ec3: Elliptic Curve Cryptography Compiler

Michael McLoughlin

Recurse Center m6’19

Diffie-Hellman

Diffie-Hellman Key Exchange

Alice and Bob want to *establish a shared secret*.

Use integers modulo a *prime p*, and choose a multiplicative *generator g*.

Protocol

Alice Bob

$$\xrightarrow{g^x \pmod{p}}$$

$$\xleftarrow{g^y \pmod{p}}$$

Protocol

Alice Bob

$$g^x \pmod{p}$$
$$\xrightarrow{\hspace{6cm}}$$

$$g^y \pmod{p}$$
$$\xleftarrow{\hspace{6cm}}$$

Alice chooses secret x.

Sends $g^x \pmod{p}$ to Bob.

Protocol

Alice $\qquad\qquad\qquad\qquad$ Bob

$$\xrightarrow{\qquad g^x \pmod{p} \qquad}$$

$$\xleftarrow{\qquad g^y \pmod{p} \qquad}$$

Bob chooses secret y.

Sends $g^y \pmod{p}$ to Alice.

Both Alice and Bob can compute

$$g^{xy}$$

Alice knows x and received g^y so computes

$$(g^y)^x \pmod{p}$$

Bob knows y and received g^x so computes

$$(g^x)^y \pmod{p}$$

You could break this if you could *find x given* g^x.

This is the discrete log problem.

$$\mathcal{O}\left(e^{c\sqrt[3]{(\log p)(\log\log p)^2}}\right)$$

Discrete log for integers modulo p
is sub-exponential.

Requirements

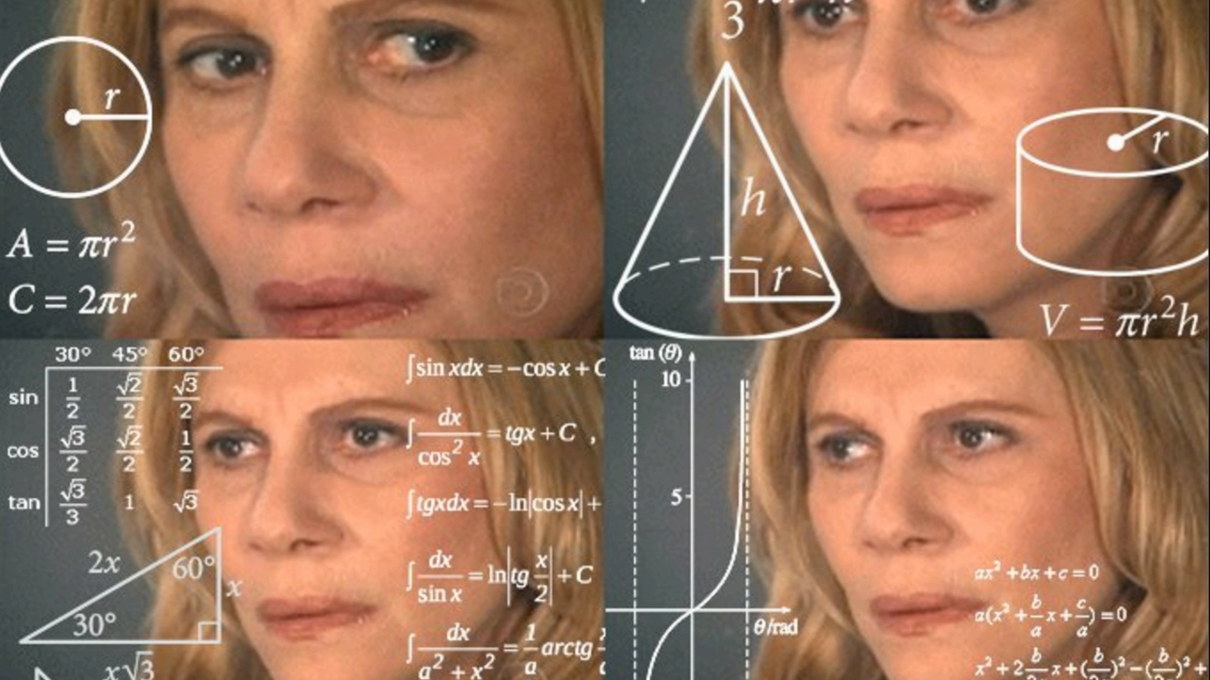
- **Set:** set of elements G to manipulate
- **Operation:** definition of g^x
- **Security:** discrete log problem is *hard*

We can perform Diffie-Hellman on any mathematical group.

Elliptic Curves

Not to be confused with Ellipse.

Smooth, projective, algebraic curve of genus one.

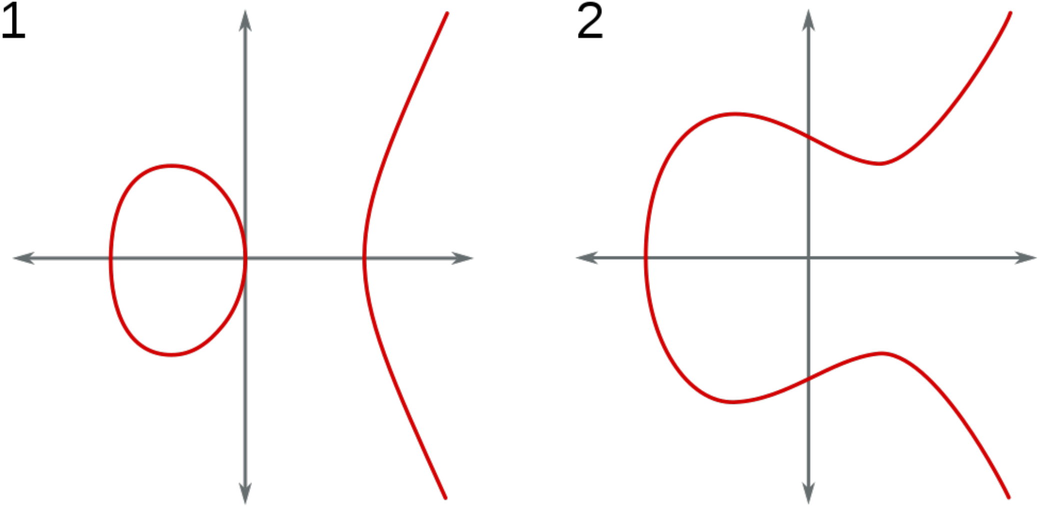


r
$A = \pi r^2$
$C = 2\pi r$
3
h
r
r
$V = \pi r^2 h$
30° 45° 60°
sin $\frac{1}{2}$ $\frac{\sqrt{2}}{2}$ $\frac{\sqrt{3}}{2}$
cos $\frac{\sqrt{3}}{2}$ $\frac{\sqrt{2}}{2}$ $\frac{1}{2}$
tan $\frac{\sqrt{3}}{3}$ 1 $\sqrt{3}$
$2x$
60°
x
30°
$x\sqrt{3}$
$\int \sin x dx = -\cos x +$
$\int \frac{dx}{\cos^2 x} = tgx + C$,
$\int tgxdx = -\ln|\cos x| +$
$\int \frac{dx}{\sin x} = \ln\left|tg\frac{x}{2}\right| + C$
$\int \frac{dx}{a^2+x^2} = \frac{1}{a} arctg$
tan (θ)
10
5
θ/rad
$ax^2 + bx + c = 0$
$a(x^2 + \frac{b}{a}x + \frac{c}{a}) = 0$

$$y^2 = x^3 + ax + b$$

1

2

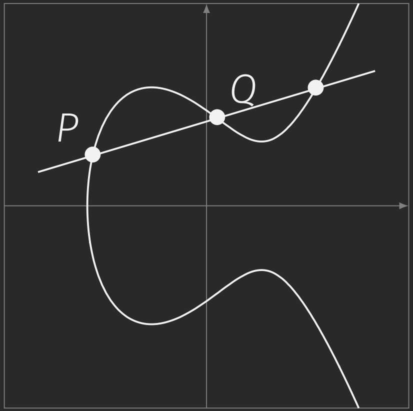


$y^2 = x^3 - x$

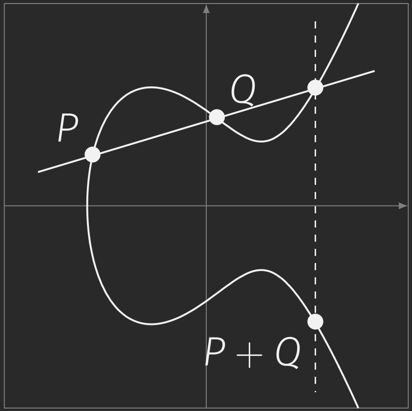
$y^2 = x^3 - x + 1$

Elliptic Curve Diffie-Hellman

- **Set:** points (x, y) on the elliptic curve
- **Operation:** repeated point addition kP
- **Security:** discrete log on elliptic curves has *exponential complexity*



Addition $P + Q$ “Chord rule”



Addition $P+Q$ “Chord rule”

In *Elliptic Curve Diffie-Hellman* we replace exponentiation g^x with repeated addition of points kP.

Elliptic Curve Diffie-Hellman

- **Set:** points (x, y) on the elliptic curve
- **Operation:** repeated point addition kP
- **Security:** discrete log on elliptic curves has *exponential complexity*

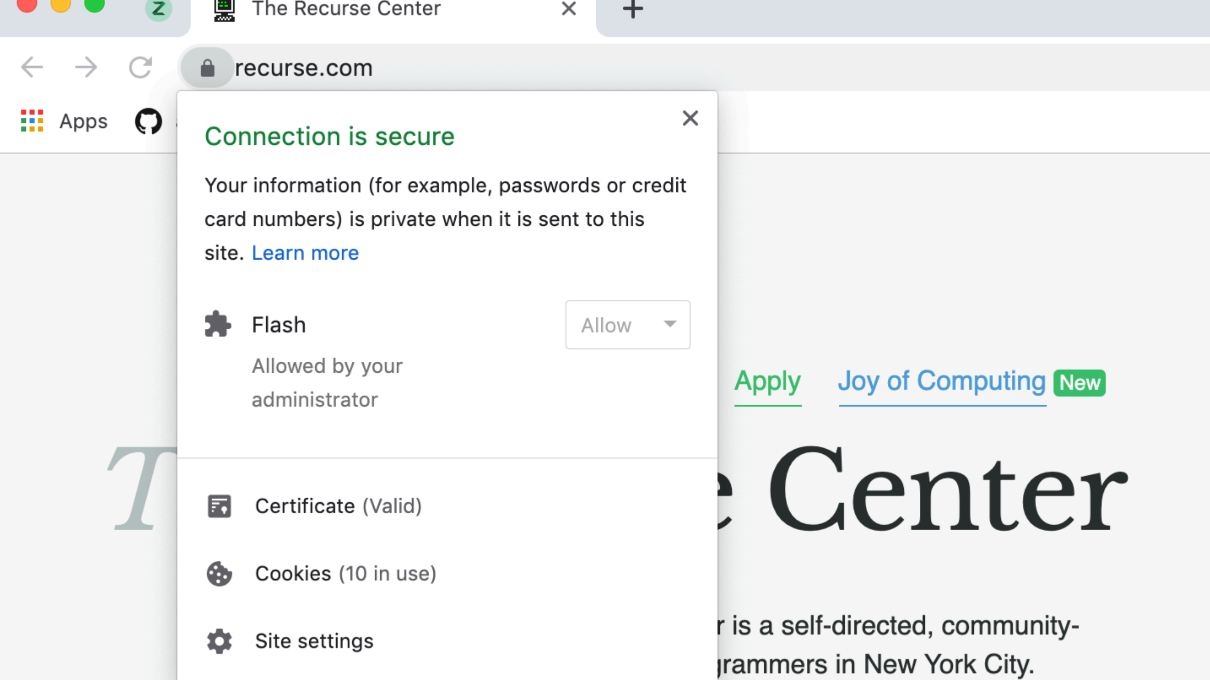
$$\mathcal{O}\left(\sqrt{p}\right)$$

Discrete log for Elliptic Curves is exponential.

Elliptic Curve Diffie-Hellman

- **Set:** points (x, y) on the elliptic curve
- **Operation:** repeated point addition kP
- **Security:** discrete log on elliptic curves has *exponential complexity*

Elliptic Curves are smaller and faster for the same security level.


The Recurse Center
recurse.com
Apps
Connection is secure
Your information (for example, passwords or credit card numbers) is private when it is sent to this site. Learn more
Flash
Allow
Allowed by your administrator
Certificate (Valid)
Cookies (10 in use)
Site settings
Apply
Joy of Computing
New
e Center
r is a self-directed, community-
grammers in New York City.

Elliptic Curve Cryptography (ECC) is very widely used in TLS.

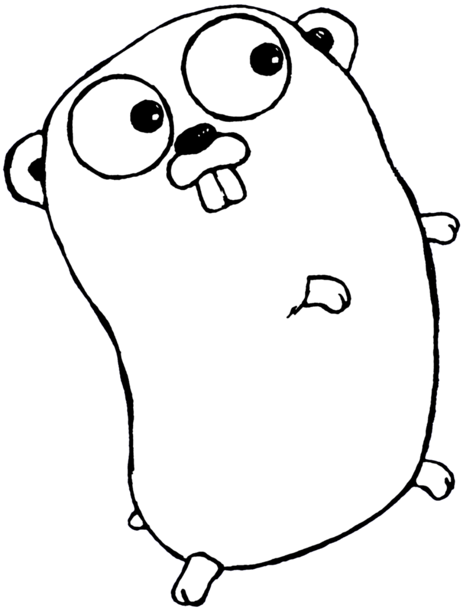
Implementation

Implementations must be secure.

Error-free and constant-time.

Implementations must be fast.

Low-level arithmetic almost always implemented in assembly.



crypto/elliptic/p256_asm_amd64.s

```
	MOVOU X3, (16*3)(DX)

	RET
/* ---------------------------------------*/
// func p256OrdMul(res, in1, in2 []uint64)
TEXT ·p256OrdMul(SB),NOSPLIT,$0
	MOVQ res+0(FP), res_ptr
	MOVQ in1+24(FP), x_ptr
	MOVQ in2+48(FP), y_ptr
	// x * y[0]
	MOVQ (8*0)(y_ptr), t0

	MOVQ (8*0)(x_ptr), AX
	MULQ t0
	MOVQ AX, acc0
	MOVQ DX, acc1
```

```
	MOVQ acc5, acc3
	MOVQ acc0, t0
	MOVQ acc1, t1
	// Subtract p256
	SUBQ p256ord<>+0x00(SB), acc4
	SBBQ p256ord<>+0x08(SB) ,acc5
	SBBQ p256ord<>+0x10(SB), acc0
	SBBQ p256ord<>+0x18(SB), acc1
	SBBQ $0, acc2

	CMOVQCS x_ptr, acc4
	CMOVQCS acc3, acc5
	CMOVQCS t0, acc0
	CMOVQCS t1, acc1

	MOVQ acc4, (8*0)(res_ptr)
```

```
/* ---------------------------------------*/
TEXT p256SqrInternal(SB),NOSPLIT,$0

	MOVQ acc4, mul0
	MULQ acc5
	MOVQ mul0, acc1
	MOVQ mul1, acc2

	MOVQ acc4, mul0
	MULQ acc6
	ADDQ mul0, acc2
	ADCQ $0, mul1
	MOVQ mul1, acc3

	MOVQ acc4, mul0
```

```
MOVL t1, sel_save
MOVL t2, zero_save
// Negate y2in based on sign
MOVQ (16*2 + 8*0)(CX), acc4
MOVQ (16*2 + 8*1)(CX), acc5
MOVQ (16*2 + 8*2)(CX), acc6
MOVQ (16*2 + 8*3)(CX), acc7
MOVQ $-1, acc0
MOVQ p256const0<>(SB), acc1
MOVQ $0, acc2
MOVQ p256const1<>(SB), acc3
XORQ mul0, mul0
// Speculatively subtract
SUBQ acc4, acc0
SBBQ acc5, acc1
SBBQ acc6, acc2
```

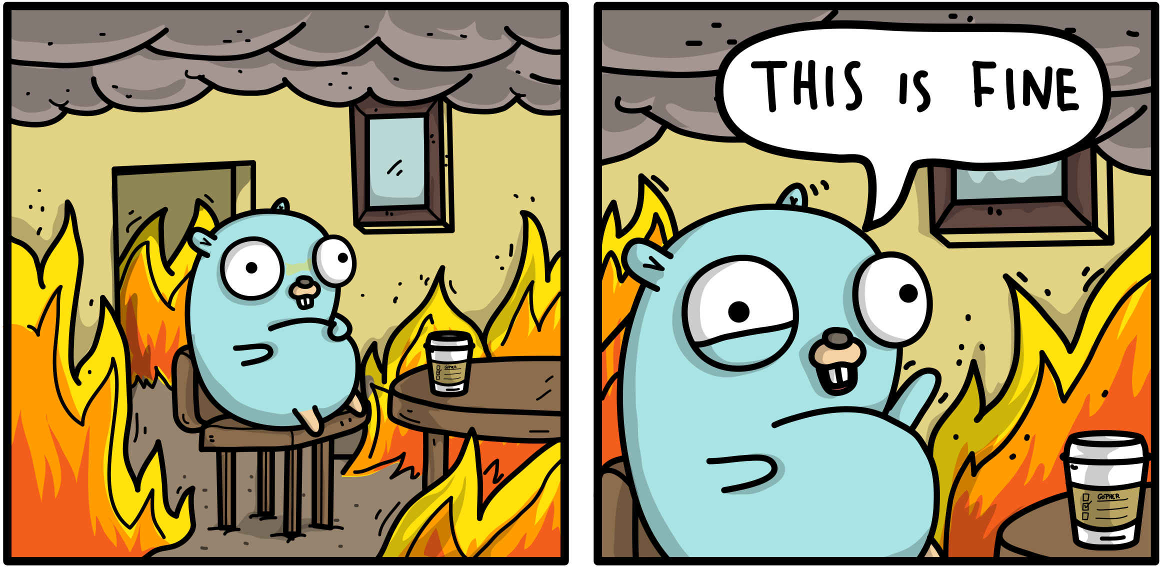
```
	LDt (y)
	CALL p256SubInternal(SB)
	MOVQ rptr, AX
	// Store y
	MOVQ acc4, (16*2 + 8*0)(AX)
	MOVQ acc5, (16*2 + 8*1)(AX)
	MOVQ acc6, (16*2 + 8*2)(AX)
	MOVQ acc7, (16*2 + 8*3)(AX)
	///////////////////////
	MOVQ $0, rptr

	RET
/* ---------------------------------------*/

```

```
        RET
/* ----------
```

Is this fine?

crypto/elliptic: carry bug in x86-64 P-256 #20040

Closed agl opened this issue on Apr 19, 2017 · 12 comments


THIS IS FINE
GOPHER

Elliptic Curve Cryptography Compiler

Most elliptic curve implementations are hand rolled.

It’s just f**king arithmetic!

Can we automate code-generation for elliptic curves?

ec3: Elliptic Curve Cryptography Compiler

ec3

Inputs:

- Curve type and equation
- Coordinate system
- Finite field implementation

Outputs assembly and Go code to implement the curve.

Status

Generates correct elliptic curve code for P-256.

```
DATA p<>+0(SB)/8, $0xffffffffffffffff
DATA p<>+8(SB)/8, $0x00000000ffffffff
DATA p<>+16(SB)/8, $0x0000000000000000
DATA p<>+24(SB)/8, $0xffffffff00000001
GLOBL p<>(SB), RODATA|NOPTR, $32

// func double(X1_ *Elt, X3_ *Elt, Y1_ *Elt, Y3_ *Elt, Z1_ *E
TEXT ·double(SB), $800-48
	MOVQ X1_+0(FP), BX
	MOVQ (BX), AX
	MOVQ 8(BX), CX
	MOVQ 16(BX), DX
	MOVQ 24(BX), BX
	MOVQ AX, 160(SP)
	MOVQ CX, 168(SP)
```

10106 lines (9230 sloc) | 170 KB

```
// Code generated by ec3. DO NOT EDIT.

#include "textflag.h"

// func lookup(p *Jacobian, tbl []Jacobian, idx int)
TEXT ·lookup(SB), $0-40
        MOVQ p+0(FP), AX
        MOVQ tbl_base+8(FP), CX
        MOVQ tbl_len+16(FP), DX
        MOVQ idx+32(FP), BX

        // Initialize a 1 register.
        PXOR    X0, X0
```

Performance

```
pkg: github.com/mmcloughlin/ec3/examples/p256
BenchmarkScalarMult-4       136772   89073 ns/op
BenchmarkStdScalarMult-4    213002   56462 ns/op
```

Thank you!

@mbmcloughlin