PRIME

a cryptography oriented programming language

Alexander Liebeskind (Project Manager), Computer Engineering
Nikhil Mehta (Language Guru), Computer Engineering
Thomas Tran (Systems Architect), Computer Science
Pedro B T Santos (Tester), Computer Science

Programming Languages and Translators, Spring 2021

Purpose and Planning

- Cryptography is hard enough without difficult and messy code
- Focus on what matters: Write crypto math as simple as possible
- Handle big numbers without too much worry
- Include general functionalities (strings, ints, etc) to facilitate usage

- Deadlines
- Work Style and Responsibilities
- Work Setup and Communication
- Resolving Setbacks

Distinctive Features: Lints

Big Number Arithmetic

- **Initialization and Declaration**

```
lint a;
lint b;
a = 1l;
b = 2l;
```

- **Lint operations**

```
a + b; a - b;
a * b; a / b;
a % b;

a/\b /* lint a raised to int b */
a‘b; /* multiplicative inverse of a mod */
a^b@c/* a raised to b mod c */

a==b; a!=b;
a<=b; a>=b; a>b; a<b;
```

```
#include <gmp.h>
#include <stdlib.h>

mpz_t a;
mpz_t b;
mpz_t c;
mpz_init(a);
mpz_init(b);
mpz_init(c);
mpz_set_si(a, (long)2);
mpz_set_si(a, (long)2);
mpz_add(c, a, b);
```

C GMP Library - addition

Distinctive Features: Curves and Pts *(theory)*

Elliptic Curve Cryptography

- Prevalent in digital signature schemes and public key cryptography
  - Depend on unfactorable large primes (lints) as coefficients
- Points form a finite abelian group over point addition on modular elliptic curves
  - Addition: Reflection of third point of intersection across y axis
  - Additive Inverse: Exists as the reflection across the x axis
  - Identity element: Point at infinity (represented as (-1, -1))
  - Generator point: Single point generating abelian group

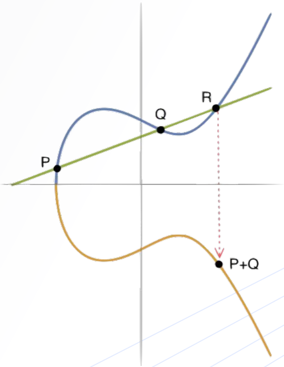
Mathematical Definition:

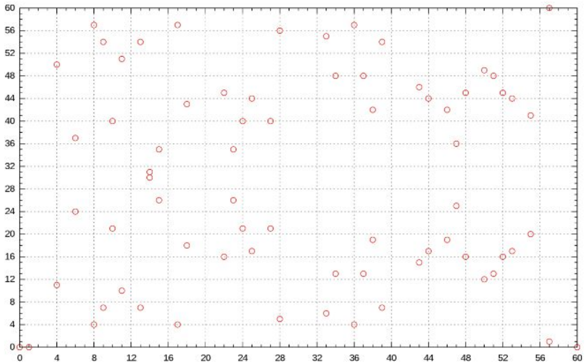
$$\boxed{y^2 \equiv x^3 + bx + c \pmod{p},} \qquad (0 \le b, c < p)$$

Addition:

$$\boxed{(x_1, y_1) \oplus (x_2, y_2) = (x_3, y_3)} \quad \text{where} \quad \boxed{\begin{aligned} x_3 &= m^2 - x_1 - x_2, \\ y_3 &= m(x_1 - x_3) - y_1. \end{aligned}}$$

$$m \equiv \begin{cases} (y_2 - y_1)(x_2 - x_1)^{-1} \pmod{p} & \text{if } (x_1, y_1) \ne (x_2, y_2), \\ (3x_1^2 + b)(2y_1)^{-1} \pmod{p} & \text{if } (x_1, y_1) = (x_2, y_2). \end{cases}$$





The set of points in $y^2 = x^3 - x \pmod{61}$
source: Wikimedia Commons

Distinctive Features: Curves and Pts (*practice*)

Initializing Curves

```
curve crv;

/* create the curve */

p = 785963102379428822376694789446897396207498568951 1;
a = 317689081251325503476317464138276932727469559271;
b = 485714067917757273461840828810056205973454266521;

crv = [(a, b) : p];
```

Initializing Points

```
pt q;
```

```
x1 = 771507216262649826170648268565579889907769254176 1;
y1 = 390157510246556628525279459266514995562533196655 1;

prints("Elliptic Curve E:");
printc(crv);

/* create the point */

q = [x1, y1] & crv;
```

Point Operations

```
/* add two points */
r = p + q;

/* find additive inverse */
r = -p;

/* multiply point by a lint */
r = 123l*p;

/* find the identity element */
r = p + -p;
```

Distinctive Features: Keywords

- types as in previous slide
- statements: if, else, for, while, return, main
- printing
- type-casting via function call
- key cryptography helpers: encode, decode, random

Examples:

- print	- int
- printl	- lint
- prints	- string
- printpt	- point
- printc	- curve

```
string s;
lint a;
s = "Hello World";
a = encode(s);
printl(a);
prints(decode(a));
```

Sample Encode/Decode

```
int a;
lint b;
a = 2387468;
printl(tolint(a));
b = tolint(a);
```

Sample int->lint casting

```
lint max; lint seed;
lint rand;
max = 123451;
seed = 101;
rand = random(seed, max);
printl(rand);
```

Sample Random function use

Distinctive Features: Interfacing

- C GMP library (all credits to contributors of that library)
- Abstract away the struct types
- The user writes the math/pseudocode without needing to know the full scope

Examples:

```c
struct point *ptadd(struct point *p1, struct point *p2)
{
    struct poly *curve = (struct poly *)malloc(sizeof(struct poly));
    struct point *sum  = (struct point *)malloc(sizeof(struct point));

    mpz_t xcoeff;
    mpz_init_set(xcoeff, p1->curve->x_coeff);

    mpz_t c;
    mpz_init_set(c, p1->curve->c);

    mpz_t mod;
    mpz_init_set(mod, p1->curve->mod);

    Poly( curve, xcoeff, c, mod);

    mpz_t zero;
    mpz_init_set_str(zero, "0", 10);

    mpz_t p3x;
    mpz_t p3y;

    mpz_init(p3x);
    mpz_init(p3y);

    /* if pt is -1, -1 -> pt at infinity acts as identity element
     * return other point
     */

    if( mpz_sgn(p1->i) == -1 && mpz_sgn(p1->j) == -1 ){
        mpz_set(p3x, p2->i);
        mpz_set(p3y, p2->j);
        Point( sum, p3x, p3y, curve);


    } else if ( mpz_sgn(p2->i) == -1 && mpz_sgn(p2->j) == -1 ){
        mpz_set(p3x, p1->i);
        mpz_set(p3y, p1->j);
        Point( sum, p3x, p3y, curve);

    }
    else{
    /* build local x and y coords */

        mpz_t p1x;
        mpz_t p1y;
        mpz_t p2x;
        mpz_t p2y;

        mpz_init(p1x);
        mpz_init(p1y);
        mpz_init(p2x);
        mpz_init(p2y);

        mpz_mod(p1x, p1->i, mod);
        mpz_mod(p1y, p1->j, mod);
        mpz_mod(p2x, p2->i, mod);
        mpz_mod(p2y, p2->j, mod);

        /* check if they are inverses of one another */

        mpz_t neg;
        mpz_init(neg);
        mpz_neg(neg, p2y);
        if(mpz_congruent_p(p1y, neg, mod))
```

```c
        {
            mpz_set_str(p3x, "-1", 10);
            mpz_set_str(p3y, "-1", 10);
            Point( sum, p3x, p3y, curve);

        /*mpz_clear(neg);
        mpz_clear(p1x);
        mpz_clear(p1y);
        mpz_clear(p2x);
        mpz_clear(p2y);*/

        /*mpz_clear(xcoeff);
        mpz_clear(c);
        mpz_clear(z:qero);
        mpz_clear(mod);*/

           // return sum;
        }
        else{

            //slope

            mpz_t m;
            mpz_init(m);

            /* if pts are not the same */
            if(mpz_cmp(p1x, p2x) != 0 || mpz_cmp(p1y, p2y) != 0)
            {
                mpz_t tmpy;
                mpz_t tmpx;
                mpz_init(tmpy);
                mpz_init(tmpx);

                mpz_sub(tmpy, p2y, p1y);
                mpz_sub(tmpy, tmpy, mod);
                mpz_sub(tmpx, p2x, p1x);
                mpz_mod(tmpx, tmpx, mod);

                mpz_invert(tmpx, tmpx, mod);
                mpz_mul(m, tmpy, tmpx);
                mpz_mod(m, m, mod);

                mpz_clear(tmpy);
                mpz_clear(tmpx);
            } else { /* if points are same */
                mpz_t tmpx;
                mpz_t tmpy;
                mpz_init(tmpx);
                mpz_init(tmpy);

                mpz_mul(tmpx, p1x, p1x);
                mpz_mod(tmpx, tmpx, mod);
                mpz_mul_si(tmpx, tmpx, (long) 3);
                mpz_mod(tmpx, tmpx, mod);
                mpz_add(tmpx, tmpx, xcoeff);
                mpz_mul_si(tmpy, p1y, (long) 2);
                mpz_mod(tmpy, tmpy, mod);
                mpz_invert(tmpy, tmpy, mod);
                mpz_mul(m, tmpx, tmpy);
                mpz_mod(m, m, mod);

                mpz_clear(tmpx);
                mpz_clear(tmpy);
            }

            /* find p3x */
            mpz_t tmp;
            mpz_init(tmp);
            mpz_mul(tmp, m, m);
            mpz_mod(tmp, tmp, mod);
            mpz_sub(tmp, tmp, p1x);
            mpz_sub(tmp, tmp, p2x);
            mpz_mod(tmp, tmp, mod);
```

```c
            mpz_set(p3x, tmp);

            /* find p3y */

            mpz_sub(tmp, p1x, p3x);
            mpz_mul(tmp, tmp, m);
            mpz_sub(tmp, tmp, p1y);
            mpz_mod(tmp, tmp, mod);
            mpz_set(p3y, tmp);

            /* build pt */

            Point( sum, p3x, p3y, curve );

            mpz_clear(m);
            mpz_clear(tmp);
        }

        mpz_clear(neg);

        mpz_clear(p1x);
        mpz_clear(p1y);
        mpz_clear(p2x);
        mpz_clear(p2y);
    }

    mpz_clear(xcoeff);
    mpz_clear(c);
    mpz_clear(zero);
    mpz_clear(mod);


    mpz_clear(p3x);
    mpz_clear(p3y);


    return sum;
    /*int i, j;
    int m;
    int b = 1;

    if(p1.i == p2.i && p1.j == p2.j){
  m = (3*(p1.i)^2 + b)/(2*p1.j);
    }
    else{
  m = (p2.j-p1.j)/(p2.i-p1.i);
    }

    i = m^2 - p1.i - p2.i;
    j = m*(p1.i - i) - p1.j;

    return Point(i, j);*/
}
```

```
int main()
{
    /* creation and assignment */
    pt p1;
    pt p2;
    pt p3;
    curve crv;

    crv = [(51, 121) : 131];
    p1 = [21, 21] & crv;
    p2 = [21, 111] & crv;
    printpt(p2 + p1);
```

Distinctive Features: Interfacing

- C GMP library (all credits to contributors of that library)
- Abstract away the struct types
- The user writes the math/pseudocode without needing to know the full scope
- Get Element Ptr
- Implicit Ptr passing

Examples:

```
    and formal_types =
  Array.of_list (let new_params = (match fdecl.styp with
                                      A.Lint  -> (A.Lint, "sret") :: fdecl.sparams
                                    | A.Point -> (A.Point, "sret") :: fdecl.sparams
                                    | _       -> fdecl.sparams
                                    ) in
                  List.map (fun (t,n) -> match t with
                    A.Lint  when n = "sret" -> L.pointer_type (ltype_of_typ t)
                  | A.Point when n = "sret" -> L.pointer_type (ltype_of_typ t)
                  | _       -> ltype_of_typ t) new_params)
    in let ftype = L.function_type (match fdecl.styp with
                                      A.Lint -> L.pointer_type mpz_t
                                    | A.Point -> L.pointer_type point_t
                                    | _       -> ltype_of_typ fdecl.styp) formal_types
  in
    StringMap.add name (L.define_function name ftype the_module, fdecl) m in
  List.fold_left function_decl StringMap.empty functions in
```

```
   let llargs = (match fdecl.styp with
                  A.Lint -> let space = L.build_alloca mpz_t "sret_space" builder
                            in
                            L.build_in_bounds_gep space [| zero |] "" builder ::
  llargs
                | A.Point -> let space = L.build_alloca point_t "sret_space" builder
                            in
                            L.build_in_bounds_gep space [| zero |] "" builder ::
  llargs
                | _      -> llargs) in
      L.build_call fdef (Array.of_list llargs) result builder
```

Testing

- Test-Driven Development
- Automate as much as possible with Bash and CircleCI
- Github
- Communication
- Test at a local and global level with Regression Test Suite
- Best way to find bugs and feels rewarding when we see OK

```
Test: fail_lcast OK
Test: fail_lint1 OK
Test: fail_mpow OK
Test: fail_point_acc OK
Test: fail_point_match OK
Test: fail_point_type OK
Test: fail_poly_type OK
Test: fail_pt_mul OK
Test: fail_return OK
Test: fail_var1 OK
Test: fail_while OK
Test: fail_while1 OK
Test: test_add OK
Test: test_ass OK
Test: test_big_curve OK
Test: test_big_num OK
Test: test_decode OK
Test: test_elseif OK
Test: test_encode OK
Test: test_for OK
Test: test_func OK
Test: test_hello OK
```

Prime 243	Success	Build_Test	demos2 93ea4d2 finished RSA Demo	13h ago	51s	...
Prime 242	Success	Build_Test	string_parsing 24f8b30 change encode syntax	13h ago	1m 8s	...
Prime 241	Success	Build_Test	demos2 5b9b7e0 added ecc demo, fixed string parsing	13h ago	1m 25s	...
Prime 240	Success	Build_Test	main 32b646c Merge pull request #45 from thomasundo2/access	15h ago	1m 6s	...
Prime 239	Success	Build_Test	access 89265da removed test file	15h ago	52s	...
Prime 238	Success	Build_Test	main 7d2a146 Fix point ret return and ocaml warnings	15h ago	1m 17s	...
Prime 237	Failed	Build_Test	access 621ba4d chaning to printc from printpoly	16h ago	47s	...

Demos: RSA, Elliptic Curves, Diffie-Hellman

Takeaways

- Programming languages are hard
  - LLVM even more so
  - If you have to think about whether something works, it doesn't
- OCaml and some functional programming
- Time and Planning
- OH are important
- WFH Communication
- Zoom pro accounts

Further objectives:

- Operator overloading
- Conciseness: Multiple assignment, declare and assign
- Unified print function (consolidate print(), printl(), printc(), printpt())
- Garbage collection

Questions

Acknowledgements

- Professor Edwards
- All PLT TA's
- Professor Dorian Goldfeld's MATH UN3025 Lecture Notes