Squeezing a key through a carry bit

Sean Devlin, Filippo Valsorda

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

Closed **agl** opened this issue on Apr 19 · 11 comments

agl commented on Apr 19 Member

Cloudflare reported a carry bug in the P-256 implementation that they submitted for x86-64 in `7bacfc6`. I can reproduce this via random testing against BoringSSL and, after applying the patch that they provided, can no longer do so, even after ~2^{31} iterations.

This issue is not obviously exploitable, although we cannot rule out the possibility of someone managing to squeeze something through this hole. (It would be a cool paper.) Thus this should be treated as something to fix, but not something on fire, based on what we currently know.

Fix will be coming in just a second.

👍 19

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

Closed agl opened this issue on Apr 19 · 11 comments

agl commented on Apr 19 Member

Cloudflare reported a carry bug in the P-256 implementation that they submitted for x86-64 in `7bacfc6`. I can reproduce this via random testing against BoringSSL and, after applying the patch that they provided, can no longer do so, even after ~2^{31} iterations.

This issue is not obviously exploitable, although we cannot rule out the possibility of someone managing to squeeze something through this hole. (It would be a cool paper.) Thus this should be treated as something to fix, but not something on fire, based on what we currently know.

Fix will be coming in just a second.

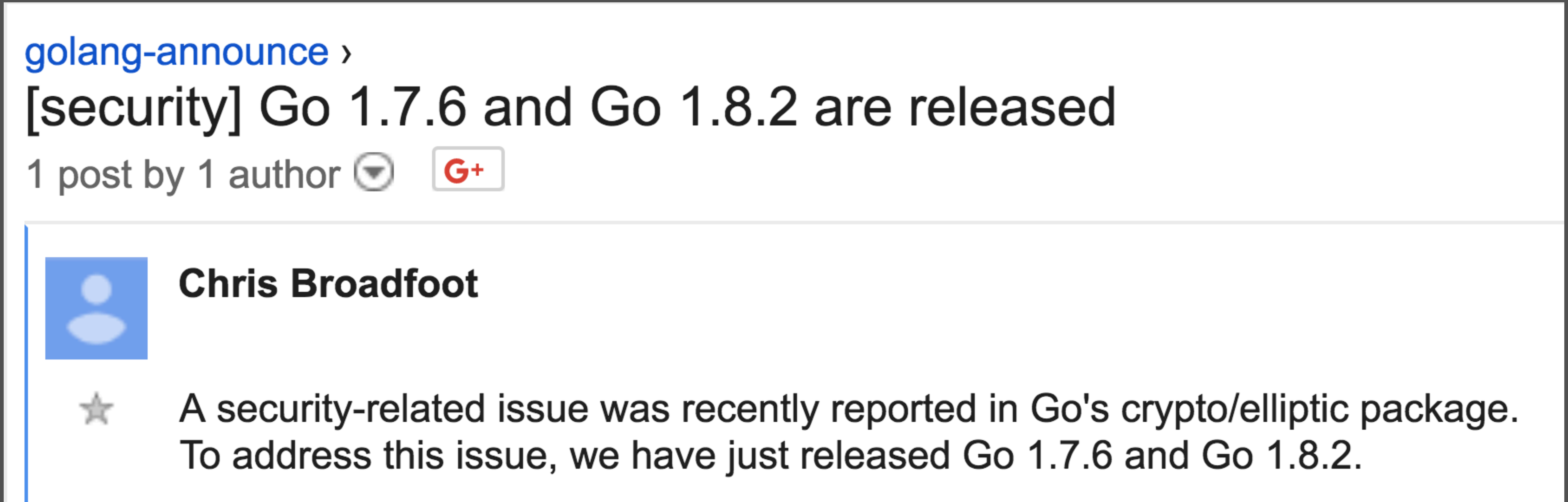
👍 19

One month later

agl commented on May 23

Member

(This issue is CVE-2017-8932.)



golang-announce ›

[security] Go 1.7.6 and Go 1.8.2 are released

1 post by 1 author

Chris Broadfoot

A security-related issue was recently reported in Go's crypto/elliptic package.
To address this issue, we have just released Go 1.7.6 and Go 1.8.2.

The code

a = a - b mod p

```
TEXT p256SubInternal(SB),NOSPLIT,$0
    XORQ mul0, mul0
    SUBQ t0, acc4
    SBBQ t1, acc5
    SBBQ t2, acc6
    SBBQ t3, acc7
    SBBQ $0, mul0

    MOVQ acc4, acc0
    MOVQ acc5, acc1
    MOVQ acc6, acc2
    MOVQ acc7, acc3

    ADDQ $-1, acc4
    ADCQ p256const0<>(SB), acc5
    ADCQ $0, acc6
    ADCQ p256const1<>(SB), acc7

    ADCQ $0, mul0

    CMOVQNE acc0, acc4
    CMOVQNE acc1, acc5
    CMOVQNE acc2, acc6
    CMOVQNE acc3, acc7

    RET
```

The code

$a = a - b \bmod p$

```
TEXT p256SubInternal(SB),NOSPLIT,$0
	XORQ mul0, mul0
	SUBQ t0, acc4
	SBBQ t1, acc5
	SBBQ t2, acc6
	SBBQ t3, acc7
	SBBQ $0, mul0

	MOVQ acc4, acc0
	MOVQ acc5, acc1
	MOVQ acc6, acc2
	MOVQ acc7, acc3

	ADDQ $-1, acc4
	ADCQ p256const0<>(SB), acc5
	ADCQ $0, acc6
	ADCQ p256const1<>(SB), acc7

	ADCQ $0, mul0

	CMOVQNE acc0, acc4
	CMOVQNE acc1, acc5
	CMOVQNE acc2, acc6
	CMOVQNE acc3, acc7

	RET
```

```
a = a - b

t = a

    t += p

a ?= t
```

The code

$a = a - b \mod p$

```
TEXT p256SubInternal(SB),NOSPLIT,$0
	XORQ mul0, mul0
	SUBQ t0, acc4
	SBBQ t1, acc5
	SBBQ t2, acc6
	SBBQ t3, acc7
	SBBQ $0, mul0

	MOVQ acc4, acc0
	MOVQ acc5, acc1
	MOVQ acc6, acc2
	MOVQ acc7, acc3

	ADDQ $-1, acc4
	ADCQ p256const0<>(SB), acc5
	ADCQ $0, acc6
	ADCQ p256const1<>(SB), acc7

	ADCQ $0, mul0

	CMOVQNE acc0, acc4
	CMOVQNE acc1, acc5
	CMOVQNE acc2, acc6
	CMOVQNE acc3, acc7

	RET
```

$a = a - b$

$a < b$

$t = a$

$t \mathrel{+}= p$

$a \overset{?}{=} t$

The bug

```
TEXT p256SubInternal(SB),NOSPLIT,$0
     XORQ mul0, mul0
     SUBQ t0, acc4
     SBBQ t1, acc5
     SBBQ t2, acc6
     SBBQ t3, acc7
     SBBQ $0, mul0

     MOVQ acc4, acc0
     MOVQ acc5, acc1
     MOVQ acc6, acc2
     MOVQ acc7, acc3

     ADDQ $-1, acc4
     ADCQ p256const0<>(SB), acc5
     ADCQ $0, acc6
     ADCQ p256const1<>(SB), acc7

     ADCQ $0, mul0

     CMOVQNE acc0, acc4
     CMOVQNE acc1, acc5
     CMOVQNE acc2, acc6
     CMOVQNE acc3, acc7

     RET
```

a = a - b

t = a

t += p

a ?= t

The bug

```
 TEXT p256SubInternal(SB),NOSPLIT,$0
    XORQ mul0, mul0
    SUBQ t0, acc4
    SBBQ t1, acc5
    SBBQ t2, acc6
    SBBQ t3, acc7
    SBBQ $0, mul0

    MOVQ acc4, acc0
    MOVQ acc5, acc1
    MOVQ acc6, acc2
    MOVQ acc7, acc3

    ADDQ $-1, acc4
    ADCQ p256const0<>(SB), acc5
    ADCQ $0, acc6
    ADCQ p256const1<>(SB), acc7

-   ADCQ $0, mul0
+   ANDQ $1, mul0

-   CMOVQNE acc0, acc4
-   CMOVQNE acc1, acc5
-   CMOVQNE acc2, acc6
-   CMOVQNE acc3, acc7
+   CMOVQEQ acc0, acc4
+   CMOVQEQ acc1, acc5
+   CMOVQEQ acc2, acc6
+   CMOVQEQ acc3, acc7

    RET
```

The bug

```
 TEXT p256SubInternal(SB),NOSPLIT,$0
	XORQ mul0, mul0
	SUBQ t0, acc4
	SBBQ t1, acc5
	SBBQ t2, acc6
	SBBQ t3, acc7
	SBBQ $0, mul0

	MOVQ acc4, acc0
	MOVQ acc5, acc1
	MOVQ acc6, acc2
	MOVQ acc7, acc3

	ADDQ $-1, acc4
	ADCQ p256const0<>(SB), acc5
	ADCQ $0, acc6
	ADCQ p256const1<>(SB), acc7

-	ADCQ $0, mul0
+	ANDQ $1, mul0

-	CMOVQNE acc0, acc4
-	CMOVQNE acc1, acc5
-	CMOVQNE acc2, acc6
-	CMOVQNE acc3, acc7
+	CMOVQEQ acc0, acc4
+	CMOVQEQ acc1, acc5
+	CMOVQEQ acc2, acc6
+	CMOVQEQ acc3, acc7

	RET
```

Wrong result with probability 2^{-32}

A carry propagation bug

ECCCCCCC

Elliptic Curve Cryptography Crash Course for CCC

- Field: numbers modulo p
- Points: like (3, 7); fitting an equation
- Group: a generator point and addition
- Multiplication: repeated addition

ECCCCCCCC

Elliptic Curve Cryptography Crash Course for CCC (cont.)

- Multiplication: $5Q = Q + Q + Q + Q + Q$
- ECDH private key: a big integer d
- ECDH public key: $Q = dG$ (think $y = g^a$)
- ECDH shared secret: $Q_2 = dQ_1$

Double and add

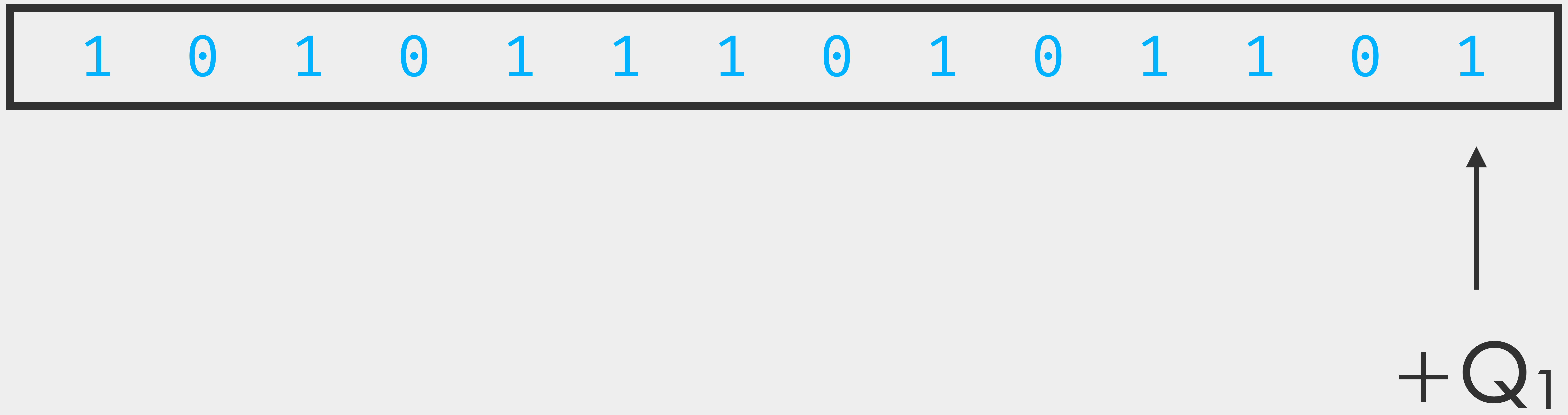
$$Q_2 = dQ_1$$

d is BIG. Like, 256 bit.

Can't add Q to itself 2^{256} times.

Double and add

$$Q_2 = dQ_1$$



$Z + Q$

Double and add

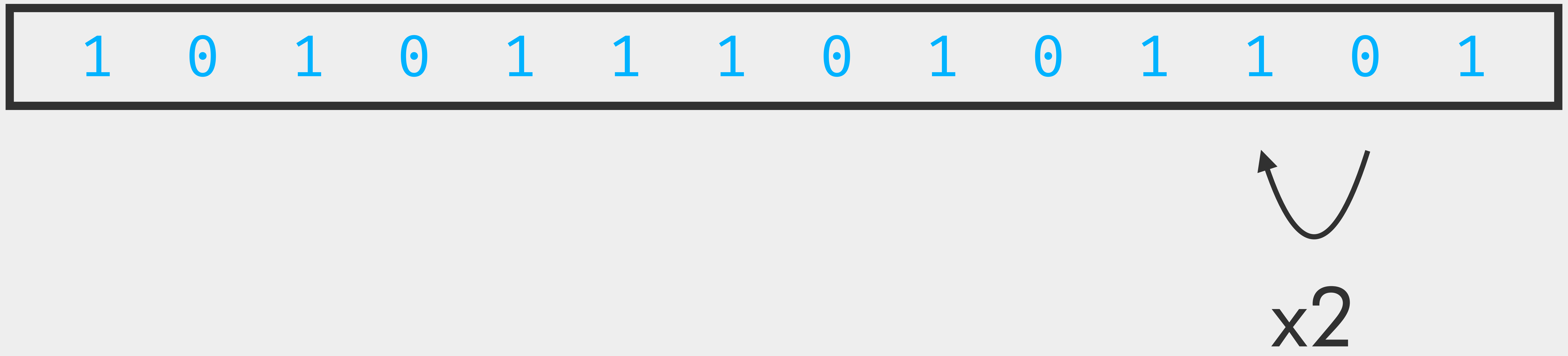
$Q_2 = dQ_1$

1	0	1	0	1	1	1	0	1	0	1	1	0	1

Z +Q x2

Double and add

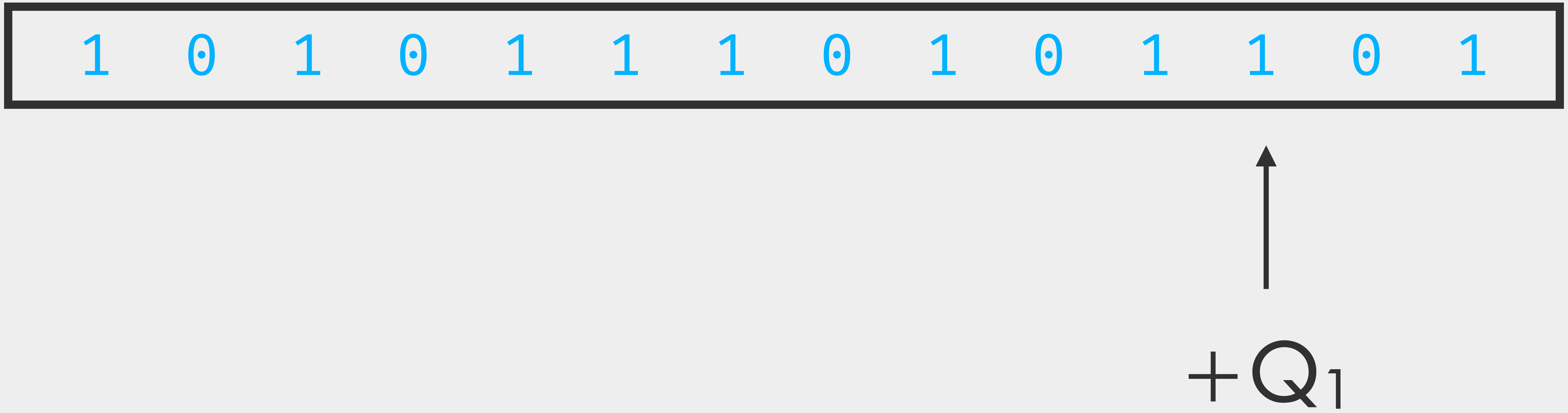
$$Q_2 = dQ_1$$



Z +Q x2 x2

Double and add

$$Q_2 = dQ_1$$



Z +Q x2 x2 +Q

Double and add

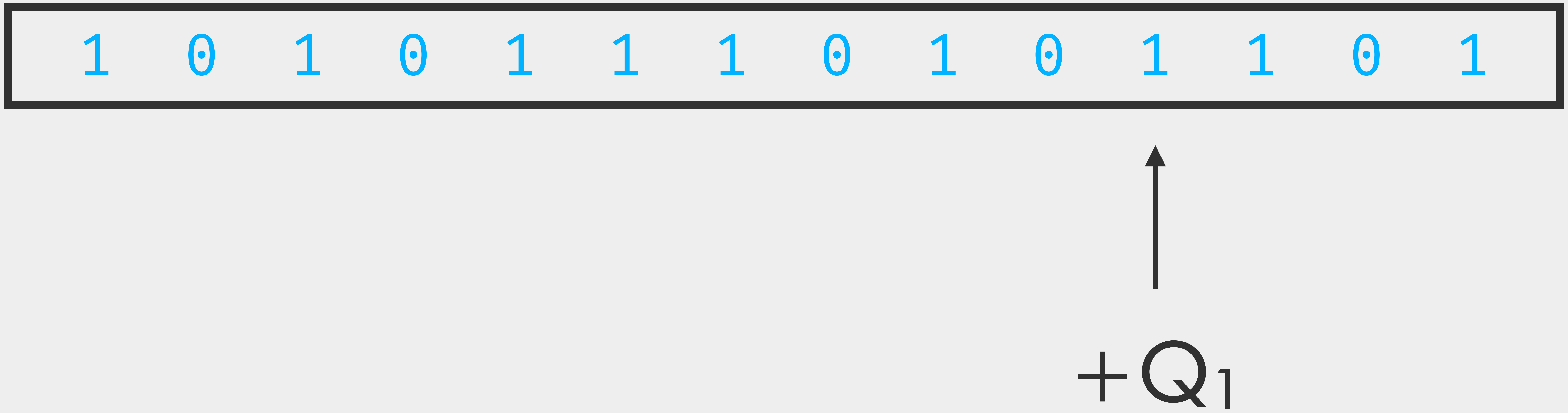
$$Q_2 = dQ_1$$

`1 0 1 0 1 1 1 0 1 0 1 1 0 1`

Z +Q x2 x2 +Q x2

Double and add

$Q_2 = dQ_1$



Z +Q x2 x2 +Q x2 +Q

Double and add

$$Q_2 = dQ_1$$

`1 0 1 0 1 1 1 0 1 0 1 1 0 1`

Z +Q x2 x2 +Q x2 +Q x2

Double and add

$$Q_2 = dQ_1$$

$$\boxed{1\quad 0\quad 1\quad 0\quad 1\quad 1\quad 1\quad 0\quad 1\quad 0\quad 1\quad 1\quad 0\quad 1}$$

Z +Q x2 x2 +Q x2 +Q x2 x2 ...

Back to the carry bug

session key attacker supplied secret key

secret = ScalarMult(point, scalar) ← $Q_2 = dQ$

└── p256PointAddAffineAsm

  └── p256SubInternal 💥

$Q_1 \rightarrow \text{ScalarMult}(Q_1, \boxed{1\ \ 1\ \ 1\ \ 0\ \ 1})$

$Z + Q_1 \times 2 \times 2 + Q_1 \times 2 + Q_1 \times 2 + Q_1$ 💥

$Q_2 \rightarrow \text{ScalarMult}(Q_2, \boxed{0\ \ 1\ \ 1\ \ 0\ \ 1})$

$Z + Q_2 \times 2 \times 2 + Q_2 \times 2 + Q_2 \times 2 \times 2$ 💥

$Q_1 \rightarrow \text{ScalarMult}(Q_1, \boxed{\texttt{? 1 1 0 1}}) \rightarrow$ 💥

$Q_2 \rightarrow \text{ScalarMult}(Q_2, \boxed{\texttt{? 1 1 0 1}}) \rightarrow$ ✅

$$\downarrow$$

$$\boxed{\texttt{1 1 1 0 1}}$$

$Q_1 \rightarrow$			0	1	1	0	1	💥
$Q_2 \rightarrow$			1	1	1	0	1	
$Q_1 \rightarrow$		0	0	1	1	0	1	
$Q_2 \rightarrow$		1	0	1	1	0	1	💥
$Q_1 \rightarrow$	0	1	0	1	1	0	1	
$Q_2 \rightarrow$	1	1	0	1	1	0	1	

Practical realisation and elimination of an ECC-related software bug attack*

B. B. Brumley[1], M. Barbosa[2], D. Page[3], and F. Vercauteren[4]

[1] Department of Information and Computer Science,
Aalto University School of Science, P.O. Box 15400, FI-00076 Aalto, Finland.
billy.brumley@aalto.fi
[2] HASLab/INESC TEC
Universidade do Minho, Braga, Portugal.
mbb@di.uminho.pt
[3] Department of Computer Science, University of Bristol,
Merchant Venturers Building, Woodland Road, Bristol, BS8 1UB, UK.
page@cs.bris.ac.uk
[4] Department of Electrical Engineering, Katholieke Universiteit Leuven,
Kasteelpark Arenberg 10, B-3001 Leuven-Heverlee, Belgium.
fvercaut@esat.kuleuven.ac.be

Go implementation of ScalarMult

Booth's multiplication in 5-bit windows.

Precomputed table of 1Q to 16Q. Add, double 5 times.

```
01 00010 01110 01010 01010 10010 00001 01111 10011 01101 ...
```

Precomp table

```
func (p *p256Point) p256ScalarMult(scalar []uint64) {
    // precomp is a table of precomputed points that stores powers of p
    // from p^1 to p^16.
    var precomp [16 * 4 * 3]uint64
    var t0, t1, t2, t3 p256Point

    // Prepare the table
    p.p256StorePoint(&precomp, 0) // 1

    p256PointDoubleAsm(t0.xyz[:], p.xyz[:])
    p256PointDoubleAsm(t1.xyz[:], t0.xyz[:])
    p256PointDoubleAsm(t2.xyz[:], t1.xyz[:])
    p256PointDoubleAsm(t3.xyz[:], t2.xyz[:])
    t0.p256StorePoint(&precomp, 1)  // 2
    t1.p256StorePoint(&precomp, 3)  // 4
    t2.p256StorePoint(&precomp, 7)  // 8
    t3.p256StorePoint(&precomp, 15) // 16

    p256PointAddAsm(t0.xyz[:], t0.xyz[:], p.xyz[:])
    p256PointAddAsm(t1.xyz[:], t1.xyz[:], p.xyz[:])
    p256PointAddAsm(t2.xyz[:], t2.xyz[:], p.xyz[:])
    t0.p256StorePoint(&precomp, 2) // 3
    t1.p256StorePoint(&precomp, 4) // 5
    t2.p256StorePoint(&precomp, 8) // 9

    p256PointDoubleAsm(t0.xyz[:], t0.xyz[:])
    p256PointDoubleAsm(t1.xyz[:], t1.xyz[:])
    t0.p256StorePoint(&precomp, 5) // 6
    t1.p256StorePoint(&precomp, 9) // 10

    p256PointAddAsm(t2.xyz[:], t0.xyz[:], p.xyz[:])
    p256PointAddAsm(t1.xyz[:], t1.xyz[:], p.xyz[:])
    t2.p256StorePoint(&precomp, 6)  // 7
    t1.p256StorePoint(&precomp, 10) // 11

    p256PointDoubleAsm(t0.xyz[:], t0.xyz[:])
    p256PointDoubleAsm(t2.xyz[:], t2.xyz[:])
    t0.p256StorePoint(&precomp, 11) // 12
    t2.p256StorePoint(&precomp, 13) // 14

    p256PointAddAsm(t0.xyz[:], t0.xyz[:], p.xyz[:])
    p256PointAddAsm(t2.xyz[:], t2.xyz[:], p.xyz[:])
    t0.p256StorePoint(&precomp, 12) // 13
    t2.p256StorePoint(&precomp, 14) // 15
```

Multiplication loop

```
for index > 4 {
    index -= 5
    p256PointDoubleAsm(p.xyz[:], p.xyz[:])
    p256PointDoubleAsm(p.xyz[:], p.xyz[:])
    p256PointDoubleAsm(p.xyz[:], p.xyz[:])
    p256PointDoubleAsm(p.xyz[:], p.xyz[:])
    p256PointDoubleAsm(p.xyz[:], p.xyz[:])

    if index < 192 {
        wvalue = ((scalar[index/64] >> (index % 64)) + (scalar[in
    } else {
        wvalue = (scalar[index/64] >> (index % 64)) & 0x3f
    }

    sel, sign = boothW5(uint(wvalue))

    p256Select(t0.xyz[0:], precomp[0:], sel)
    p256NegCond(t0.xyz[4:8], sign)
    p256PointAddAsm(t1.xyz[:], p.xyz[:], t0.xyz[:])
    p256MovCond(t1.xyz[0:12], t1.xyz[0:12], p.xyz[0:12], sel)
    p256MovCond(p.xyz[0:12], t1.xyz[0:12], t0.xyz[0:12], zero)
    zero |= sel
}
```

Go implementation of ScalarMult

Booth's multiplication in 5-bit windows.

Precomputed table of 1Q to 16Q. Add, double 5 times.

```
01 00010 01110 01010 01010 10010 00001 01111 10011 01101 ...
```

Limbs representation: less overlap and aliasing problems.

```
{1 0} {15 1} {7 0} {5 0} {5 0} {9 0} {1 0} {8 1} {6 1} {9 1} ...
```

Go implementation of ScalarMult

Booth's multiplication in 5-bit windows.

Precomputed table of 1Q to 16Q. Add, double 5 times.

```
01 00010 01110 01010 01010 10010 00001 01111 10011 01101 ...
```

Attack one limb at a time, instead of one bit.

34 limb values → 17 points / 5 key bits on average.

Multiplication loop

```
for index > 4 {
    index -= 5
    p256PointDoubleAsm(p.xyz[:], p.xyz[:])
    p256PointDoubleAsm(p.xyz[:], p.xyz[:])
    p256PointDoubleAsm(p.xyz[:], p.xyz[:])   💥
    p256PointDoubleAsm(p.xyz[:], p.xyz[:])
    p256PointDoubleAsm(p.xyz[:], p.xyz[:])

    if index < 192 {
        wvalue = ((scalar[index/64] >> (index % 64)) + (scalar[i
    } else {
        wvalue = (scalar[index/64] >> (index % 64)) & 0x3f
    }

    sel, sign = boothW5(uint(wvalue))

    p256Select(t0.xyz[0:], precomp[0:], sel)
    p256NegCond(t0.xyz[4:8], sign)
💥  p256PointAddAsm(t1.xyz[:], p.xyz[:], t0.xyz[:])
    p256MovCond(t1.xyz[0:12], t1.xyz[0:12], p.xyz[0:12], sel)
    p256MovCond(p.xyz[0:12], t1.xyz[0:12], t0.xyz[0:12], zero)
    zero |= sel
}
```

Assembly hook

```
TEXT paris256SubInternal(SB), NOSPLIT, $0
    XORQ mul0, mul0
    SUBQ t0, acc4
    SBBQ t1, acc5
    SBBQ t2, acc6
    SBBQ t3, acc7
    SBBQ $0, mul0

    MOVQ acc4, acc0
    MOVQ acc5, acc1
    MOVQ acc6, acc2
    MOVQ acc7, acc3

    ADDQ $-1, acc4
    ADCQ p256const0<>(SB), acc5
    ADCQ $0, acc6
    ADCQ p256const1<>(SB), acc7

    // Paris256: if the carry bit is clear, the bug would be triggered.
    SBBQ hlp, hlp
    SUBQ hlp, mul0 // was: ADCQ $0, mul0; but we stole the carry bit above
    XORQ $-1, hlp
    ANDQ $1, hlp

    ADDQ $0, mul0

    CMOVQNE acc0, acc4
    CMOVQNE acc1, acc5
    CMOVQNE acc2, acc6
    CMOVQNE acc3, acc7

    RET
```

💥

```go
func (t *paris256Trace) Fuzz(precomp [16 * 4 * 3]uint64, prev limb, zero int, pp *p256Point) {
    var t0 p256Point
    for _, b := range boothSpace {
        p := pp
        if b.Sel == 0 && zero == 0 {
            // If this round, the one before, and all the ones before are 0,
            // all the operations are discarded. Spot this by exclusion.
            continue
        } else if zero == 0 { // p = {-sign}precomp[sel]
            p256Select(t0.xyz[0:], precomp[0:], b.Sel)
            p256NegCond(t0.xyz[4:8], b.Sign)
            p = &t0
        } else if b.Sel != 0 { // p = p + {-sign}precomp[sel]
            p256Select(t0.xyz[0:], precomp[0:], b.Sel)
            p256NegCond(t0.xyz[4:8], b.Sign)
            t.X("fuzz-add", paris256PointAddAsm(t0.xyz[:], pp.xyz[:], t0.xyz[:]), b.Sel, true)
            p = &t0
        } // else p = p

        t.X("fuzz-double-1", paris256PointDoubleAsm(t0.xyz[:], p.xyz[:]), b.Sel, true)
        t.X("fuzz-double-2", paris256PointDoubleAsm(t0.xyz[:], t0.xyz[:]), b.Sel, true)
        t.X("fuzz-double-3", paris256PointDoubleAsm(t0.xyz[:], t0.xyz[:]), b.Sel, true)
        t.X("fuzz-double-4", paris256PointDoubleAsm(t0.xyz[:], t0.xyz[:]), b.Sel, true)
        t.X("fuzz-double-5", paris256PointDoubleAsm(t0.xyz[:], t0.xyz[:]), b.Sel, true)
    }
}
```

```
func nextPoint(dlog []byte, bigX, bigY *big.Int) (x, y *big.Int) {
    for i := range dlog {
        dlog[len(dlog)-1-i] += 1
        if dlog[len(dlog)-1-i] ≠ 0 {
            break
        }
    }

    p := p256PointFromAffine(bigX, bigY)
    p256PointAddAsm(p.xyz[:], p.xyz[:], basePoint)
    return p.p256PointToAffine()
}

// Import the patched p256PointAddAsm.
//go:linkname p256PointAddAsm crypto/elliptic.p256PointAddAsm

func p256PointAddAsm(res, in1, in2 []uint64)
```

```
06e3634359a8a4077f5770e39ba3502ebef6ec56644c86c1dbe4cedf898bbae9:ooooooooooooooXoooooooooooooooooo
0804f5c147053a1e53ff9204eb00677d55d1ded582d85c3b4c3a6be161061831:ooooooooooooooooooooooooooXoooooo
09f88cc112f8eaaf9f5ea5b05855fc04615652df1f44f14e562928b0476eda93:oooooXooooooooooooooooooooooooooo
0acee85067262b9bcfab64a39a6a1ed5220e445914e6403b1bd4b01e6a379578:ooooooooooooooooooXoooooooooooooo
0d142ba2ca895fe22e147f42a6e52e26ed1a5d0ad91d67466d374e29e28b14d5:ooooooooooooooooooooXoooooooooooo
0fcf4cf46cf88a3d229060c6034f0e8be0b8e79b3540d41a9379de19a437273c:ooooooooooooooooooooooooooooooXoo
26d97cf1e6144c729ef6ff72e1c8f31fbb0a22627058ef01e9e3bdbcf849fb92:oXooooooooooooooooooooooooooooooo
4b427c6d8d777dbfa6cf64cdc63e301e97e324df023749ef6384989ab615c52b:Xoooooooooooooooooooooooooooooooo
52061d542ad5578004c7b0b334f65dc75489f73483c6aefa9b9459e7b03f4aba:oooooooooooooooooooooooooooXooooo
53d1800629e891013c98edcc6c04516a23d18f04760bd03d75e2a106076ae396:ooooooooooooooooooooooooooooXoooo
554bd89958286e458e5bf966fe2a369c1b9aa3a29a4c04d81cdbdf385fa382a0:ooooooooooooooooooooooooooooooooX
620d527b80bd2f6a513bb88b2b6c492983391d73ce325b15d8307fbd8c3df079:oooooooooooooooooooooooooXooooooo
6335a174ca8240114ccc160b86c15d51ab11d74910ca3ca00a6aca9b0ed3ea6a:oooooooooooooXooooooooooooooooooo
6db73c2a1770e130bc008a9644b6c706725e02c1025d2825f87114185634e4d6:ooooooooXoooooooooooooooooooooooo
758e784d8b8f88f40c7e6b0df9a60e24af0bb2628ce7533f81eb78351ccf7a41:ooXoooooooooooooooooooooooooooooo
8e8c6f17318f4ce4d2c8da27d198b39e516d72273d43d513ee93c26489f9db1b:oooooooooooooooooooooXooooooooooo
902fa900ff59048d96d4d4b959dc11a2118b0a3e1122b33d39a42ad10b766758:oooooooooooooooooooooooXooooooooo
90edaec27b7f9d7474d1de35ea0d47c873a7eb6e226c0e0346158e583117ec15:oooXooooooooooooooooooooooooooooo
93adbefec85ba25cccdaa307df4c0089523a6e1449ba9b04e9670aeedcb8d37c:oooooooooooooooooXooooooooooooooo
97fbee1ce3ac3afeb4f5faa716cdd777d66382c5b102cb1549d68dfb9fd7b54f:ooooooooooooooooooooooooXoooooooo
9dcd924931f9ff6692d73c7c9428a10557f78a1dc7ab9aadf9576f0c4a312e04:oooooooooooXooooooooooooooooooooo
a3843c860a536513b4633f015a7876bf8b941328579fa0daeb0e33efa6207b2b:ooooooooooooooooXoooooooooooooooo
aa9258f89744f258974f3a29273655b95ffe2d6b1916de2df9e5c64e3315c3fc:oooooooXooooooooooooooooooooooooo
aad62057d4bce85343dbf46ce0c5829173259d02945bd5c249a553722fd829f2:ooooXoooooooooooooooooooooooooooo
acf23c097954b9a96464563f3152fd26d717770bcb0b993898350825986c2176:ooooooooooooXoooooooooooooooooooo
b0c6621294f9fcc6c3819f62695716a1c29d3312741ca3378dd0f5382a1cfce5:oooooooooooooooooooXooooooooooooo
b1d1b9691e4050f39d22b4fb54faf90e94df7079d8e22d3129c9c488ed04edc2:oooooooooooooooXooooooooooooooooo
bf55b34af182ea1bbe9f4f4ec3a6d174fca8555e8b0f7908135a247b9eb22419:ooooooooooXoooooooooooooooooooooo
ce421de34c6c8b8ec1f6cf50b1077a6a0f736e40cfe60cb00669034f75cc189a:oooooooooooooooooooooooooooooooXo
d4f4860079f824e11b9d2f6f51ee3da84a2fe469ff6febd633bb615001209fbd:oooooooooXooooooooooooooooooooooo
d8eaf1851f30737620e0dfa9e339a3029c82c103708e43ad802aebe4e612ae0b:oooooooooooooooooooooooooooooXooo
```

The first limb

Precomp	Limb	Doubling		
3	3	x2 x2 x2 x2 x2	→	3 $x2^5$
		💥		

The first limb

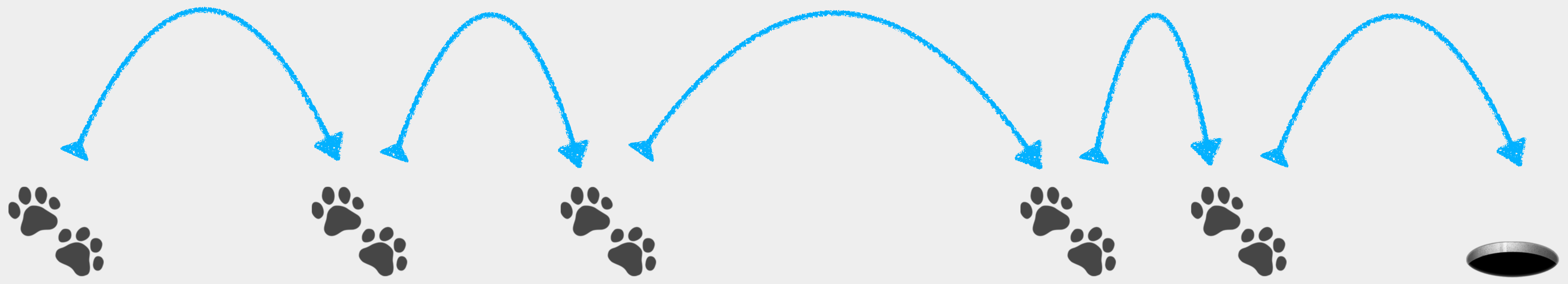
Precomp	Limb	Doubling						
3	3	x2	x2	x2	x2	x2	→	3 $x2^5$
						💥		
3 x2	6	x2	x2	x2	x2	x2	→	3 $x2^6$
					💥			
3 x2 x2	12	x2	x2	x2	x2	x2	→	3 $x2^7$
				💥				

The first limb

Precomp	Limb	Doubling		
3	3	x2 x2 x2 x2 x2 💥	→	3 x2^{5}
3 x2	6	x2 x2 x2 x2 x2 💥 🔥	→	3 x2^{6}
3 x2 x2	12	x2 x2 x2 x2 x2 💥 🔥 💣	→	3 x2^{7}

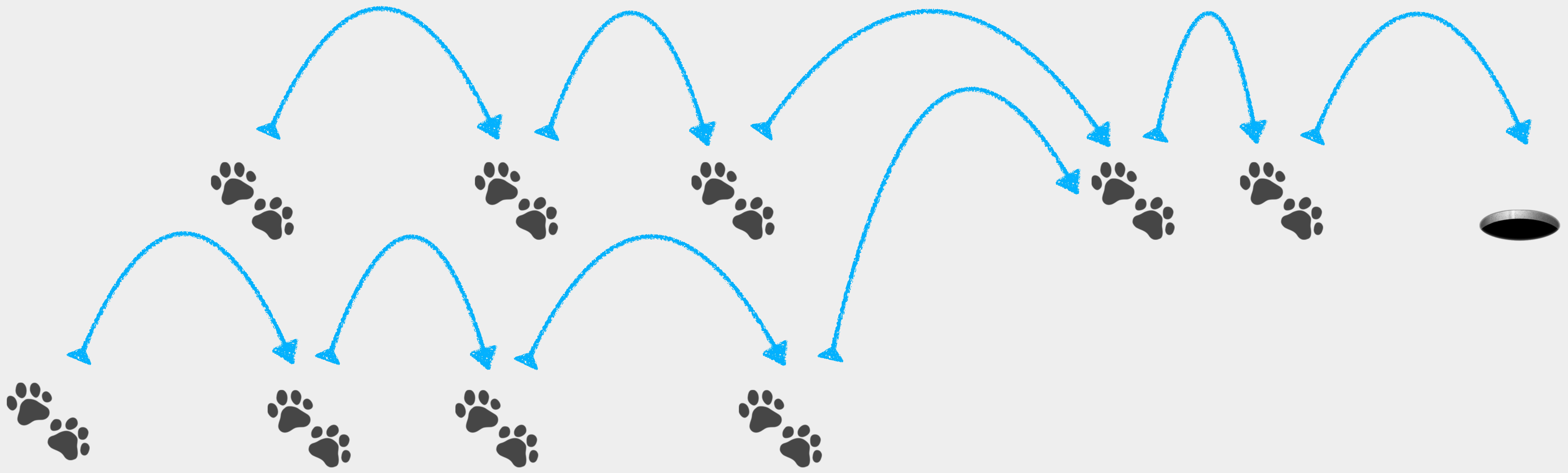
The last bits

Kangaroo jumps depend from the terrain at the start point.



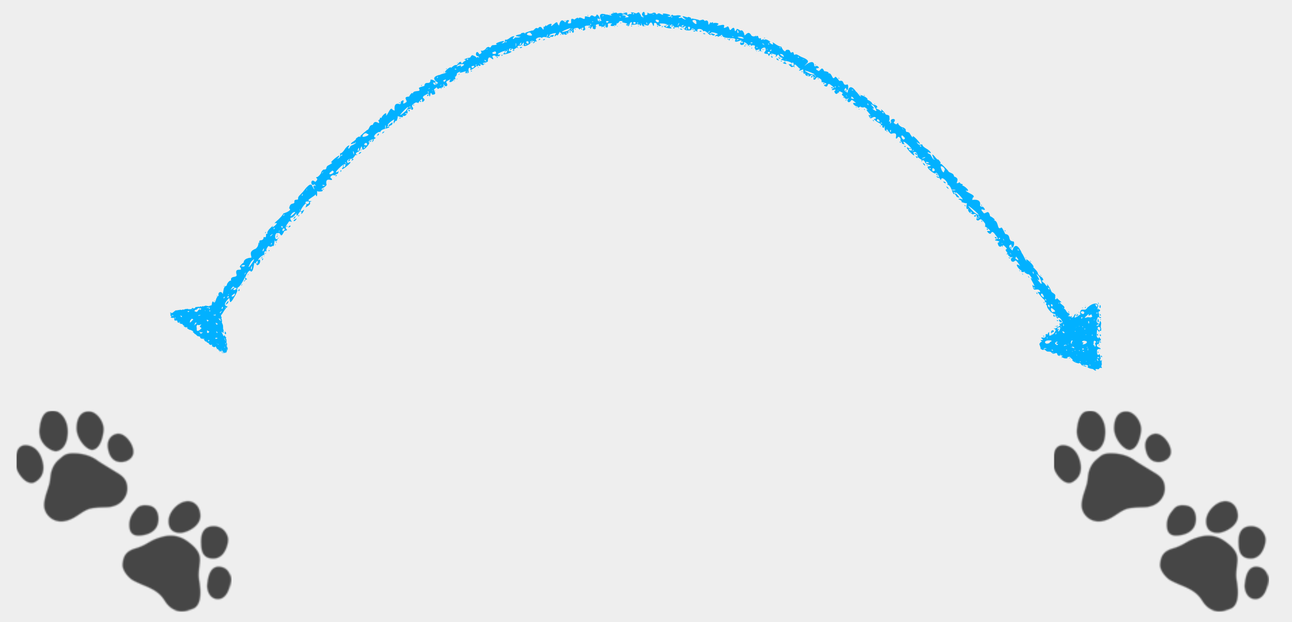
Let a tracked kangaroo loose. Place a trap at the end.

Kangaroo jumps depend from the terrain at the start point.



If the wild kangaroo intersects the path at any point,
it ends up in the trap.

Back to elliptic curves.



A jump is $Q_{N+1} = Q_N + H(Q_N)$ where H is a hash.

Same starting point, same jump.

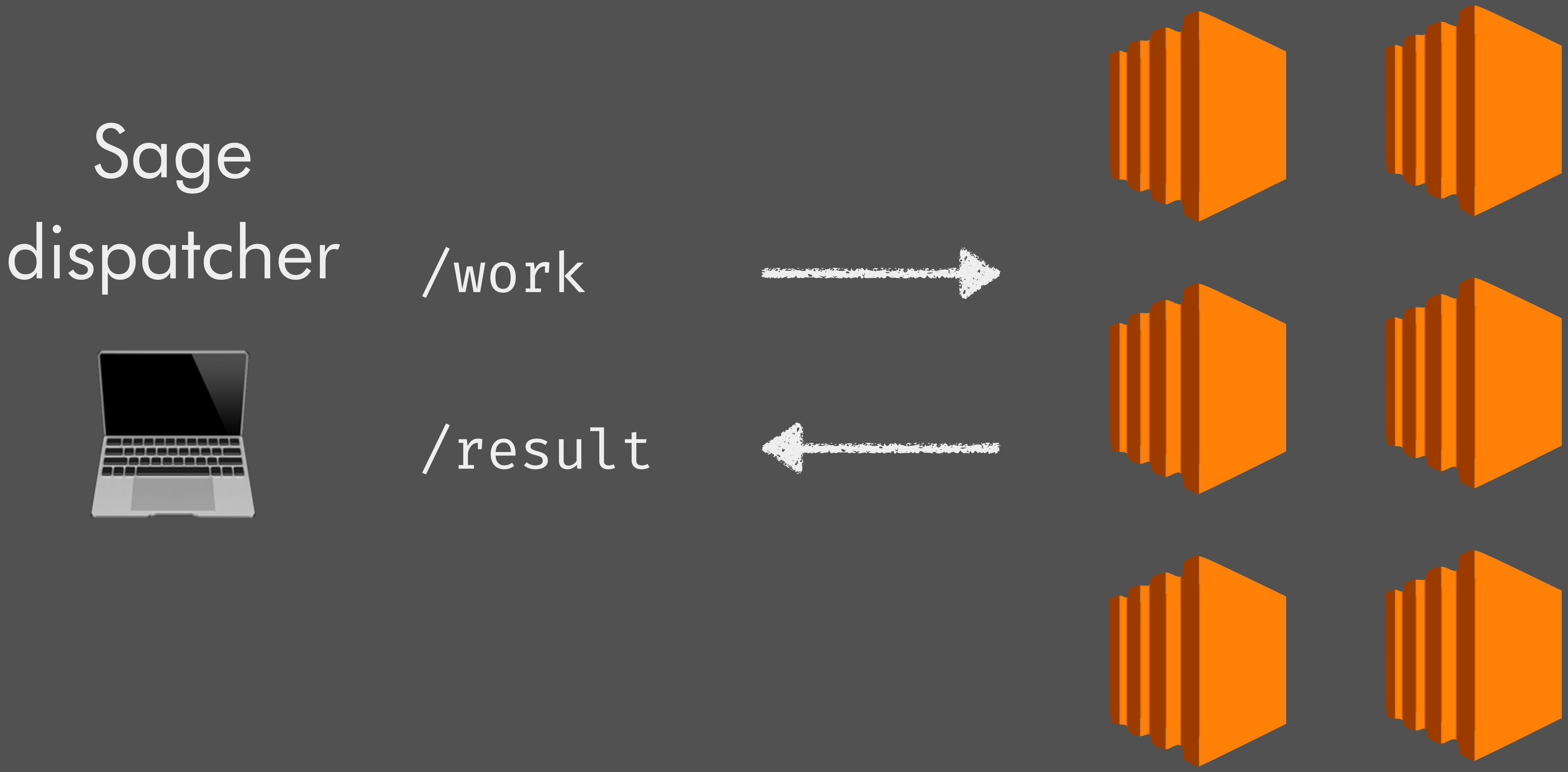
You run from a known starting point, then from dG.
If you collide, you traceback to d!

A target

- JSON Object Signing and Encryption, JOSE (JWT)
- ECDH-ES public key algorithm
- go-jose and Go 1.8.1

- Check if the service successfully decrypts payload

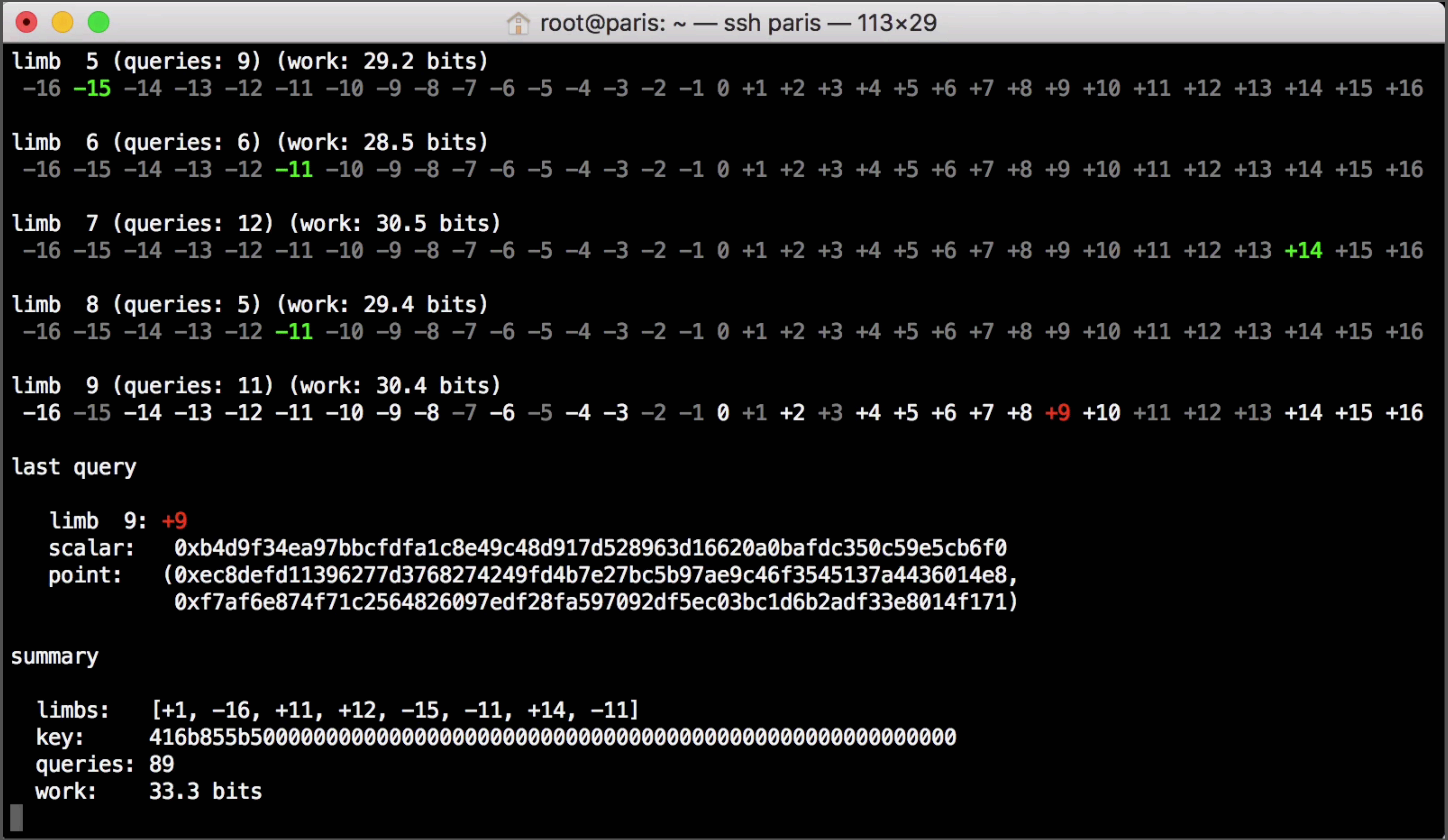
Spot instance infrastructure



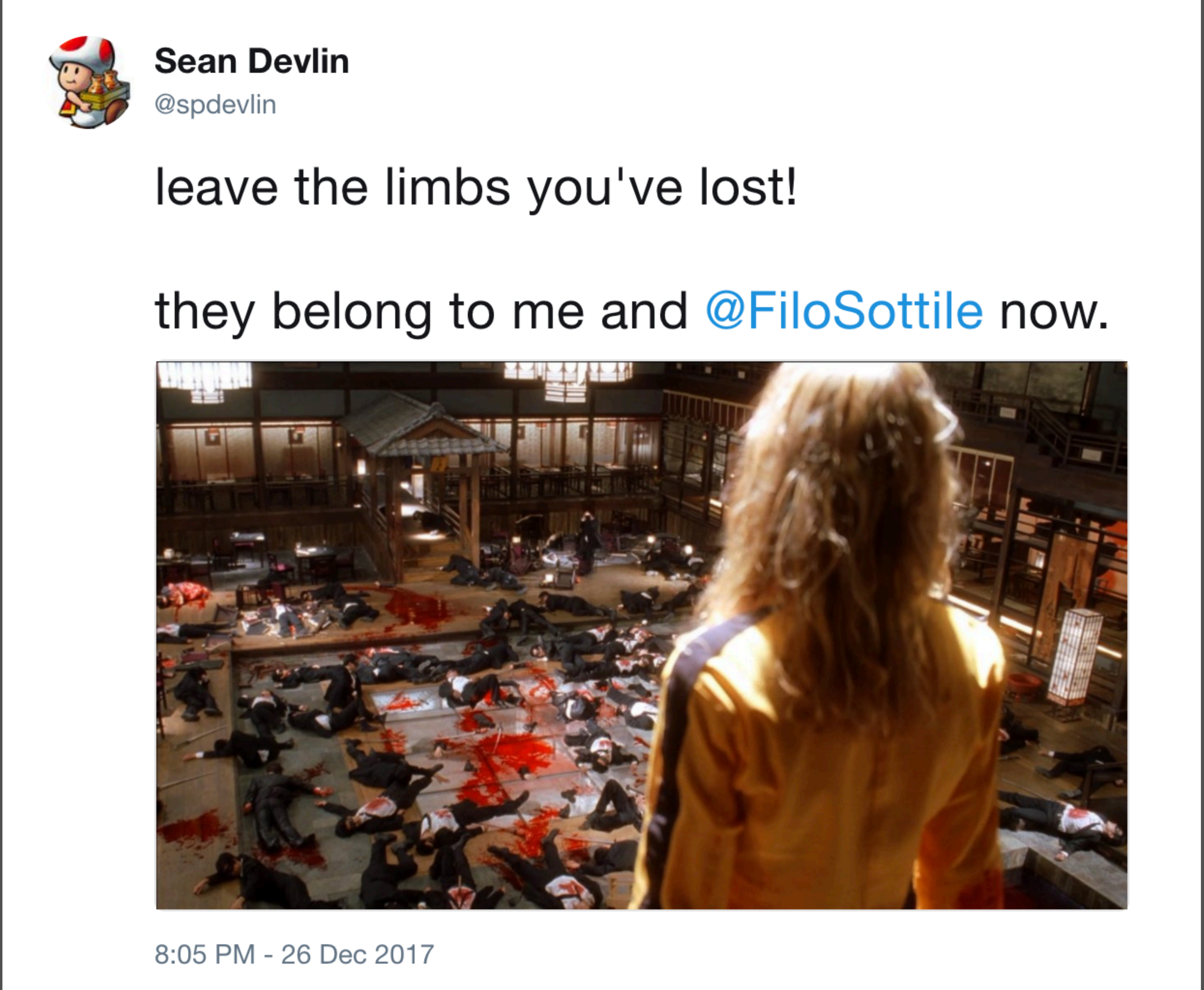
Figures!

- Each key: ~52 limbs, modulo the kangaroo
- Each limb: ~16 points on average
- Each point: ~2^{26} candidate points
- (2^{26} * 16) candidate points: ~85 CPU hours
- 85 CPU hours: $1.26 EC2 spot instances
- Total: 4,400 CPU hours / $65 on EC2

Demo


root@paris: ~ — ssh paris — 113×29

```
limb  5 (queries: 9) (work: 29.2 bits)
 -16 -15 -14 -13 -12 -11 -10 -9 -8 -7 -6 -5 -4 -3 -2 -1 0 +1 +2 +3 +4 +5 +6 +7 +8 +9 +10 +11 +12 +13 +14 +15 +16

limb  6 (queries: 6) (work: 28.5 bits)
 -16 -15 -14 -13 -12 -11 -10 -9 -8 -7 -6 -5 -4 -3 -2 -1 0 +1 +2 +3 +4 +5 +6 +7 +8 +9 +10 +11 +12 +13 +14 +15 +16

limb  7 (queries: 12) (work: 30.5 bits)
 -16 -15 -14 -13 -12 -11 -10 -9 -8 -7 -6 -5 -4 -3 -2 -1 0 +1 +2 +3 +4 +5 +6 +7 +8 +9 +10 +11 +12 +13 +14 +15 +16

limb  8 (queries: 5) (work: 29.4 bits)
 -16 -15 -14 -13 -12 -11 -10 -9 -8 -7 -6 -5 -4 -3 -2 -1 0 +1 +2 +3 +4 +5 +6 +7 +8 +9 +10 +11 +12 +13 +14 +15 +16

limb  9 (queries: 11) (work: 30.4 bits)
 -16 -15 -14 -13 -12 -11 -10 -9 -8 -7 -6 -5 -4 -3 -2 -1 0 +1 +2 +3 +4 +5 +6 +7 +8 +9 +10 +11 +12 +13 +14 +15 +16

last query

   limb  9: +9
   scalar:   0xb4d9f34ea97bbcfdfa1c8e49c48d917d528963d16620a0bafdc350c59e5cb6f0
   point:   (0xec8defd11396277d3768274249fd4b7e27bc5b97ae9c46f3545137a4436014e8,
             0xf7af6e874f71c2564826097edf28fa597092df5ec03bc1d6b2adf33e8014f171)

summary

  limbs:   [+1, -16, +11, +12, -15, -11, +14, -11]
  key:     416b855b50000000000000000000000000000000000000000000000000000000
  queries: 89
  work:    33.3 bits
```



Sean Devlin
@spdevlin

leave the limbs you've lost!

they belong to me and @FiloSottile now.

8:05 PM - 26 Dec 2017

Thank you!

No bug is small enough.

Sean Devlin

@spdevlin

Filippo Valsorda

@FiloSottile