Crash Course: SageMath for Post-Quantum Cryptography

Post-Quantum Crypto Mini-School 2022, Taipei, Taiwan

Lorenz Panny, Academia Sinica

July 12, 2022

This list of programming exercises is intended to allow you to familiarize yourself with some core functionalities of SageMath frequently needed in cryptography. Most of the tools needed here are introduced in my slides [1]. SageMath also has a comprehensive documentation page online [2], although the logic behind its organization is sometimes a bit confusing. When in doubt, plugging "sagemath [thing_you_need]" into an internet search engine will usually work.

Sample solutions are given at the end.

1 Diffie–Hellman

- Fix a large integer q and an element $g \in \mathbb{Z}/q$. Define them in Sage.
- Choose two secret keys $a, b \in \mathbb{Z}$ for Alice and Bob.
- Verify that *the key exchange works*; that is, check $(g^a)^b = (g^b)^a$.

2 RSA

- Pick two large secret primes $p, q \in \mathbb{Z}$. Let $n = p \cdot q$ and $e = 65537$.
- Define $\lambda = \operatorname{lcm}(p-1, q-1)$ and $d = e^{-1} \bmod \lambda$. (If the inverse doesn't exist, choose different p, q.)
- Choose a message $m \in \mathbb{Z}/n$ and verify that *decryption works*, that is, check $(m^e)^d = m$.

Caveat #1: `is_prime()` by default *proves* primality, which can take a long time for big numbers. Use `is_pseudoprime()` instead, or call `proof.all(False)` once at the beginning of your program to instruct Sage to use probabilistic primality testing.

Caveat #2: `is_prime()` tests primality *in the parent*. For example, `is_prime(10/2)` returns `False` as `10/2` is a rational number in Sage! To make sure you're testing primality in $\mathbb{Z}$, you can use `ZZ(10/2).is_prime()` or `is_prime(ZZ(10/2))`.

3 Linear algebra

Define the matrix

$$A = \begin{pmatrix} 1 & 2 & 3 & 4 \\ 6 & 7 & 8 & 9 \\ 5 & 5 & 5 & 5 \end{pmatrix}$$

and vector

$$y = \begin{pmatrix} 55 \\ 61 \\ 6 \end{pmatrix}$$

over the finite field $\mathbb{F}_{79}$.

1. Use Sage to find a solution x to the equation $A \cdot x = y$.
2. Then find a solution x such that the *second* coefficient of x is zero.
3. Does your method still work if 79 is replaced by a prime of a few hundred bits?

4 Root finding

The following steps give a somewhat unoptimized (but polynomial-time) algorithm to find a root of a polynomial $f \in \mathbb{F}_q[x]$, where q is an odd prime power.

1. Let $f_1 = \gcd((x^q \bmod f) - x, f)$.
2. If f_1 is constant, then f has no root in $\mathbb{F}_q$. Abort.
3. While $\deg(f_1) > 1$:
   (a) Pick $a \in \mathbb{F}_q[x]$ of degree smaller than f_1 at random.
   (b) Compute $g = a^{(q-1)/2} \bmod f_1$ and let $h := \gcd(g - 1, f_1)$.
   (c) If $\deg(h) < 1$, retry with a new random a.
   (d) Otherwise, let $f_1 := h$ and continue with a new random a.
4. At this point f_1 is a linear polynomial; return the obvious root.

Try the algorithm "by hand", using Sage interactively as a "pocket calculator", for the polynomial:

```
sage: q = 101^2
sage: R.<x> = GF(q)[]
sage: f = x^77 + x + 1
```

You should be able to find three distinct roots, one in $\mathbb{F}_{101}$ and two only in $\mathbb{F}_{101^2}$.

Then implement the general form of the algorithm in Sage and test it for some choices of (q, f).

Note: Of course, Sage can do root finding on its own by simply calling `f.roots()`.

To learn more about this algorithm in German, see `https://yx7.cc/docs/tum/polynomfaktorisierung.pdf`. To learn more about this algorithm in English, see §14 of the excellent book *Modern Computer Algebra* by von zur Gathen and Gerhard.

5 Solving multivariate polynomial systems

Find all solutions to the system of equations

$$\begin{aligned}
-7x_2^2 - 3x_3^2 + 3x_0x_4 - x_1x_4 - 2x_3x_4 &= 8 \\
9x_0x_2 + 3x_0x_3 + 5x_0x_4 + 9x_4 &= 4 \\
8x_0x_1 - 9x_1x_3 - 3x_4^2 + 2x_1 + 6x_2 &= 3 \\
3x_1x_2 + 6x_0x_3 + 8x_3x_4 - 6x_0 + 3x_2 &= 0 \\
-9x_2^2 - 5x_0x_4 + 4x_1x_4 - 6x_2x_4 &= 7
\end{aligned}$$

over the finite field $\mathbb{F}_{19}$.

Note: This is a very small toy example of the problem an attacker has to solve in multivariate signature algorithms (such as UOV): The polynomials would make up the public key, and the target values on the right-hand side come from a hash of the message to be signed. By comparison, the legitimate owner of the public key has some additional (secret) information about the polynomial system that makes it easier to solve.

References

[1] Lorenz Panny. *Crash Course: SageMath for Post-Quantum Cryptography*. Presentation slides. 12th July 2022. URL: `https://yx7.cc/docs/sage/sagepqc_taipei_slides.pdf`.

[2] The Sage Developers. *SageMath Reference Manual*. URL: `https://doc.sagemath.org/html/en/reference/`.

Sample solutions

1 Diffie–Hellman

```
sage: q = 10000000000000000000000000000000000000000000001
sage: g = Mod(2, q)
sage: a = randrange(9^999)
sage: b = randrange(9^999)
sage: (g^a)^b == (g^b)^a
True
```

2 RSA

```
sage: proof.all(False)
sage: while True:
sage:     p = randrange(2^999)
sage:     if is_prime(p):
sage:         break
sage: while True:
sage:     q = randrange(2^999)
sage:     if is_prime(q):
sage:         break
sage: n = p*q
sage: e = 65537
sage: lam = lcm(p-1, q-1)
sage: d = (~Mod(e, lam)).lift()
sage: m = Mod(57358167630738549533881472379 21, n)
sage: (m^e)^d == m
True
```

3 Linear algebra

```
sage: A = matrix(GF(79), [[1,2,3,4], [6,7,8,9], [5,5,5,5]])
sage: y = vector([55, 61, 6])
sage: # 1.
sage: x = A.solve_right(y)
sage: x
(58, 38, 0, 0)
sage: # 2.
sage: ker = A.right_kernel()
sage: ker
Vector space of degree 4 and dimension 2 over Finite Field of size 79
Basis matrix:
[ 1  0 76  2]
[ 0  1 77  1]
sage: K = ker.basis_matrix()
sage: K
[ 1  0 76  2]
[ 0  1 77  1]
sage: z = K[:,1].solve_left(vector([x[1]]))
sage: diff = z * K
sage: x -= diff
sage: x
(58, 0, 76, 41)
sage: A*x
(55, 61, 6)
sage: # 3.
sage: # yes
```

4 Root finding

```
sage: def find_root(f):
sage:
sage:     R = f.parent()
sage:     x = R.gen()
sage:
sage:     q = f.base_ring().cardinality()
sage:     assert q % 2 == 1    # odd
sage:
sage:     modf = R.quotient(f)
sage:     xq = lift(modf(x) ^ q)
sage:     f1 = gcd(xq-x, f)
sage:
sage:     if f1.degree() < 1:
sage:         raise ValueError('no root')
sage:
sage:     while f1.degree() > 1:
sage:
sage:         a = R.random_element(f1.degree())
sage:
sage:         modf1 = R.quotient(f1)
sage:         g = lift(modf1(a) ^ ((q-1)//2))
sage:         h = gcd(g-1, f1)
sage:
sage:         if h.degree() >= 1:
sage:             f1 = h
sage:
sage:     assert f1[1] == 1
sage:     return -f1[0]
```

```
sage: q = 101^2
sage: R.<x> = GF(q)[]
sage: f = x^77 + x + 1
sage: find_root(f)
47*z2 + 57
sage: find_root(f)
23
sage: find_root(f)
47*z2 + 57
sage: find_root(f)
54*z2 + 43
sage: find_root(f)
54*z2 + 43
sage: find_root(f)
23
```

5 Solving multivariate polynomial systems

```
age: R.<x0,x1,x2,x3,x4> = GF(19)[]
sage: eqs = [
....:           -7*x2^2 - 3*x3^2 + 3*x0*x4 - x1*x4 - 2*x3*x4 - 8,
....:           9*x0*x2 + 3*x0*x3 + 5*x0*x4 + 9*x4 - 4,
....:           8*x0*x1 - 9*x1*x3 - 3*x4^2 + 2*x1 + 6*x2 - 3,
....:           3*x1*x2 + 6*x0*x3 + 8*x3*x4 - 6*x0 + 3*x2 - 0,
....:           -9*x2^2 - 5*x0*x4 + 4*x1*x4 - 6*x2*x4 - 7,
....:       ]
sage: Ideal(eqs).dimension()
0
sage: Ideal(eqs).variety()
[{x4: 6, x3: 6, x2: 4, x1: 12, x0: 8}, {x4: 3, x3: 17, x2: 5, x1: 0, x0: 14}]
```