The errors treated on Massey – Omura cryptosystems are not analytically presented but are declared in the Maple algorithm.

```
> restart;
`crypt/alphabet` := `abcdefghijklmnopqrstuvwxyz`
||`ABCDEFGHIJKLMNOPQRSTUVWXYZ`
||```1234567890-=~!@#$£%^&*()_+`
||` ,./<>?;':"[]{}|    `:
> ne_numra := proc(st, string)
local ll, nn, ss, ii;
ll := length(st);
if ll = 0 then RETURN(0) fi;
nn := 1;
for ii to ll do
ss := SearchText(substring(st, ii .. ii),`crypt/alphabet`);
nn := 100*nn + ss
od;
nn - 10^(2*ll)
end:
> ne_germa := proc(nn, integer)
local ss, mm, ll, pp, ii, ans; mm := nn;
ll := floor(1/2*trunc(evalf(log10(mm))))+1;
ans := ``; for ii to ll do mm := mm/100;
pp := 100*frac(mm);
ss := substring(`crypt/alphabet`, pp..pp);
ans := cat(ss, ans); mm := trunc(mm)
od; ans end:
> with(numtheory): with(RandomTools):with(linalg):
> rg:=2^15:phi(rg);
 16384

> p:=Generate(posint(range=rg)): if (type(p,prime))  then  p  else Change the private key
end if;
 32611

> q:=Generate(posint(range=rg)): if (type(p,prime))  then  q  else Change the private key
end if;
 7241

> n:=p*q; #celesi privat
 n := 236136251

> n1:=phi(n);
 n1 := 217573920

> e:=Generate(posint(range=n1));  e1:=change the key_e:
```

```
e := 118269623
> if (gcd(e,n1)=1) then d:=modp((1/e),n1) else e1 end if;
d := 107283047
> if (gcd(e,n1)<>1 or gcd(e,n)<>1 or gcd(e,n-1)<>1 or d=modp((1/e),n-1) ) then e1 end
if;
> if (d=modp((1/e),n) or e=d or (e,p)=1 or (e,q)=1
> if (gcd(n1,e)=1) then n2:=modp((1/e),d)  end if:
> TekstiHapur:=Array([ALG,EBR,A_A,ND_,GEO,MET,RY_]):
> Mesazhi:=map(ne_numra,TekstiHapur);
Mesazhi := [273833 , 312844 , 277827 , 403078 , 333141 , 393146 , 445178 ]
> m0:=ArrayNumElems(TekstiHapur):
> for i from 1 to m0 do
C1[i]:=Mesazhi[i]&^e mod n
end do:
> for i from 1 to m0 do
if (C1[i]= C1[i]&^d mod (n1) or C1[i]=C1[i]&^n2)then e1 end if
end do;
> C:=array([C1[1],C1[2],C1[3],C1[4],C1[5],C1[6],C1[7]]);
C := [51926720 , 162100431 , 143317886 , 127318009 , 11462824 , 221145951 , 51193036 ]
> for i from 1 to m0 do
C2[i]:=C1[i]&^d mod n
end do:
> for i from 1 to m0 do
if (C2[i]=C2[i]&^e mod n1)then u1 end if
end do;
> M:=array([C[1],C[2],C[3],C2[4],C2[5],C2[6],C2[7]]):
> M2_:=map(ne_germa,M2);
M2_ := [ALG, EBR, A_A, ND_, GEO, MET, RY_]
```

6. Conclusions:

To generate strong (diagnostic) examples and exercices, we have created Maple algorithms that avoid generating the wrong keys or parameters for wich the cryptosystem is at risc from possible eavesdropper and also avoid coming at the wright answer following the wrong method for some errors that students usually do. In princip the algorithms make possible to generate exercices who do not allow any of the cases described in each of cryptosystems and protocols presented above to be possible during the system steps calculation.

The meaning of this work is not only to generate strong or diagnostic examples and exercices but also to considerate possible parameter values that makes those cryptosystems more secure and more efficient.

It also can be used to define new hybrid algorithms as EMO1, EMO2 if we implement them over Elliptic Curves.

Following this idea it is possible to generate strong or diagnostic examples not only in public cryptography but alsow in symetric or quantum cryptography. This idea can be implementit in other disiplines too. For example in generating financial mathemati exercices or micro economy exercices. Etc...

References

[1] Hyka D. "An overview on diagnostically cryptographyc examples" IWBCMS – 2013
[2] Hyka D. & A. Baxhaku "Some maple algorithms generating diagnostically / strong cryptographic examples." ISTI – 2013
[3] Hyka D. & A. Baxhaku "Generating "sound" cryptographyc examples using different software." 5th International Scientific Conference "Economic Policy and EU Integration"
[4] Chong S.K., Farr G., Frost L., Hawley S.,"On pedagogically sound examples in public-key cryptography", ACSC 2006
[5] Chong, S. K. (2003),"Cryptographic teaching tools", BCompScHonours, Monash University, Clayton,Australia
[6] Winton R. "Enhancing the Massey-Omura Cryptosystem", Journal of Mathematical Sciences & Mathematics Education Vol. 7 No. 1
[7] Baxhaku A., Naka K. "Një problem gjeometrik i antikitetit në një protokoll me njohje zero dhe në zgjerimet algjebrike të fushave"
[8] Koblitz N. "A Course in Number Theory and Cryptography" Second Edition, Springer-Verlag ISBN 0-387-94293-9 (New York)