Constant Time Big Numbers (For Go)

Lúcás C. Meier

Supervisor: Prof. Bryan Ford

Overview

- Big Numbers?
- Timing Attacks?
- Go?
- Safenum (Our Work)
- Further Work

Big Numbers

Useful in Cryptography

- $\mathbb{N}$ (Natural Numbers)
- $\mathbb{Z}/N\mathbb{Z}$ (Modular Arithmetic)
- $\mathbb{F}_p$ (Prime Fields)

RSA

Public key (e, N), encrypt m with:

$$m^e \mod N$$

$$N \approx 2048 \text{ bits}$$

Too Big!

Elliptic Curve Cryptography

Prime Fields!

$$\mathbb{Z}/p\mathbb{Z}$$

for example:

$$p = 2^{255} - 19$$

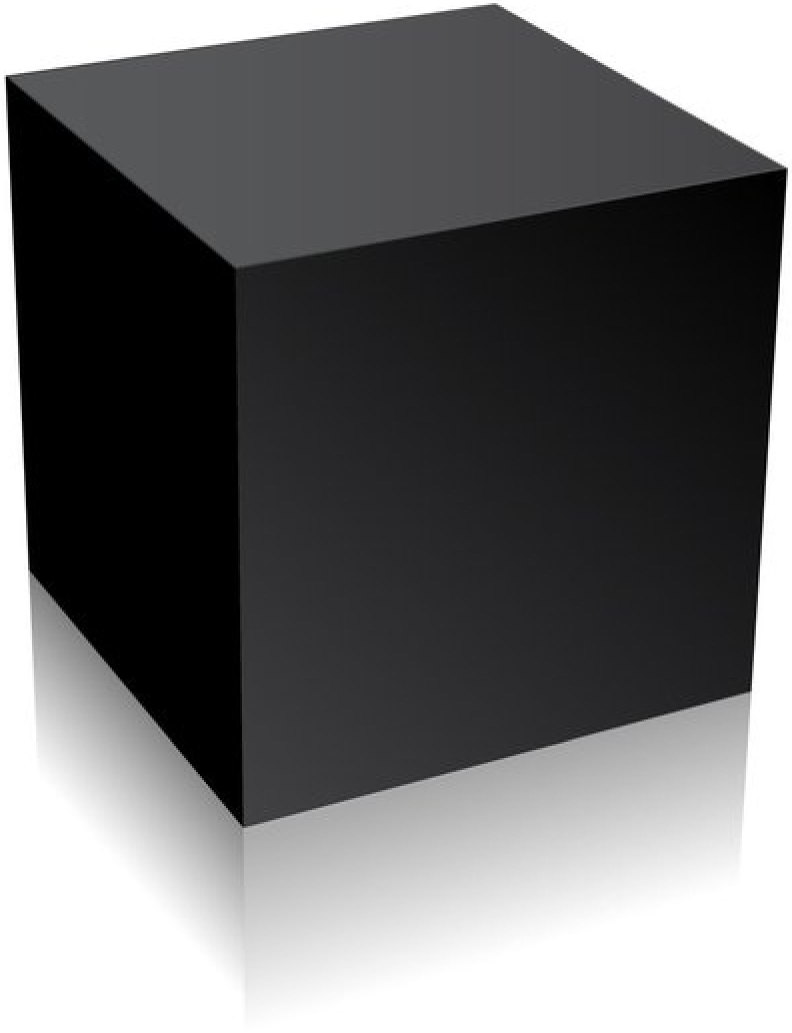
Somewhat big

Implementation Strategies

- Hand-written implementation
- Generated (e.g. FiatCrypto)
- Dynamic (`big.Int`, our library)

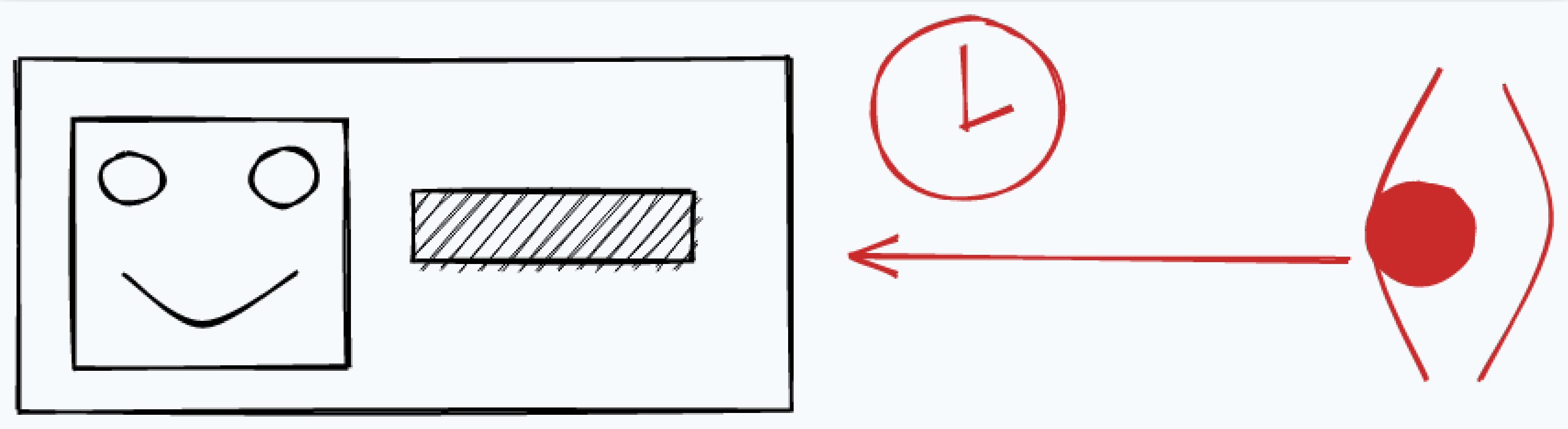

Timing Side Channels
HD
HISTORY.COM

Implementations in Theory

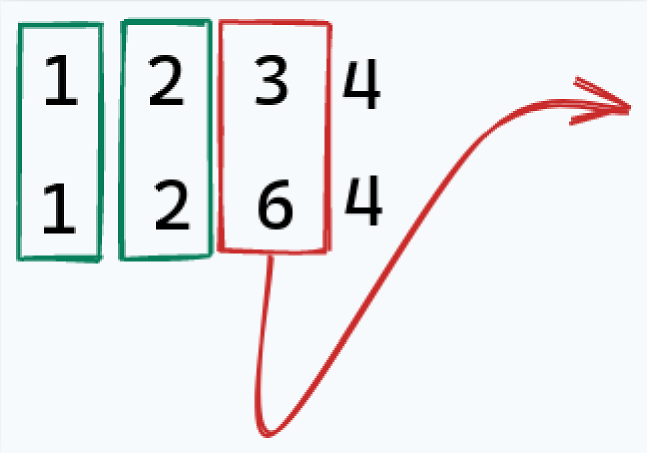


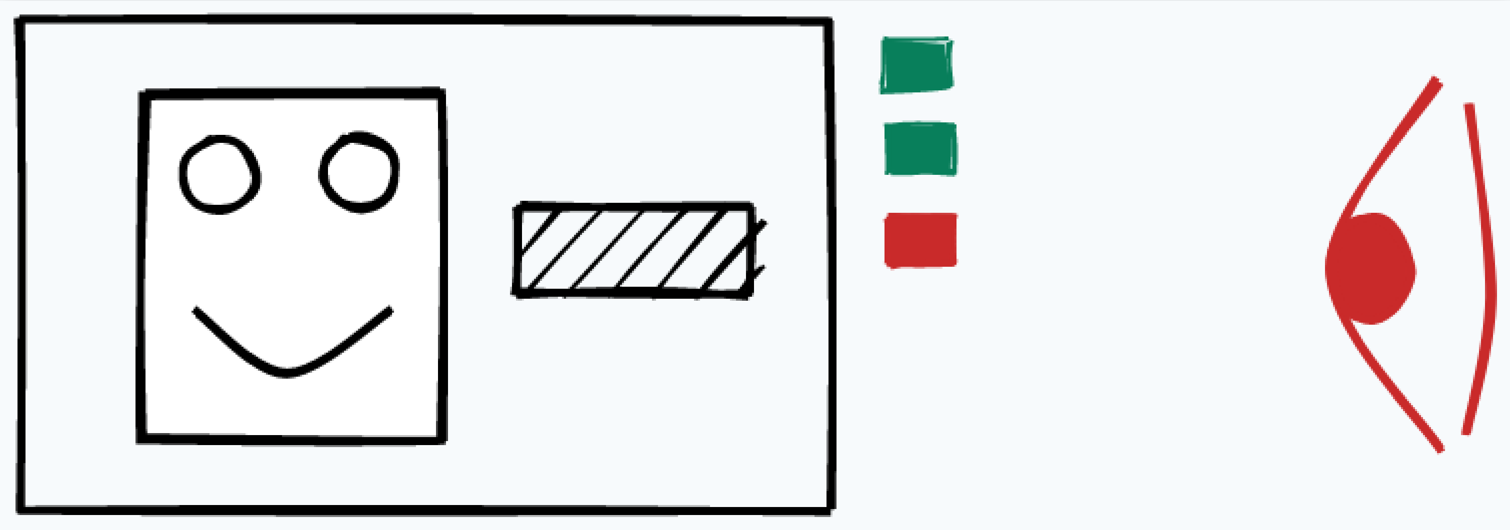
Implementations in Practice

Timing

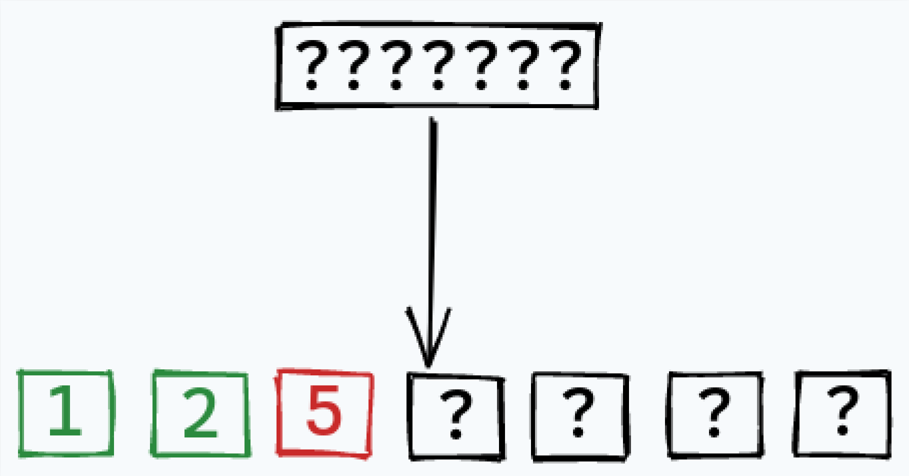


Guessing Passwords





4.6


???????
1
2
5
?
?
?
?

Side-Channel Overview

Subtle Behavior:

- Caches
- Branch Prediction
- Microcode Pipelines

Further Information

A Survey of Microarchitectural Timing Attacks and Countermeasures on Contemporary Hardware

Qian Ge[1], Yuval Yarom[2], David Cock[1,3], and Gernot Heiser[1]

[1]Data61, CSIRO and UNSW, Australia, qian.ge, gernot@data61.csiro.au
[2]Data61, CSIRO and the University of Adelaide, Australia, yval@cs.adelaide.edu.au
[3]Present address: Systems Group, Department of Computer Science, ETH Zürich, david.cock@inf.ethz.ch

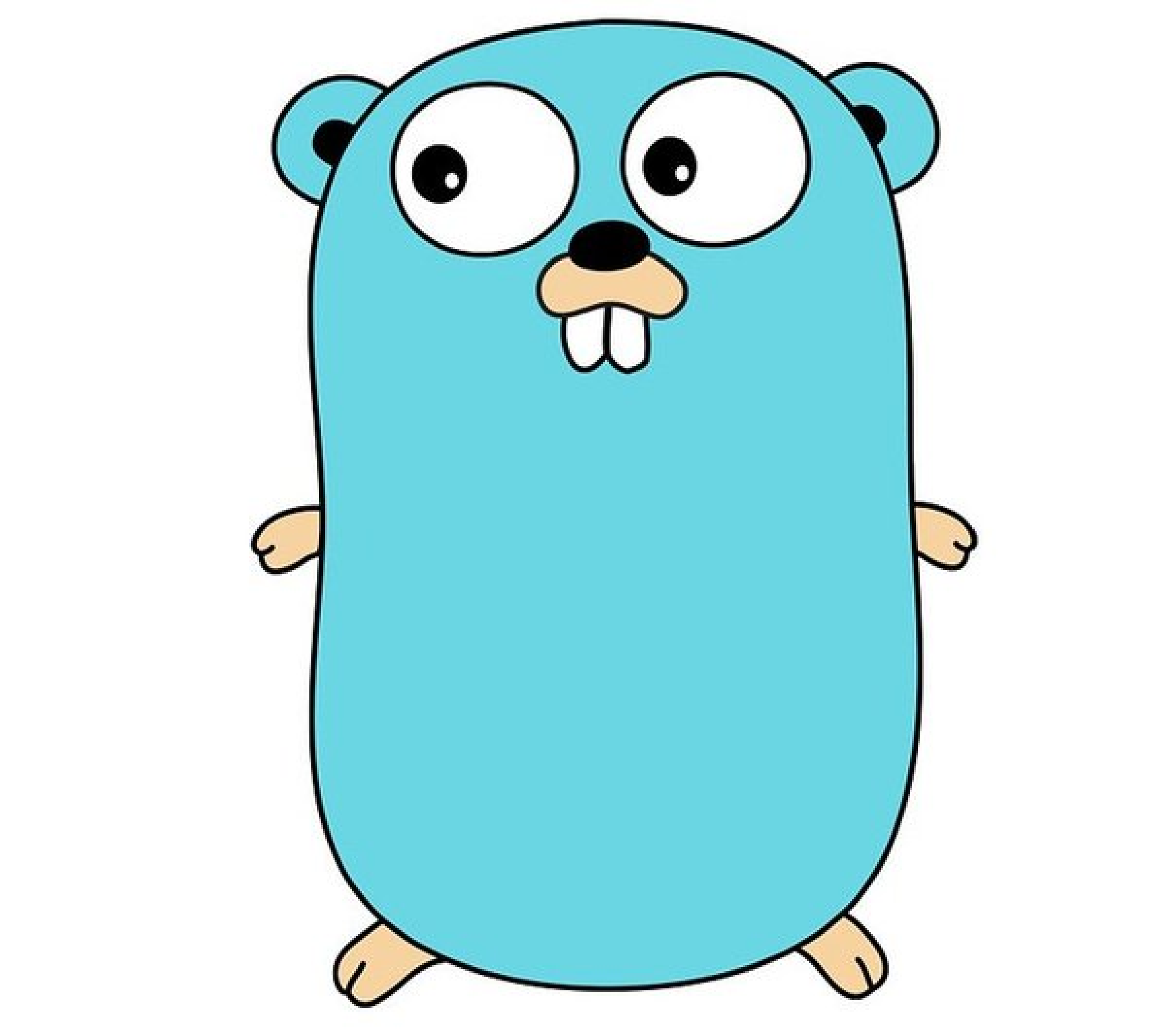
Threat Model

- Loops leak the number of iterations
- Memory accesses leak addresses
- Branching leaks condition

Constant-Time Computing Base

- Addition `+`
- Multiplication `*`
- Logical Operations `|`, `&`, `^`
- Shifts `<<`, `>>`

Go



big.Int

Package big

```
import "math/big"
```

Overview
Index
Examples

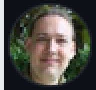
Overview ▾

Package big implements arbitrary-precision arithmetic (big numbers). The following numeric types are supported:

```
Int    signed integers
Rat    rational numbers
Float  floating-point numbers
```


go/crypto
We use big.Int
for Cryptography
It's constant-time right?
It's constant-time right?

Not Constant-Time



bford commented on Jun 13, 2017 Contributor

Problem: Constant-Time Arithmetic for Cryptographic Uses

The math/big package naturally and inevitably gets used for cryptographic purposes, including in the standard Go crypto libraries. However, this usage is currently unsafe because math/big does not support constant-time operation and thus may well be leaking secret keys and other sensitive information via timing channels. This is a well-known problem already documented in math/big's godoc documentation.

A much more specific issue related to this was raised in 2011 (#2445) but eventually closed for lack of attention for too long.

See the preliminary companion patch 45490 presenting a first-cut at an implementation of this proposal: https://go-review.googlesource.com/c/45490/ But the most important details and considerations are discussed here.

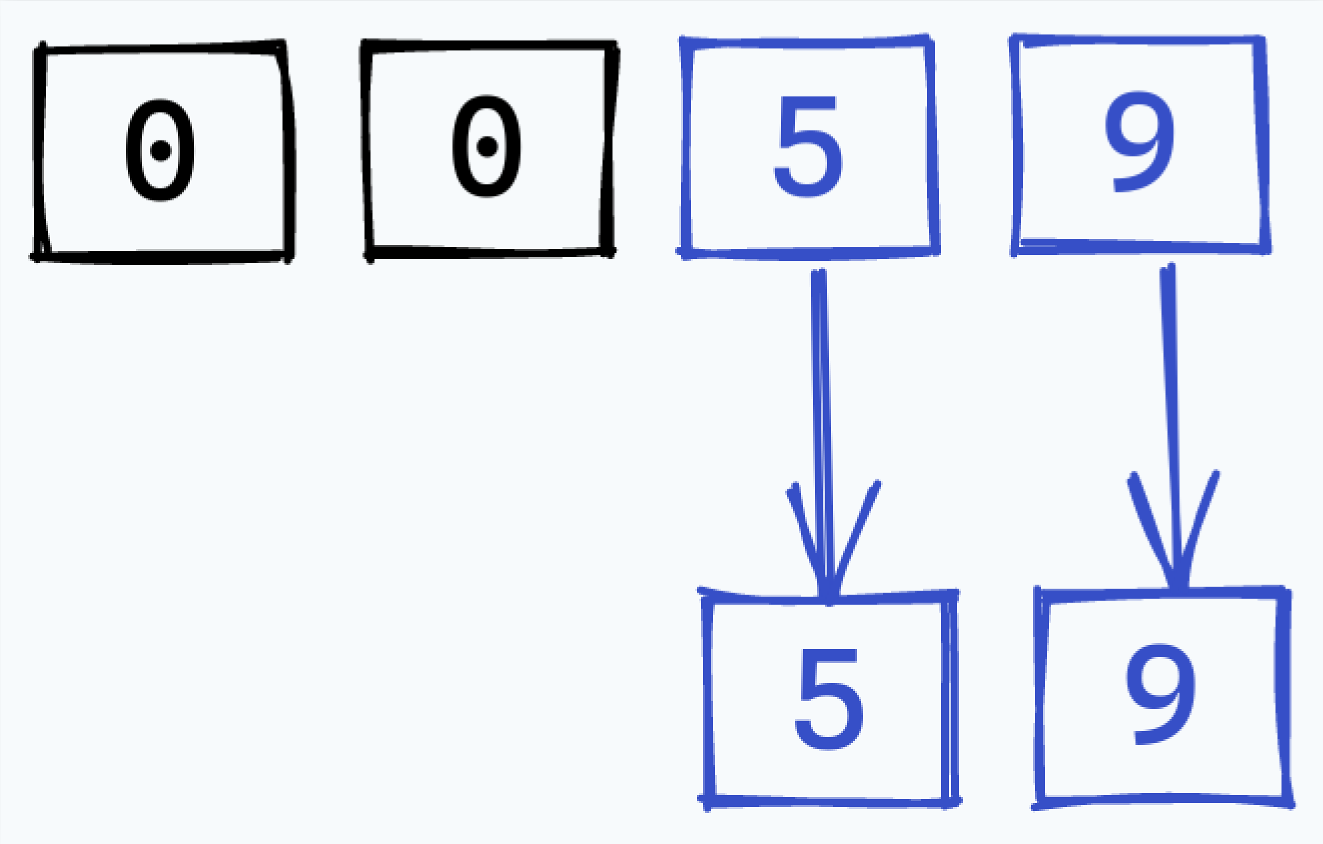
Why? Bad Algorithms

nat.go

math > big > nat.go > ...

```go
	}
	if c != 0 {
		subVV(z[:n], z[n:], m)
	} else {
		copy(z[:n], z[n:])
	}
	return z[:n]
}
```

Why? Padding



in `go/crypto`

- Extensively in **RSA**, and **DSA**
- **ECC**: Elliptic Curve interface uses `big.Int`
- Only **P384** uses `big.Int` for field arithmetic

Mitigations

In RSA: *blinding*:

Instead of:

$$c^d \mod N$$

Calculate:

$$\frac{1}{r}(c \cdot r^e)^d \mod N$$

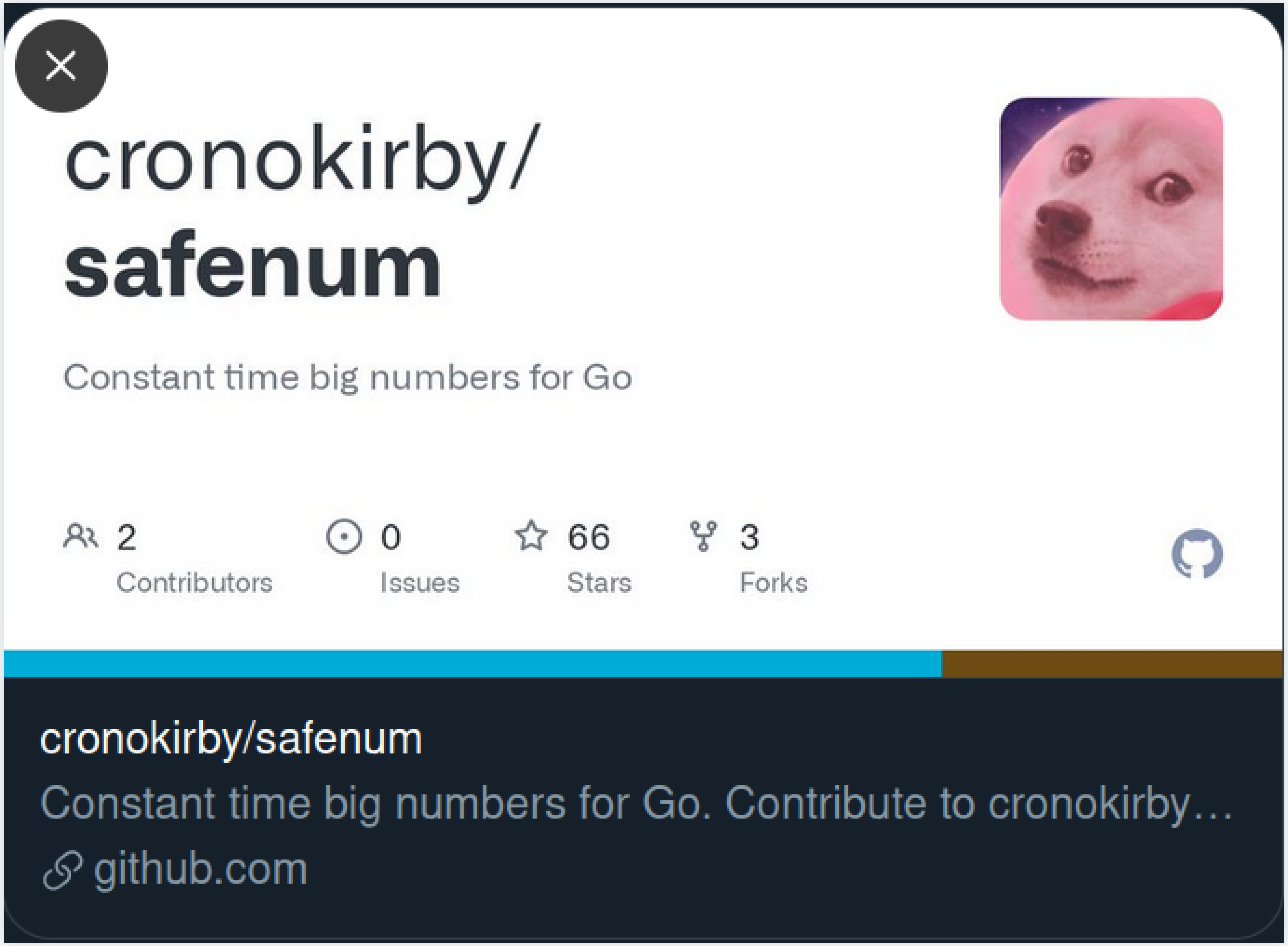
There be Dragons?

rsa.go

crypto > rsa > rsa.go > DecryptOAEP

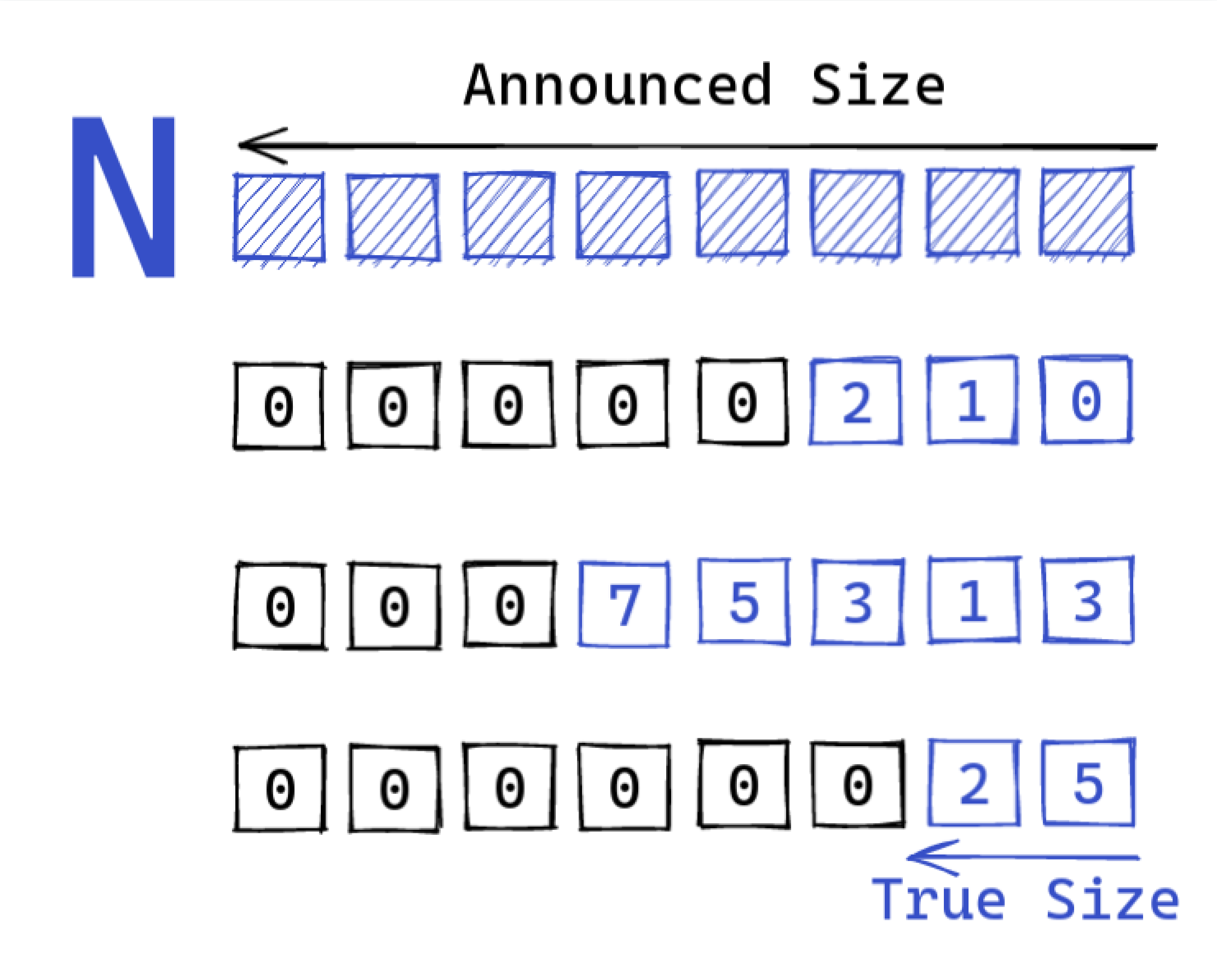
```go

// We probably leak the number of leading zeros.
// It's not clear that we can do anything about this.
em := m.FillBytes(make([]byte, k))
```

Our Library


cronokirby/
safenum
Constant time big numbers for Go
2
Contributors
0
Issues
66
Stars
3
Forks
cronokirby/safenum
Constant time big numbers for Go. Contribute to cronokirby…
github.com

Operations

- Modular addition, subtraction, exponentation, etc.
- Modular square roots
- “Raw” addition and multiplication



Announced Size
N
0 0 0 0 0 2 1 0
0 0 0 7 5 3 1 3
0 0 0 0 0 0 2 5
True Size

Constant-Time Choice

Performance: Operations

Operation	op / s (`big.Int`)	op / s (`Nat`)	ratio
Addition	10,980,842	12,164,599	0.90
Modular Addition	6,986,739	3,075,188	2.27
Multiplication	1,316,322	542,385	2.43
Modular Reduction	454,917	63,253	7.19
Modular Multiplication	1,000,000	44,596	22.42
Modular Inversion	1,000,000	621	1610
Modular Exponentiation	223	86	2.59

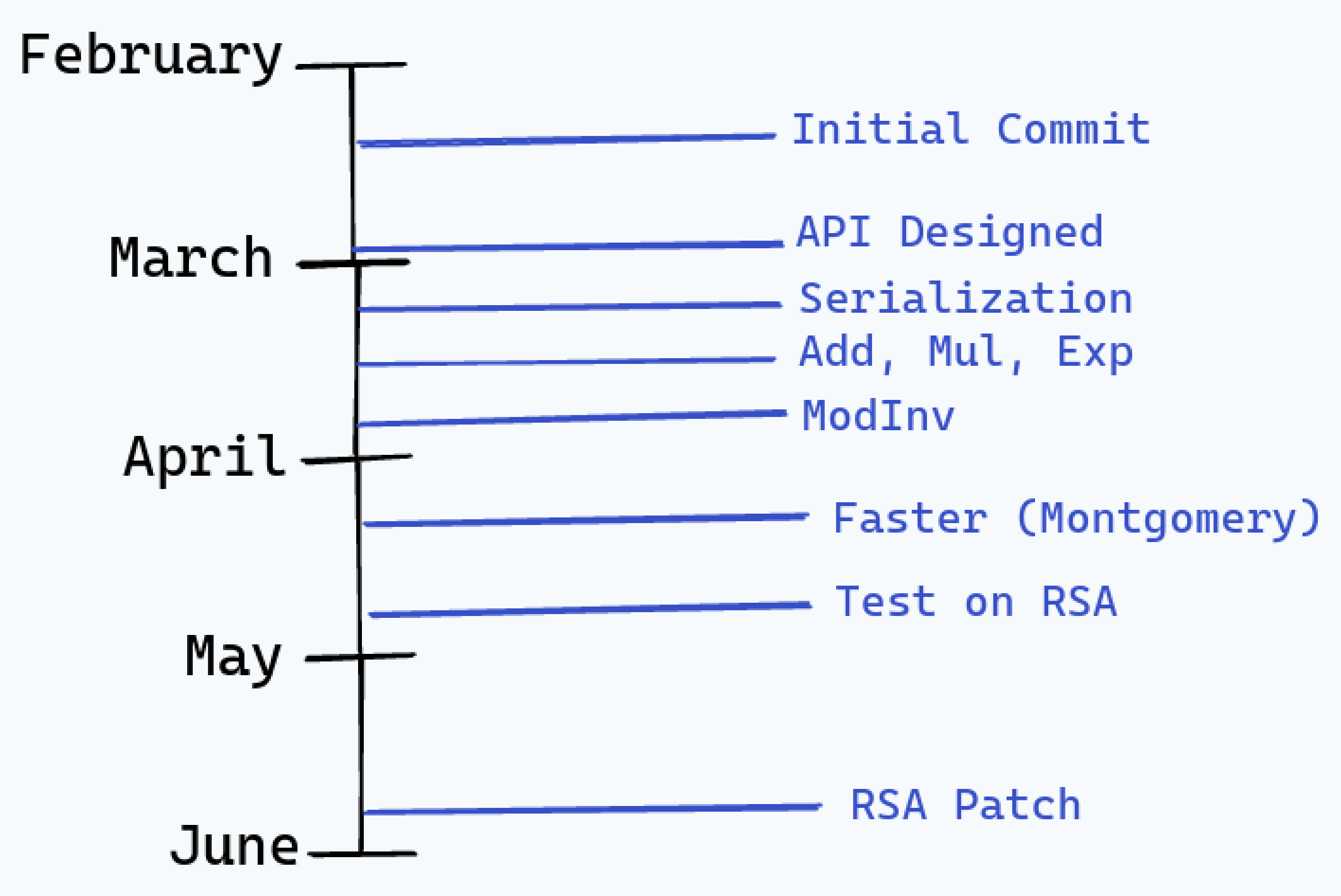
Operation	op / s (`big.Int`)	op / s (`Nat`)	ratio
$\sqrt{z} \mod p_3$	40,464	26,886	1.50
$\sqrt{z} \mod p_1$	-	7,867	-

Performance: Cryptography

Operation	op / s (`big.Int`)	op / s (`Nat`)	ratio
RSA Decrypt	670	312	2.15
RSA Sign	675	372	1.81
RSA Decrypt (3 Prime)	1173	596	1.97
DSA Sign	6202	2625	2.36
DSA Parameters	0.89	1.64	0.54

Patching RSA

Timeline


February
Initial Commit
API Designed
March
Serialization
Add, Mul, Exp
ModInv
April
Faster (Montgomery)
Test on RSA
May
RSA Patch
June

The most important artifact?

Understanding!

Further Work

- Verifying security properties
- Improving performance: Assembly?
- More scenarios: **ECC**, **PQC**?

In Summary

We made an alternative to `big.Int` for Cryptography. It's only 2x slower.