2023 암호분석경진대회

6번 문제

목차

- 답
- 풀이 과정
- 문제풀이 코드
- 실행 결과
- 참조
- 부록

* 문제풀이 코드는 6번 폴더 안에서도 확인할 수 있습니다.

*6번 문제 관련 파일 목록*

1. *6/params.sage: 문제에서 주어진 파라미터를 정의한 파일*
2. *6/solver.sage: Alice가 획득한 서명값 s를 이용한 개인키 복원 공격 코드*
3. *6/implementation.sage: 6번 문제에서 주어진 서명 알고리즘에 대한 sagemath 구현체*
4. *6/analysis.sage: 6/implementation.sage 구현체에 대한 공격 검증 코드*

[답]

주어진 전자서명 알고리즘은 생성된 서명값 중 $s$에서 개인키인 $n_S$, $\phi_S$를 복원할 수 있으며, 공격자는 복원한 서명을 통하여 임의의 메세지 $m$을 공격자 스스로 서명할 수 있음 ([풀이 과정] 섹션 참고).

문제에서 복원된 개인키값은 아래와 같음.

$n_S$ = 0x5a0c7dc0eb986a066077645b72cd4b57f9aa64608bab7ecccffc64

$\phi_S(x,y) =$

(x^2+(153266247282646412875433016919523088686109405271846163667339172350781821966953215675267029214576725168855347207390688169220953720 64*i+136240907267658649873872012304596178230435707363083510945366189975897429519523024523546250263801876326559457191549441058938831096 00)*x+62534538838824290240502898366722311701923795819689919765018336424979583799107878513713748803610114690599634877570058777203554538 11*i+17907893919861322037110961181125753975264526601727213882463495496843957056611966950205677177332292602100973478698379568613088654 836)/(x+1532662472826464128754330169195230886861094052718461636673391723507818219669532156752670292145767251688553472073906881692209537 2064*i+1362409072676586498738720123045961782304357073630835109453661899758974295195230245235462502638018763265594571915494410589388310 9600,

x^2*y+(6213825795184061023177458372447124118136100810607636222142027133951143154058666409083189171086726587872350414785252694501876010561*i+2808757792186508422865257449461742027001361228855105677747430658974264664572628178739033380931756819413172411617003272445451485633)*x*y+(9357900897491556725182957486612500241185691063258918478120695037606841510112503461639541512798343862456298927384199135076167141569*i+24340894229446221712949840632696395191927720165598164939607522203914819053289507720920763245846309810513081168969373544719060517339)*y)/(x^2+(6213825795184061023177458372447124118136100810607636222142027133951143154058666409083189171086726587872350414785252694501876010561*i+2808757792186508422865257449461742027001361228855105677747430658974264664572628178739033380931756819413172411617003272445451485633)*x+156113547813739857492332473232847314113780706452279104546225286801047998900232913130109163931593553315162624151412050127965225953 80*i+178093644879623221981516568023646555481064665235637823107452103645535548705694979451562237513499839667153356209748681739898344386 08)

(^: 제곱연산, *: 곱연산)

[풀이 과정]
1. $e$ 복원
Alice가 획득한 $s$를 통하여 $e$를 복구할 수 있음
$s=(x_{0,}...,x_{255})$에서 각각의 $x_i$가 $E$, $E_S$중 어느 곡선에 속하는 점인지 확인하는 것으로 $e_i$가 0,1중 어떤 값이었는지 알 수 있음.
즉,
$e_i=0 \Leftarrow \exists\,(x_i,y)\in E$ s.t. $\mathrm{y}^2=\mathrm{x}^3+6\mathrm{x}^2+\mathrm{x}\in \mathrm{F}_{\mathrm{p}^2}$,
$e_i=1 \Leftarrow \exists\,(x_i,y)\in E_S$ s.t. $\mathrm{y}^2=\mathrm{x}^3+\mathrm{A}'\mathrm{x}^2+\mathrm{x}\in \mathrm{F}_{\mathrm{p}^2}$
와 같은 관계가 성립함

이 과정을 통하여 $e$를 복구할 뿐 아니라, $E$위의 점 $G_i$들과 $E_S$위의 점 $\phi_S(R_i)$들의 정보를 획득할 수 있음

2. $S$ 복원
$G_i=S+R_i$에서 $S$의 order($2^n$, $n\le 216$)와 $R_i$의 order($3^n$, $n\le 137$)로 인하여 다음과 같은 관계가 성립함
$3^{137}G_i=3^{137}(S+R_i)=3^{137}S+3^{137}R_i=3^{137}S$

$S$를 기준으로 양변을 정리하면 다음과 같은 관계를 얻을 수 있음
$S=G_i\cdot 3^{137}\cdot(3^{-137} \bmod 2^{216})$
($S$의 order가 $2^{216}$인 점을 고려)

$e$를 복원하는 과정에서 $G_i$의 좌표정보를 획득하였기 때문에, 임의의 $G_i$에 대해 $G_i\cdot 3^{137}\cdot(3^{137} \bmod 2^{216})$를 연산할 경우 ker $\phi_\mathrm{S}$의 generator인 $S$를 복원할 수 있다는 것을 알 수 있음

3. $n_S$ 복원
$S=P_S+n_S\cdot Q_S$를 $Q_S$에 대해 정리하면 다음과 같은 관계를 얻을 수 있음
$n_s\cdot Q_S=S-P_S$

위 관계에서 $n_S$를 찾는 문제는 전형적인 Elliptic Curve Discrete Logarithm Problem에 속하며, 곡선이 supersingular한 경우 효율적으로 문제를 해결 가능한 알고리즘이 알려져있음([참조] 섹션 참고).
따라서, $E$가 supersingular하기 때문에 알려진 알고리즘(MOV attack 등)을 이용하여 개인키 $n_S$을 효율적으로 복원할 수 있음.

실제로 문제에서 주어진 서명값에 대해서 order가 일정한 특정 점들에 대해서 성공적으로 $n_S$가 복원됨을 확인함.

4. $\phi_S$ 복원
서명검증을 위하여 계산하는 isogeny $\alpha_i:E\to E_i'$에 대해서 ker $\alpha_\mathrm{i}=\mathrm{G_i}=\mathrm{S}+\mathrm{R_i}$이기 때문에 $S$, $R_i$ 각각을 kernel로 하는 함수로 합성됨을 확인함. 즉
$\alpha_i=\beta_i\circ\phi_S$
가 성립하며 (증명은 [부록] 섹션에 첨부함) $\alpha_i$를 degree가 2, 3인 각각의 isogeny로 분해 후 degree가 2인 isogeny만 모아서 다시 합성하는 방식으로 $\phi_S$를 복원할 수 있음.

실제로 문제에서 주어진 서명값에 대해서 order가 일정한 특정 점들에 대해서 성공적으로 $\phi_S$가 복원됨을 확인함.

5. 결론

전자서명에 필요한 개인키 $n_S$, $\phi_S$를 서명값중 $s$를 통하여 전부 복원할 수 있기 때문에, 임의의 메세지에 대한 서명값을 획득한 공격자는 개인키를 복원한 후 원하는 메세지에 대한 전자서명을 진행할 수 있음. 따라서 해당 서명 알고리즘은 안전하지 않음

[문제풀이 코드]

문제풀이 코드는 전부 sagemath script로 작성하였으며, 모든 코드가 2023pqc_s.txt 파일과 동일한 폴더 아래 위치할 경우 정상적으로 작동함

1, 2번 파일이 실제 문제풀이(개인키 복원)에 필요한 코드며, 3, 4번은 전자서명 알고리즘 분석을 위해 작성한 공격과 관련없는 코드기 때문에 확인하지 않아도 무방함.

```
============== 파일 1: params.sage (문제풀이에 필요한 변수 선언 코드) ==============
# 문제에서 주어진 파라미터 세팅
(lA,eA), (lB,eB) = (2,216), (3,137)
p = lA^eA * lB^eB - 1  # SIKEp434에서 사용하는 p와 동일
assert p.is_prime()
# y^2 = x^3 + 6*x^2 + x over Fp^2, i^2 + 1 = 0
Fp2.<i> = GF(p^2, modulus=x^2+1)
E0 = EllipticCurve(Fp2, [0,6,0,1,0])
assert E0.is_supersingular()
# Ps
Ps_x_Re =
0x00003CCFC5E1F050030363E6920A0F7A4C6C71E63DE63A0E6475AF621995705F7C84500CB2BB61E950E19EA
B8661D25C4A50ED279646CB48
Ps_x_Im =
0x0001AD1C1CAE7840EDDA6D8A924520F60E573D3B9DFAC6D189941CB22326D284A8816CC4249410FE80D68
047D823C97D705246F869E3EA50
Ps_y_Re =
0x0001AB066B84949582E3F66688452B9255E72A017C45B148D719D9A63CDB7BE6F48C812E33B68161D5AB3A0
A36906F04A6A6957E6F4FB2E0
Ps_y_Im =
0x0000FD87F67EA576CE97FF65BF9F4F7688C4C752DCE9F8BD2B36AD66E04249AAF8337C01E6E4E1A844267B
A1A1887B433729E1DD90C7DD2F
Ps_x = Ps_x_Re + Ps_x_Im * i
Ps_y = Ps_y_Re + Ps_y_Im * i
Ps = E0(Ps_x, Ps_y)
# Qs
Qs_x_Re =
0x0000C7461738340EFCF09CE388F666EB38F7F3AFD42DC0B664D9F461F31AA2EDC6B4AB71BD42F4D7C058E1
3F64B237EF7DDD2ABC0DEB0C6C
Qs_x_Im =
0x000025DE37157F50D75D320DD0682AB4A67E471586FBC2D31AA32E6957FA2B2614C4CD40A1E27283EAAF42
72AE517847197432E2D61C85F5
Qs_y_Re =
```

```
0x0001D407B70B01E4AEE172EDF491F4EF32144F03F5E054CEF9FDE5A35EFA3642A11817905ED0D4F193F3112
4264924A5F64EFE14B6EC97E5
Qs_y_Im                                                                                =
0x0000E7DEC8C32F50A4E735A839DCDB89FE0763A184C525F7B7D0EBC0E84E9D83E9AC53A572A25D19E1464
B509D97272AE761657B4765B3D6
Qs_x = Qs_x_Re + Qs_x_Im * i
Qs_y = Qs_y_Re + Qs_y_Im * i
Qs = E0(Qs_x, Qs_y)
# Ps - Qs
Ds_x_Re                                                                                =
0x0000F37AB34BA0CEAD94F43CDC50DE06AD19C67CE4928346E829CB92580DA84D7C36506A2516696BBE3AE
B523AD7172A6D239513C5FD2516
Ds_x_Im                                                                                =
0x000196CA2ED06A657E90A73543F3902C208F410895B49CF84CD89BE9ED6E4EE7E8DF90B05F3FDB8BDFE489
D1B3558E987013F9806036C5AC
Ds_y_Re                                                                                =
0x00007F65B303A50EF1B4192237611E226A3D13384EF608A6B117365A16E0EB5112156F2012CB029C819F3330
F69BD5C73CCC9A1F1C06CD15
Ds_y_Im                                                                                =
0x0000749095AB8A36C841FBF25A5671A67FDE5023131C73F0EC6031C7E472DAE138FBED0A0BE63C6706CD89
3EF88D32CC766EC67EC056ED33
Ds_x = Ds_x_Re + Ds_x_Im * i
Ds_y = Ds_y_Re + Ds_y_Im * i
Ds = E0(Ds_x, Ds_y)
assert Ps - Qs == Ds
# Pr
Pr_x_Re                                                                                =
0x00008664865EA7D816F03B31E223C26D406A2C6CD0C3D667466056AAE85895EC37368BFC009DFAFCB3D97E
639F65E9E45F46573B0637B7A9
Pr_x_Im = 0x0
Pr_y_Re                                                                                =
0x00006AE515593E73976091978DFBD70BDA0DD6BCAEEBFDD4FB1E748DDD9ED3FDCF679726C67A3B2CC12B
39805B32B612E058A4280764443B
Pr_y_Im = 0x0
Pr_x = Pr_x_Re + Pr_x_Im * i
Pr_y = Pr_y_Re + Pr_y_Im * i
Pr = E0(Pr_x, Pr_y)
# Qr
Qr_x_Re                                                                                =
0x00012E84D7652558E694BF84C1FBDAAF99B83B4266C32EC65B10457BCAF94C63EB063681E8B1E7398C0B241
C19B9665FDB9E1406DA3D3846
Qr_x_Im = 0x0
Qr_y_Re = 0x0
Qr_y_Im                                                                                =
0x0000EBAAA6C731271673BEECE467FD5ED9CC29AB564BDED7BDEAA86DD1E0FDDF399EDCC9B49C829EF53
C7D7A35C3A0745D73C424FB4A5FD2
Qr_x = Qr_x_Re + Qr_x_Im * i
```

```
Qr_y = Qr_y_Re + Qr_y_Im * i
Qr = E0(Qr_x, Qr_y)
# Pr - Qr
Dr_x_Re =
0x0001CD28597256D4FFE7E002E87870752A8F8A64A1CC78B5A2122074783F51B4FDE90E89C48ED91A8F4A0C
CBACBFA7F51A89CE518A52B76C
Dr_x_Im =
0x000147073290D78DD0CC8420B1188187D1A49DBFA24F26AAD46B2D9BB547DBB6F63A760ECB0C2B20BE52FB
77BD2776C3D14BCBC404736AE4
Dr_y_Re =
0x0000DA7A98EA26469B843EBF8D1EE0F00E6786E680AC535F5FF26D25819549C959497D8E8FB14B1BF6764BD
27BAE970D0791AF091E344F22
Dr_y_Im =
0x000048704FEC03D05B06D8A8197DF08D4946E465099F31B75C63A865A23CA2AD41A74F05074E9DC3F45C5A
26F741A0EA1F3C2E6CDA0BB344
Dr_x = Dr_x_Re + Dr_x_Im * i
Dr_y = Dr_y_Re + Dr_y_Im * i
Dr = E0(Dr_x, Dr_y)
assert Pr - Qr == Dr
# Es
Es_A_Re =
0x0000BC39A8C22AFDCAC43EFDD3AB05B45AF0A795D823CD1EC0931D386BFDE2D477DFFFBF2C8463460DE0
586510E99F24323AB8E54BD0026B
Es_A_Im =
0x0000045E901E3BAA12BA1A2D0A37634DEF74A6791039D723962496EB9C4C368FD50BD06BC7D7EF0B2582AD
F73577537BDAA9A064C9AB0DA5
Es = EllipticCurve(Fp2, [0, Es_A_Re + Es_A_Im * i, 0, 1, 0])
# s
s = open("./2023pqc_s.txt", "r").readlines()
s = list(map(lambda x: int(x, 16), s))
sx = s[0::2]
sy = s[1::2]
s = list(zip(sx, sy))
============================================================

============== 파일 2: solver.sage (개인키 복원 공격을 실제로 수행하는 코드) ==============
from sage.schemes.elliptic_curves.hom_composite import EllipticCurveHom_composite
# 문제에서 주어진 환경 세팅
load("params.sage")
# 문제 파라미터 확인
print(f"[*] E0: {E0}")
print(f"[*] Generator of E0[2^216]")
print(f"     Ps.x: {Ps.xy()[0]}")
print(f"     Ps.y: {Ps.xy()[1]}")
print(f"     Qs.x: {Qs.xy()[0]}")
print(f"     Qs.y: {Qs.xy()[1]}")
print(f"[*] Generator of E0[3^137]")
```

```
print(f"      Pr.x: {Pr.xy()[0]}")
print(f"      Pr.y: {Pr.xy()[1]}")
print(f"      Qr.x: {Qr.xy()[0]}")
print(f"      Qr.y: {Qr.xy()[1]}")
print(f"[*] Es: {Es}")
# e값 복구
# 주어진 점이 E0 위에 있는 점이면 Gi, Es에 있는 점이면 phi(Ri)로 판단
Gi = list()
phiSRi = list()
e = str()
for idx in range(len(s)):
    x = s[idx][0] + s[idx][1] * i
    try:
        P = E0.lift_x(x)
        e += "0"
        Gi.append(P)
    except:
        P = Es.lift_x(x)
        e += "1"
        phiSRi.append(P)
print(f"[*] Recovered e => {e} ({len(e)})")
# 풀이 1
# Gi에서 ns 복구 시도
print("[*] Recovering ns...")
for gi in Gi:
    try:
        ns = Qs.discrete_log(gi * (3^137) * inverse_mod(3^137, 2^216) - Ps)
        print(f"[+] Recovered S => {gi * (3^137) * inverse_mod(3^137, 2^216)}")
        print(f"[+] Recovered ns => {ns}")
        break
    except ValueError:
        print("[-] Trying with another Gi ...")
        pass
k = list()
# 풀이 2
# Gi를 kernel로 하는 isogeny 분해
print(f"[*] Recovering phiS...")
for gi in Gi:
    try:
        iso = E0.isogeny(kernel=gi, algorithm="factored")
        factors = list(filter(lambda i: i.degree() == 2, iso.factors()))
        assert len(factors) == 216
        phiS = EllipticCurveHom_composite.from_factors(factors)
        print(f"[+] Recovered phiS => {phiS}")
        mapX, mapY = phiS.factors()[0].rational_maps()
        for phis in phiS.factors()[1:]:
            mapX = phis.rational_maps()[0]
```

```
            mapY = phis.rational_maps()[1]
        print(f"[+] Rational map of phiS: (x, y) -> (x', y') is as follows")
        print(f"x' = {mapX}")
        print(f"y' = {mapY}")
        k.append(phiS)
    except AssertionError:
        print("[-] Trying with another Gi ...")
        pass
============================================================

===== 파일 3: implementation.sage (전자서명 알고리즘 구현체, 분석용) =====
import random
from hashlib import sha256
from Crypto.Util.number import long_to_bytes
print(f"[*] 파라미터 세팅...")
# SIKEp434 파라미터 선언
(lA,eA), (lB,eB) = (2,216), (3,137)
p = lA^eA * lB^eB - 1  # SIKEp434 spec.
assert p.is_prime()
# E : y^2 = x^3 + 6x^2 + x over Fp^2
Fp2.<i> = GF(p^2, modulus=x^2+1)
E = EllipticCurve(Fp2, [0,6,0,1,0])
assert E.is_supersingular()
# Ps
Ps_x_Re                                                                      =
0x00003CCFC5E1F050030363E6920A0F7A4C6C71E63DE63A0E6475AF621995705F7C84500CB2BB61E950E19EA
B8661D25C4A50ED279646CB48
Ps_x_Im                                                                      =
0x0001AD1C1CAE7840EDDA6D8A924520F60E573D3B9DFAC6D189941CB22326D284A8816CC4249410FE80D68
047D823C97D705246F869E3EA50
Ps_y_Re                                                                      =
0x0001AB066B84949582E3F66688452B9255E72A017C45B148D719D9A63CDB7BE6F48C812E33B68161D5AB3A0
A36906F04A6A6957E6F4FB2E0
Ps_y_Im                                                                      =
0x0000FD87F67EA576CE97FF65BF9F4F7688C4C752DCE9F8BD2B36AD66E04249AAF8337C01E6E4E1A844267B
A1A1887B433729E1DD90C7DD2F
Ps_x = Ps_x_Re + Ps_x_Im * i
Ps_y = Ps_y_Re + Ps_y_Im * i
Ps = E(Ps_x, Ps_y)
# Qs
Qs_x_Re                                                                      =
0x0000C7461738340EFCF09CE388F666EB38F7F3AFD42DC0B664D9F461F31AA2EDC6B4AB71BD42F4D7C058E1
3F64B237EF7DDD2ABC0DEB0C6C
Qs_x_Im                                                                      =
0x000025DE37157F50D75D320DD0682AB4A67E471586FBC2D31AA32E6957FA2B2614C4CD40A1E27283EAAF42
72AE517847197432E2D61C85F5
Qs_y_Re                                                                      =
```

```
0x0001D407B70B01E4AEE172EDF491F4EF32144F03F5E054CEF9FDE5A35EFA3642A11817905ED0D4F193F3112
4264924A5F64EFE14B6EC97E5
Qs_y_Im =
0x0000E7DEC8C32F50A4E735A839DCDB89FE0763A184C525F7B7D0EBC0E84E9D83E9AC53A572A25D19E1464
B509D97272AE761657B4765B3D6
Qs_x = Qs_x_Re + Qs_x_Im * i
Qs_y = Qs_y_Re + Qs_y_Im * i
Qs = E(Qs_x, Qs_y)
print(f"[*] 공개키, 개인키 생성...")
# S 연산
ns = random.randrange(1, lA^eA - 1)
S = Ps + ns * Qs
# 공개키, 개인키 연산
phiS = E.isogeny(kernel=S, algorithm="factored")
Es = phiS.codomain()
print(f"[+] 공개키: {Es}")
print(f"[+] 개인키: {ns} ({phiS})")
print(f"[*] 서명 생성...")
# 서명생성
m = b"This is my message"
R = list()
Curves = list()
# Pr
Pr_x_Re =
0x00008664865EA7D816F03B31E223C26D406A2C6CD0C3D667466056AAE85895EC37368BFC009DFAFCB3D97E
639F65E9E45F46573B0637B7A9
Pr_x_Im = 0x0
Pr_y_Re =
0x00006AE515593E73976091978DFBD70BDA0DD6BCAEEBFDD4FB1E748DDD9ED3FDCF679726C67A3B2CC12B
39805B32B612E058A4280764443B
Pr_y_Im = 0x0
Pr_x = Pr_x_Re + Pr_x_Im * i
Pr_y = Pr_y_Re + Pr_y_Im * i
Pr = E(Pr_x, Pr_y)
# Qr
Qr_x_Re =
0x00012E84D7652558E694BF84C1FBDAAF99B83B4266C32EC65B10457BCAF94C63EB063681E8B1E7398C0B241
C19B9665FDB9E1406DA3D3846
Qr_x_Im = 0x0
Qr_y_Re = 0x0
Qr_y_Im =
0x0000EBAAA6C731271673BEECE467FD5ED9CC29AB564BDED7BDEAA86DD1E0FDDF399EDCC9B49C829EF53
C7D7A35C3A0745D73C424FB4A5FD2
Qr_x = Qr_x_Re + Qr_x_Im * i
Qr_y = Qr_y_Re + Qr_y_Im * i
Qr = E(Qr_x, Qr_y)
# 랜덤 점 추출
```

```
print(f"[*] Ri 추출중...")
while len(R) != 256:
    print(". ", end='', flush=True)
    Ri = Pr * random.randrange(1, lB^eB - 1) + Qr * random.randrange(1, lB^eB - 1)
    R.append(Ri)
print()
# 각 Ri에 대한 Es -> Ei isogeny 연산
G = list()
print(f"[*] beta 연산중...")
for Ri in R:
    print(". ", end='', flush=True)
    Gi = S + Ri
    G.append(Gi)
    betai = Es.isogeny(kernel=phiS(Ri), algorithm="factored")
    Curves.append(betai.codomain())
print()
# r, e 계산
print(f"[*] r, e 계산중...")
r = b""
for Ei in Curves:
    print(". ", end='', flush=True)
    re, im = Ei.j_invariant()
    r += long_to_bytes(re.lift()) + long_to_bytes(im.lift())
print()
e = sha256()
e.update(r+m)
e = e.hexdigest()
print(f"[+] e = {e}")
# s 계산
print(f"[*] s 계산중...")
s = list()
b = [int(bit) for byte in bytes.fromhex(e) for bit in format(byte, '08b')]
for i in range(len(b)):
    print(". ", end='', flush=True)
    if b[i] == 0:
        s.append(G[i])
    elif b[i] == 1:
        s.append(phiS(R[i]))
print()
print(f"[+] s: {s}") # 448bit 잘라 저장하는 부분은 생략
============================================================


===== 파일 4: analysis.sage (공격이 실제 성공하는지 3번 파일 구현체에 대해 검증해본 코드, 분석용) =====
from sage.schemes.elliptic_curves.hom_composite import EllipticCurveHom_composite
load("implementation.sage")
'''
# 분석 1
```

만약 S를 알수 있는 경우 다음과 같은 관계가 성립
S-Ps = ns*Qs
주어진 Curve가 supersingular하기 때문에 discrete logarithm을 효율적으로 수행할 수 있는 알고리즘이 존재하며,
따라서 개인키인 ns를 효율적으로 복구할 수 있음
(참고자료: MOV attack)
https://www.dima.unige.it/~morafe/MaterialeCTC/p80-menezes.pdf
https://people.cs.nctu.edu.tw/~rjchen/ECC2009/19_MOVattack.pdf
S+Ri로 계산되는 Gi에서 Ri를 다음과 같이 제거할 수 있음
Gi * 3^137
= (S+Ri) * 3^137: Gi=S+Ri 대입
= S*3^137 + Ri*3^137
= S*3^137: Ri의 order가 3^137임
양 변을 정리하면 다음과 같은 관계가 성립함
S = Gi * 3^137 * (3^-137 mod 2^216): S의 order가 2^216임
최종적으로 S=Ps+ns*Qs 를 대입하면 아래 관계가 성립하며 discrete logarithm을 해결하여 ns를 복구할 수 있음
ns*Qs = Gi * 3^137 * (3^-137 mod 2^216) - Ps
'''

```
print("[*] 분석1: ns 복구 공격 검증")
for Gi in G:
    print(". ", end='', flush=True)
    assert ns == Qs.discrete_log(Gi * (3^137) * inverse_mod(3^137, 2^216) - Ps)
print()
print("[+] 검증 통과")
'''
# 분석 2
Isogeny ai: E -> E_i'의 경우 다음과 같은 합성함수로 분해할 수 있음
ai = bi o phiS'
(phiS': E -> Es, bi: Es -> E_i')
이때 phiS'는 키 생성 과정의 phiS와 동일하기 때문에 phiS를 복구할 수 있음
'''
print("[*] 분석2: phiS 복구 공격 검증")
for Gi in G:
    print(". ", end='', flush=True)
    ai = E.isogeny(kernel=Gi, algorithm="factored")
    phis = EllipticCurveHom_composite.from_factors(ai.factors()[:216])
    assert phiS == phis
print()
print("[+] 검증 통과")
print("[+] 분석 1, 2 검증 완료")
```
==========================================================

[실행 결과]

실제로 주어진 서명값에 대해 공격 수행시(1, 2번 파일 사용) order가 일정한 특정 점들에 대해서 개인키인 $n_S$, $\phi_S$가 성공적으로 복원되며, Macbook M1 Pro 기준 10초 이내에 효율적으로 복원되는것을 확인함.

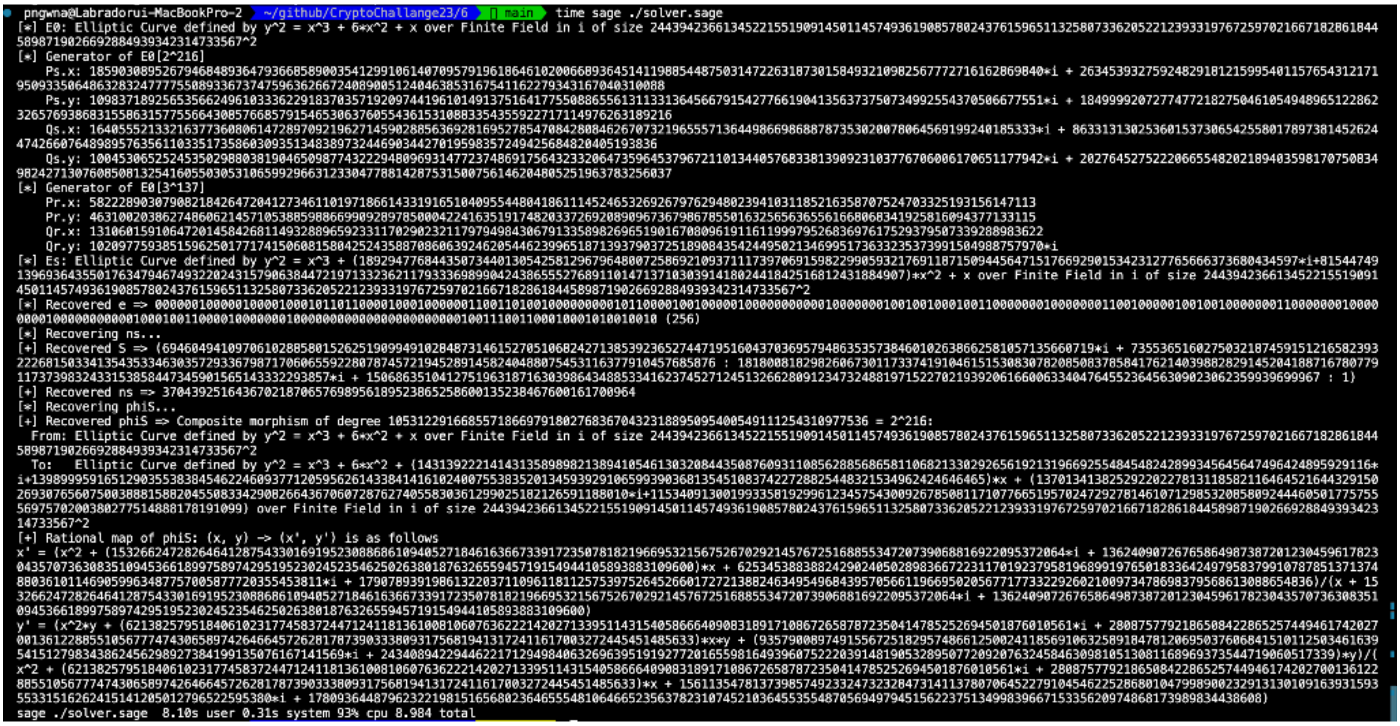


그림 1 실행 결과에 대한 스크린샷

[참조]

- Supersingular curve 상에서의 효율적인 DLP 해결 알고리즘 :https://www.dima.unige.it/~morafe/MaterialeCTC/p80-menezes.pdf
- MOV attack: :https://people.cs.nctu.edu.tw/~rjchen/ECC2009/19_MOVattack.pdf

[부록] - $\alpha_i = \beta_i \circ \phi_S$가 성립하는 이유에 대한 증명

아래는 $< S+R_i >$를 kernel로 하는 $\alpha_i$와 $< S, R_i >$를 kernel로 하는 $\beta_i \circ \phi_S$가 equivalent함을 보이고자 함.

두 isogeny의 kernel의 order가 $| < S+R_i > | = \mathrm{lcm}(\mathrm{ord}(S), \mathrm{ord}(R_i)) = 2^{136} \cdot 3^{137} = | < S, R_i > |$로 같으며 $| < S+R_i > | \subseteq | < S, R_i > |$이고 양쪽 group이 모두 abelian이기 때문에 $| < S+R_i > | = | < S, R_i > |$임.

두 isogeny의 domain, kernel, codomain이 동일하기 때문에 $\alpha_i$와 $\beta_i \circ \phi_S$는 equivalent함. 끝.
(https://www.math.auckland.ac.nz/~sgal018/crypto-book/ch25.pdf, 25-1 챕터 참고)