Constant-Time Big Numbers (for Go)

Lúcás Críostóir Meier

School of Computer and Communication Sciences

Decentralized and Distributed Systems lab (DEDIS)

BSc Semester Project

May 2021

Responsible and Supervisor
Prof. Bryan Ford
EPFL / DEDIS

Contents

1 Introduction

With the Internet cemented as a principal keystone in communication, and ever increasing activity taking place digitially, the importance of secure communication and Cryptography has never been greater. Thankfully, after 50 years of Public Key Cryptography [3], we have good theoretical systems to provide these guarantees.

Most of these systems rely on modular arithmetic with large numbers, such as RSA or Elliptic Curve Cryptography (CITATION). Working with such numbers is not natively supported by hardware, requiring a "Big Number" software library to provide this functionality.

Unfortunately, even though Public Key Cryptosystems have been heavily scrutinized *in theory*, in practice many vulnerabilities arise in software implementations of these systems.

One particularly pernicious class of vulnerability are **timing attacks** [4], where an implementation leaks information about secret values through its execution time or cache usage, among many side-channels.

Libraries for Big Numbers that are not designed with Cryptography in mind are pervasively vulnerable to this class of attack.

In particular, Go [1] provides a general purpose Big Number type, `big.Int`, which suffers from these vulnerabilities, as we detail later in this report. Unfortunately, this library gets used for Cryptography [2], including inside of Go's own standard library, in `go/crypto`.

We've addressed this issue by creating a library [5] designed to work with Big Numbers in the context of Public Key Cryptography. Our library provides the necessary operations for implementing these systems, all while avoiding the leakage of secret information. To demonstrate its utility, we've modified Go's `go/crypto` package, replacing the use of `big.Int` in the DSA and RSA systems.

2 Background

Explain the necessary math and concerns that are relevant for our project.

2.1 Big Numbers in Cryptography

Explain how Big Numbers are used in Public Key Cryptography. Mention DSA, RSA, Elliptic curves.

2.1.1 Big Numbers in `go/crypto`

Explain how things are used in Go's standard library.

2.2 Side-Channels

Explain the concept of side-channels in cryptography, and give some of the fundamental types we're concerned with.

2.2.1 Actual Attacks

Explain how these concerns actually lead to vulnerabilities in systems.

2.2.2 Our Threat-Model

Explain the threat model we have, and what side-channels we aren't concerned about.

2.3 Vulnerabilities in `big.Int`

Explain what vulnerabilities are potentially lurking in bigInt.

2.3.1 Padding and Truncation

Explain how big.Int truncates numbers internally.

Explain potential issues with padding in cryptography.

2.3.2 Leaky Algorithms

Explain how most algorithms are potentially leaky.

2.3.3 Mitigations

Explain what mitigations are deployed in Go.

3 Implementation

Describe at a high level what we've done.

3.1 Strategies for numbers in Cryptography

Describe different strategies in place for providing numbers for Cryptography.

3.2 The `safenum` library

Describe at a high level what the library provides.

3.2.1 Handling Size

Describe how we handle sizing of numbers.

3.3 Some Basic Techniques

Describe some basic techniques for constant-time operation.

3.4 Some Algorithm Choices

Describe the algorithm choices we've used for different things.

4 Results

Describe what results we've managed to perform.

4.1 Comparison with `big.Int`

Describe the final performance results we've managed to achieve.

4.2 Comparison with `go/crypto`

Describe the benchmarks on actual code.

5 Further Work

5.1 Upstreaming to `go/crypto`

Describe our work in providing a patch for RSA, and what results we've managed to achieve.

6 Conclusion

Summarize the things we put in the introduction.

Acknowledgements

References

[1] The Go Authors. The go programming language, 2016. URL: `https://golang.org/ref/spec`.

[2] Bryan Ford. proposal: math/big: support for constant-time arithmetic Issue #20654 golang/go, 2017. URL: `https://github.com/golang/go/issues/20654`.

[3] Martin E Hellman. An Overview of Public Key Cryptography. *IEEE COMMUNICATIONS SOCIETY MAGAZINE*, page 9, 1978.

[4] Paul C Kocher. Cryptanalysis of Di e-Hellman, RSA, DSS, and Other Systems Using Timing Attacks. page 6, 1995.

[5] Lúcás Críostóir Meier. safenum, 2021. URL: `https://github.com/cronokirby/safenum`.