Final Presentation

Maggie Mallernee, Zachary Silber, Michael Tong, Richard Zhang, Joshua Zweig

Overview

What is C% (and however do you pronounce it)?

Why C%?

Cryptography Implementation is Hard

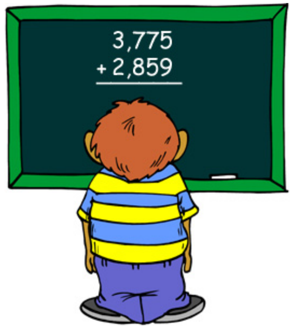
Software developers are failing to implement crypto correctly, data reveals

Lack of specialized training for developers and crypto libraries that are too complex lead to widespread encryption failures

Goals

Encourage Correctness

Improve Readability



Ease the burden of large number arithmetic

Extensibility

Overview

Basics of C%

- Compiles to LLVM
- C-like syntax and semantics
- Heap memory management: malloc() and free()
- User input: printf() and scanf()

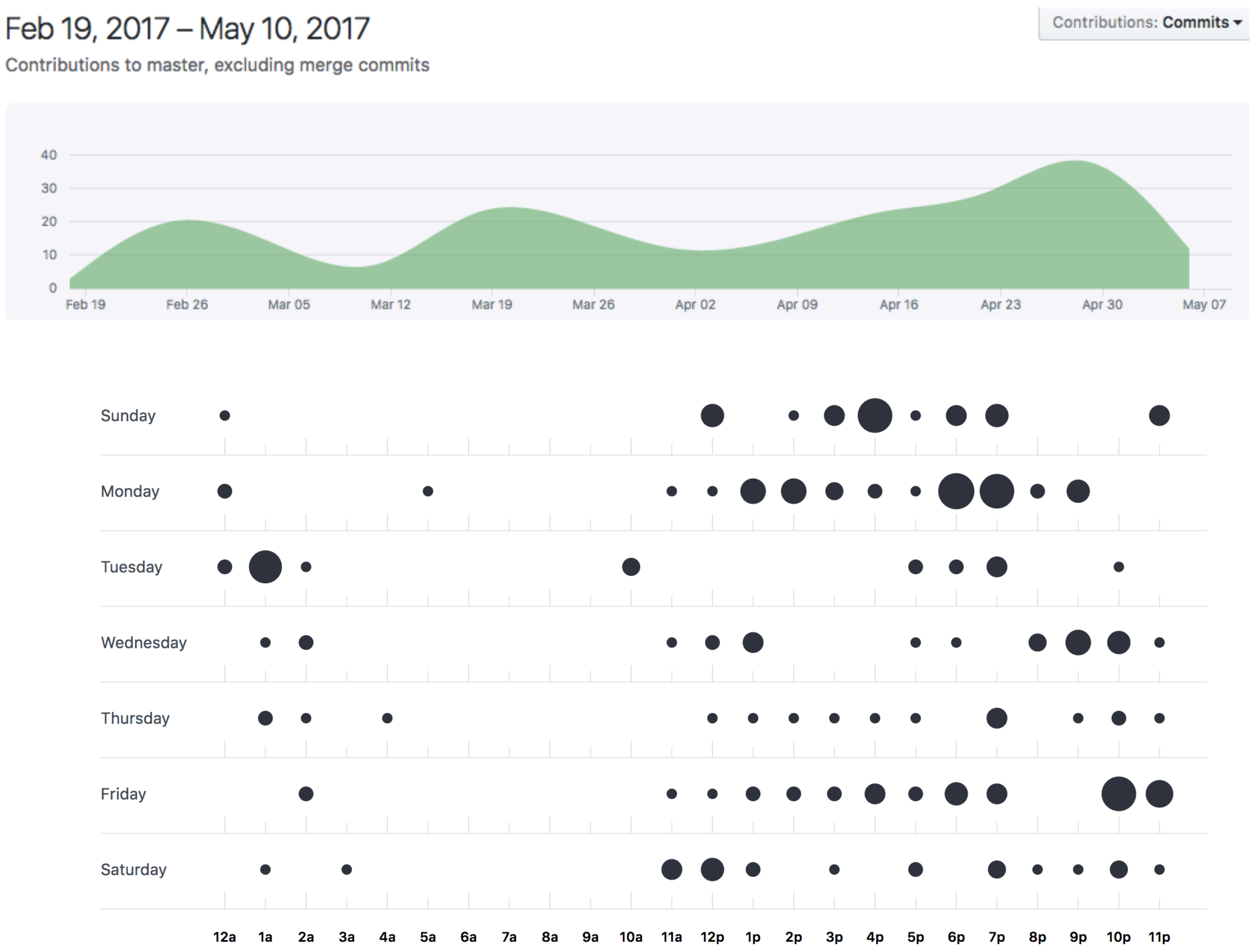
The Big Stuff

- Cryptographic types: Stones, Mints, Elliptic Curves, and Points
- Painless arbitrary precision arithmetic
- Overloaded operators covering group operations of modular integers and points over curves

Project Management

51 PRs, 37 Closed, 282 commits

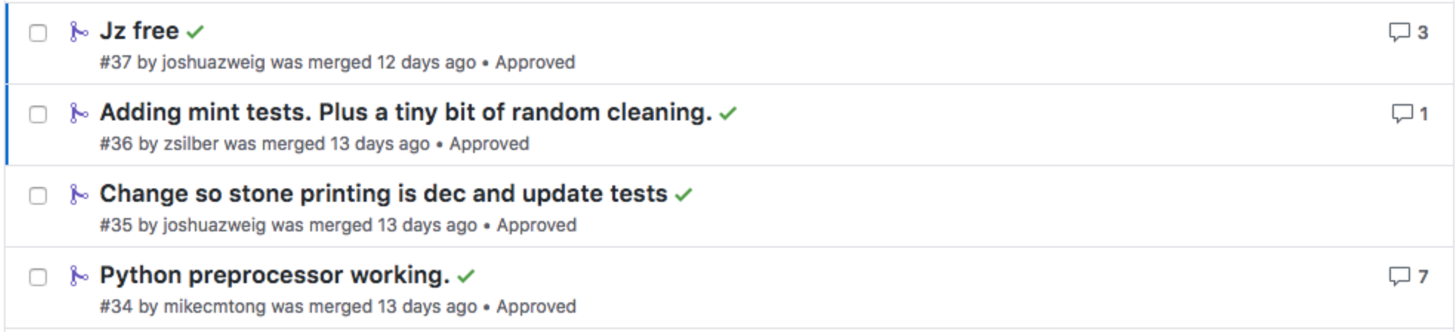
Version Control



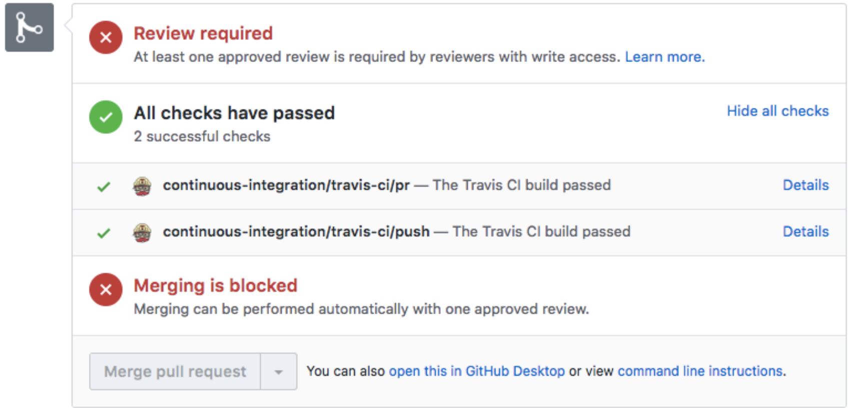
Testing

It works! This is how we know!

Continuous Integration



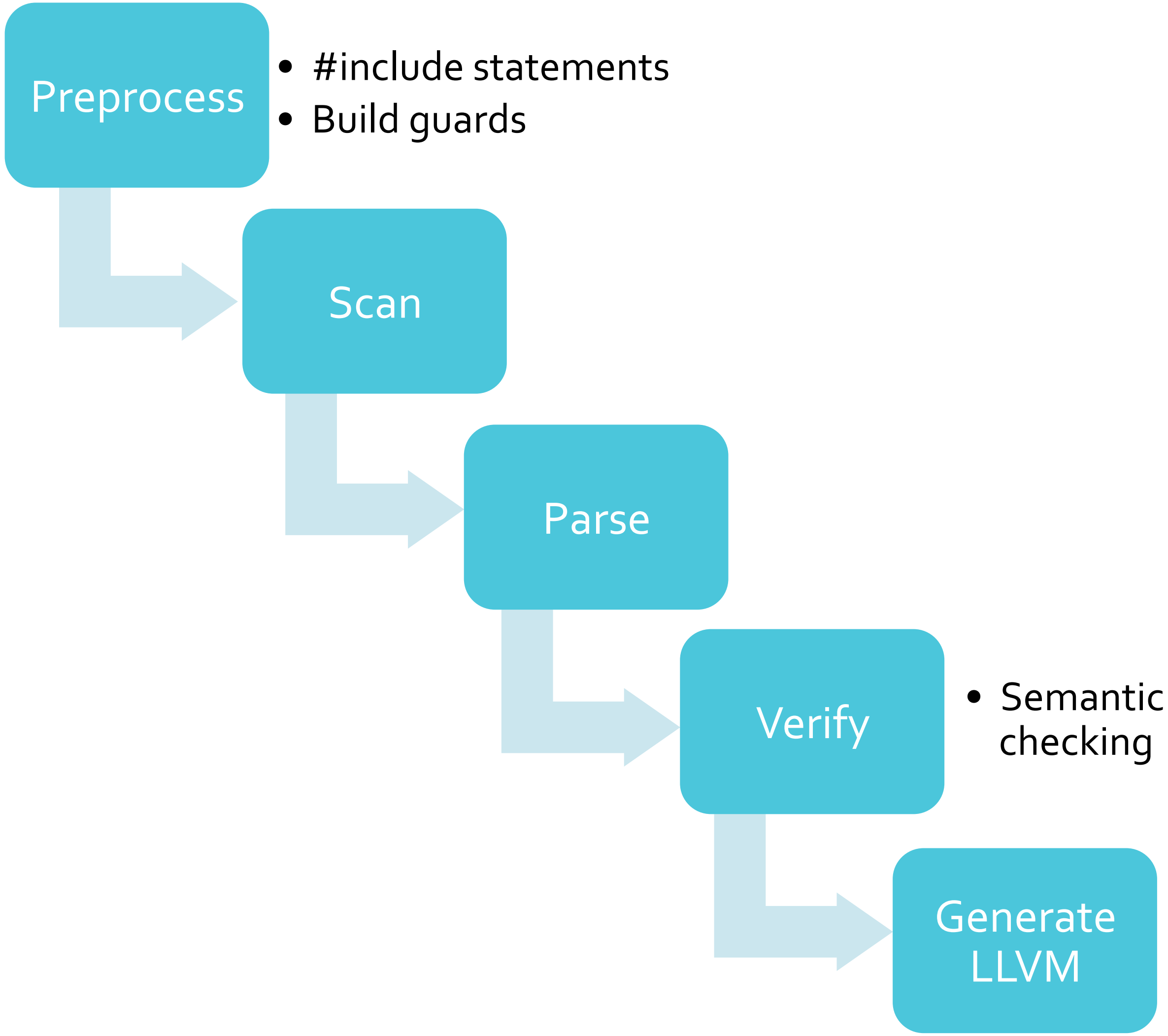
- Execute entire test suite on every push/PR
- Provide detailed feedback
- Enforce all tests passing



Architecture

A journey from source code to shared secrets

The Big Picture



- #include statements
- Build guards

- Semantic checking

- Link to openssl bignum
- Built in functions
- Type specific behavior

BigNum Arithmetic

openssl/bn.h
(BIGNUM)

special_arith.c

codegen.ml
(stone)

```
BIGNUM *r1 = BN_new();
BIGNUM *r2 = BN_new();

BN_add(r1, a, b);
BN_add(r2, r1, c);
```

```
a + b + c;
```

Compiler Interface

```
Welcome to the C% compiler CMC!

USAGE: ./bin/cmc [-h help] [-t token] [-a ast] [-l llvm] [-c ll-file] [-s s-file] [-e exe-file] <file-name>.cm


OPTIONS:
        -h      help            This option prints this message!

        -t      token           This option prints the tokenized program to stdout.

        -a      ast             This option prints the abstract syntax tree of the program to stdout.

        -l      llvm            Compiles <file-name>.cm to llvm and prints the result to stdout.

        -c      ll-file         Compiles <file-name>.cm to llvm and puts the result in <file-name>.ll. This is the default option.

        -s      assembly        Compiles <file-name>.cm to llvm, translates to assembly, and puts the result in <file-name>.s
                                        (leaves <file-name>.ll in directory as well)

        -e      executable      Creates the executable version of <file-name>.cm, simply called <file-name> to be run ./<file-name>
                                        (leaves behind the corresponding .ll and .s files as well)
```

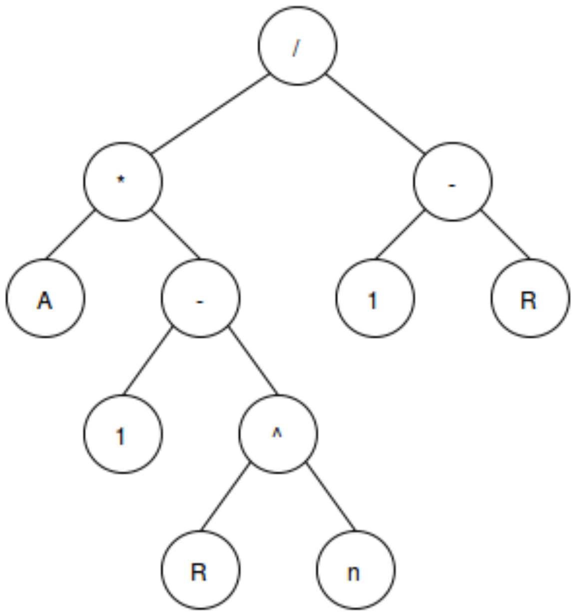
- Options allowing user access to each step in the compilation process
- Can see the tokenized program, AST, LLVM (sdtout or .ll file), assembly (.s), and compile to a full executable

What exactly does C% do for me?

We’re glad you asked

The Jist

- It's like C!
  - Syntax/Comments
  - Expressions/Statements
  - Control Flow
- Key Features
  - Pre-processing
  - Input/Output
  - Scoping
  - Declaration flexibility
  - Memory management
  - Operator overloading



Cryptographic Types

- Stones: Basis of all other cryptographic types, links to OpenSSL/BN

```
stone x;
x = "5";
```

```
stone x;
x = "999999999999999999";
```

- Mints: Integers in a finite field modulo some prime p

```
mint m;
m = <"13", "101">;
```

```
stone s1;
stone s2;
mint m;

m = <s1, s2>;
```

- Curves: Elliptic curves, comprised of two mints

```
mint m1;
mint m2;
curve *c;

c = <m1, m2>;
```

- Points: Points on a curve, with operations relative to that curve

```
curve c;
point *p;

p = <c, "12", "103">;
```

```
curve c;
point *pInf;

pInf = <c, ~>;
```

Caesar Cipher?

We have you covered

Cryptography Library

- We provide some cool examples!
  - Caesar Cipher
    - Simple shifting using Mints
  - Stream Cipher
    - Mints and Access methods provide easy tracking of repeated values mod a constant moduli with improved readbility
  - Diffie Hellman (Modular integer and ECC)
    - Points improve readability
    - No confusion on Point arithmetic
  - ElGamal Encryption
    - Extremely intuitive and clear when using built in Curves and Points

ECC:
C v C%

```c
void point_add_func_help(struct point *R, struct point *P, struct point *Q) {
    R->E = P->E;
    if (P->inf) {
        R->x = Q->x;
        R->y = Q->y;
        R->inf = Q->inf;
    } else if (Q->inf) {
        R->x = P->x;
        R->y = P->y;
        R->inf = P->inf;
    } else { /* neither points are inf */
        BIGNUM *xval = BN_new();
        BIGNUM *yval = BN_new();
        BN_CTX *ctx = BN_CTX_new();
        BIGNUM *lambda = BN_new();
        BIGNUM *t1 = BN_new();
        BIGNUM *t2 = BN_new();
        // calculate lambda
        BN_sub(t1, Q->y, P->y);
        BN_sub(t2, Q->x, P->x);
        if (BN_is_zero(t2)) {
            if (BN_is_zero(t1)) {
                /* same point, double it
                 * calculate lambda this way */
                BN_mod_sqr(t1, P->x, P->E.a.mod, ctx);
                BN_mod_add(t2, t1, t1, P->E.a.mod, ctx); /* t2 = 2 t1 */
                BN_mod_add(t2, t1, t1, P->E.a.mod, ctx); /* t1 = t1 + t2 = 3t1 */
                BN_mod_add(t1, t1, t2, P->E.a.mod, ctx);
                BN_mod_add(t1, t1, P->E.a.val, P->E.a.mod, ctx);

                BN_mod_add(t2, P->y, P->y, P->E.a.mod, ctx); /* t2 = 2 P.y */
                BN_mod_inverse(t2, t2, P->E.a.mod, ctx);

                BN_mod_mul(lambda, t1, t2, P->E.a.mod, ctx);

            } else {
                /* additive inverses, return inf
                 * Fill coords with junk values from P */
                R->x = P->x;
                R->y = P->y;
                R->inf = 1;
                BN_free(t1);
                BN_free(t2);
                BN_CTX_free(ctx);
                return;
            }
        } else {
            // finish calculating lambda for "normal" case
            BN_mod_inverse(t2, t2, P->E.a.mod, ctx);
            BN_mod_mul(lambda, t1, t2, P->E.a.mod, ctx);
        }
        //calculate xval
        BN_mod_sqr(t1, lambda, P->E.a.mod, ctx);
        BN_mod_sub(t1, t1, P->x, P->E.a.mod, ctx);
        BN_mod_sub(xval, t1, Q->x, P->E.a.mod, ctx);

        //calculate yval
        BN_mod_sub(t1, P->x, xval, P->E.a.mod, ctx);
        BN_mod_mul(t1, lambda, t1, P->E.a.mod, ctx);
        BN_mod_sub(yval, t1, P->y, P->E.a.mod, ctx);

        //put in values
        R->x = xval;
        R->y = yval;
        R->inf = P->inf;

        BN_free(t1);
        BN_free(t2);
        BN_CTX_free(ctx);
    }
}
```

```c
point a;
point b;

a + b;
```