Introducing the Mystika Project

Erik Wallace
University of Connecticut

Some History

The 20th of May 2013

Edward Snowden reveals the existence of PRISM, Bullrun and other aspects of the NSA’s surveillance program.

The 5th of September 2013

A “backdoor” is discovered in the Dual Elliptic Curve Deterministic Random Number Algorithm.

The 7th of April 2014

The Heartbleed Bug

The Heartbleed Bug is a serious vulnerability in the popular OpenSSL cryptographic software library. This weakness allows stealing the information protected, under normal conditions, by the SSL/TLS encryption used to secure the Internet. SSL/TLS provides communication security and privacy over the Internet for applications such as web, email, instant messaging (IM) and some virtual private networks (VPNs).

The Heartbleed bug allows anyone on the Internet to read the memory of the systems protected by the vulnerable versions of the OpenSSL software. This compromises the secret keys used to identify the service providers and to encrypt the traffic, the names and passwords of the users and the actual content. This allows attackers to eavesdrop on communications, steal data directly from the services and users and to impersonate services and users.

The 1st of July 2014

Mystika is born!

29th of September 2014

I leave Indiana for Jerusalem.

The Mystika project stalls!

16th of September 2016

About a dozen of my students at UConn express interest in Mystika.

The project resumes!

Overview of Mystika

Goals

- A drop-in replacement to Open SSL
- A full fledged big number library for APL
- Potential off-shoots?

Why APL?

- To test Aaron’s compiler

  ...no seriously:
- Memory leaks like Heartbleed are not likely.
- Concise code: easier to audit*

  *For someone who knows APL

Design Principles

- All algorithms critical to Cryptography must be “uniform time.”
- All algorithms must be parallel to the greatest extent possible.
- Only cutting edge crypto algorithms are used.
- Only the fastest algorithms are used.
- Each Dyalog APL primitive should have a big number analog.

Uniformity in time (example 1)

Binary exponentiation

Calculate 5^{33281}

$$33281 = 32768 + 512 + 1 = 2^{15} + 2^{9} + 2^{0}$$

i.e. the binary expansion of the exponent is

$$1\,0\,0\,0\,0\,0\,1\,0\,0\,0\,0\,0\,0\,0\,0\,1$$

By repeated squaring we can raise 5 to any power of 2.
The highest power is 15, so it takes 15 steps to compute

5^{33821} and 5^{512}

Then it takes two additional steps of multiplication (for a total of 17 steps)

$$5^{33281} = 5^{32768} \times 5^{512} \times 5^{1}$$

But for a random 16 bit exponent it would take 22.5 steps.
The maximum number of steps is 30.

Uniformity in time (example 2)

The Euclidean Algorithm

Gcd(112,71) takes 8 steps to compute:

		1	1	1	2	1	2	1	2
0	1	1	2	3	8	11	30	41	112
1	0	1	1	2	5	7	19	27	71

Gcd(119,80) takes only 4 steps to compute

		1	2	19	2
0	1	1	3	58	119
1	0	1	2	39	80

The Anatomy of a bignum

An integer: `16 0 0,(8⍴16)⊤3483374771`

```
16 0 0 12 15 10 0 1 4 11 3
```

A negative integer: `16 0 1,(8⍴16)⊤¯3483374771`

```
16 0 1 3 0 5 15 14 11 4 13
```

A decimal: `10 3 0,(10⍴10)⊤3483374771`

```
10 3 0 3 4 8 3 3 7 4 7 7 1
```

A complex number: `0J16 0 0,(8⍴0J16)⊤34833J74771`

```
0J16 0 0 0 0 0 0J1 ¯2J8 ¯8J¯5 ¯15J¯1 1J3
```

A vector of bignums

```
16⍪1 1 3 1⍪1 0 1 0⍪(8⍴16)⊤¯3706 444 ¯2381 2366

16 16 16 16
 1  1  3  1
 1  0  1  0
15  0 15  0
15  0 15  0
15  0 15  0
15  0 15  0
15  0 15  0
 1  1  6  9
 8 11 11  3
 6 12  3 14
```

Full vs. Partial Carrying

Consider what happens when adding 1 to 9 9 9…9 9 in base 10.

```
(10 0 0,20⍴9) add 10 0 0,¯20↑1
```

```
10 ¯1 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
```

$65546=10+2^{16}$ specifies partial carrying:

```
(65546 0 0,20⍴9) add 65546 0 0,¯20↑1
```

```
65546 0 0 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 10 0 0 0
```

Supported primitive operators

red /	rdf ⌿	scn \	scf ⍀
dot .	out .∘	pop ⍣	rop ⍥

Supported primitive functions

ima	11○	rea	9○	cnj/add	+	sub	-
mul	×	cat	⍪	rav	,	trn	⍉
Rot	⌽	rof	⊖	pic	⊃	sqd	⌷
eql	=	neq	≠	leq	≤	geq	≥
gth	>	lth	<	flo/min	⌊	cel/max	⌈
abs/mod	|	rho	⍴	eps	∊	ind	⍳
rol	?	tke/mix	↑	drp/spl	↓	div	÷

Exceptions to the dictionary

- Axis specification must be shifted by 1 unit:

  e.g. `[¯.5]` becomes `[.5]`

- Small nums must be turned into bignums

  e.g. 2∘add   does not work for   `2∘+`

Other functions/operators

sha (dyadic):	Sha-2
mex (operator):	Modular Exponentiation
bch:	Base Change

Limitations

- Axis specification is not supported
- The radix has a range of -2^{32} to 2^{32}
- FFT multiplication maxes out at a number of places dependent on the base
- Partial carrying for complex numbers needs fine tuning

Our License

AGPL v3

Student Contributors

John Bochicchio

Andre Cai

Thomas Crosby

Mason Dicicco

Kurtis Duggan

Emily Maciejewski

Victoria Reichelderfer

Anthony Shaw

Thank you!