GCD: Garbled, Corrected, Demonstrandum

Fixing and Proving Go's Extended GCD Implementation

Linard Arquint

Department of
Computer Science
School of Computing

Motivation

$$\gcd(17, 42) = 1$$

$$\text{extgcd}(17, 42) = (1, 5)$$

$$17 * 5 \equiv 1 \pmod{42}$$

used to compute the
RSA private key exponent

Motivation

```
extgcd(a, n) {
  u := a; v := n
  while v ≠ 0 {
    if isOdd(u) ∧ isOdd(v) {

      // make u or v even

    }
    if isEven(u) {
      u := u / 2
    } else {
      v := v / 2
    }

  }
  return u, A
}
```

Motivation

```
extgcd(a, n) {
  u := a; v := n
  while v ≠ 0 {
    if isOdd(u) ∧ isOdd(v) {
      if v < u {
        u := u − v
      } else {
        v := v − u
      }

    }
    if isEven(u) {
      u := u / 2
    } else {
      v := v / 2
    }

  }
  return u, A
}
```

Motivation

```
extgcd(a, n) {
  u := a; v := n; // init coefs A, B, C, D
  while v ≠ 0 {
    if isOdd(u) ∧ isOdd(v) {
      if v < u {
        u := u − v
      } else {
        v := v − u
      }
      // update coefs
    }
    if isEven(u) {
      u := u / 2
    } else {
      v := v / 2
    }
    // update coefs
  }
  return u, A
}
```

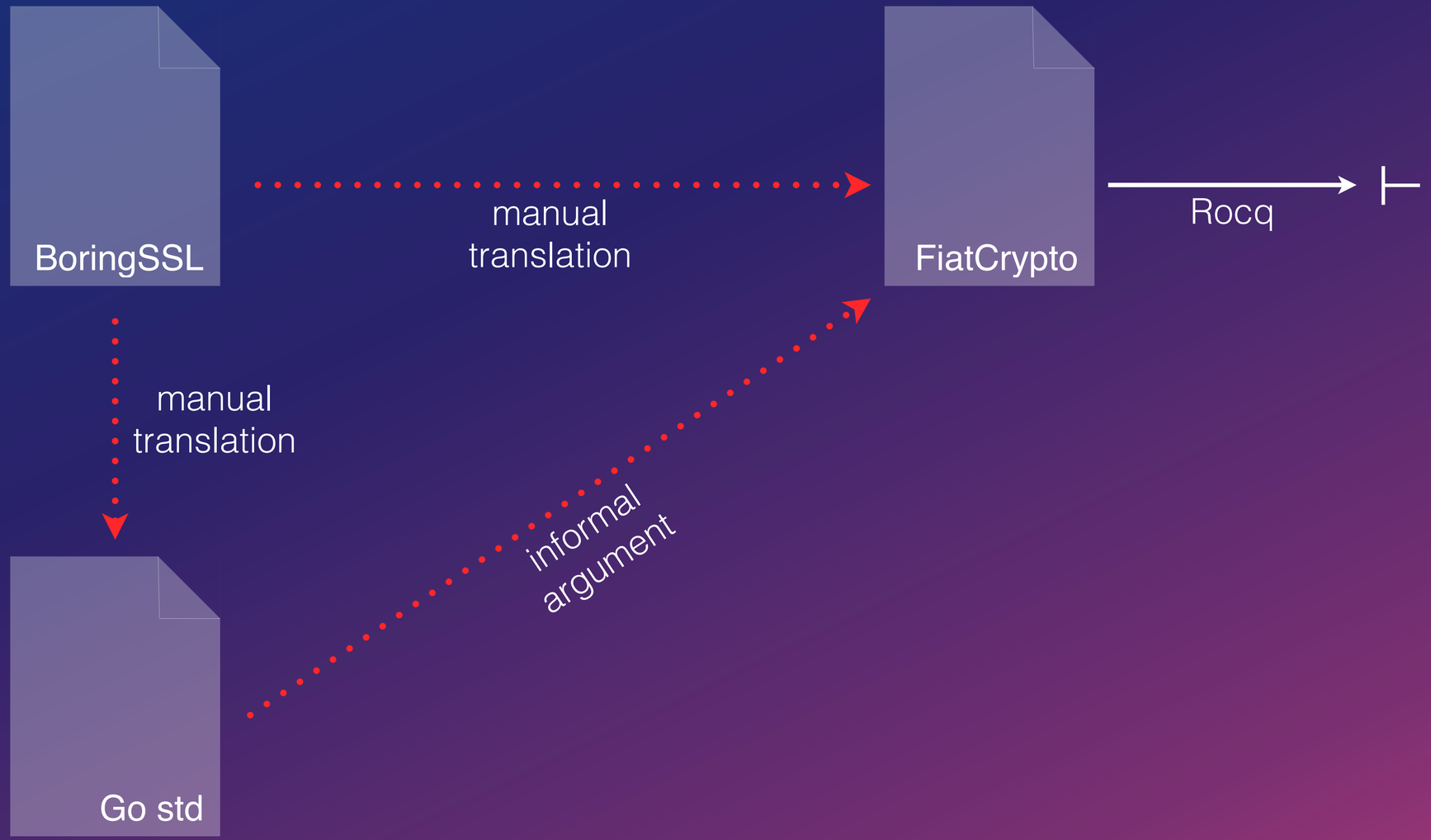
Invariants

gcd(a, n) = gcd(u, v)

u = A * a – B * n

v = D * n – C * a

State of the Art



This Work


BoringSSL
manual
translation
FiatCrypto
Rocq


Go std
Gobra

Contributions

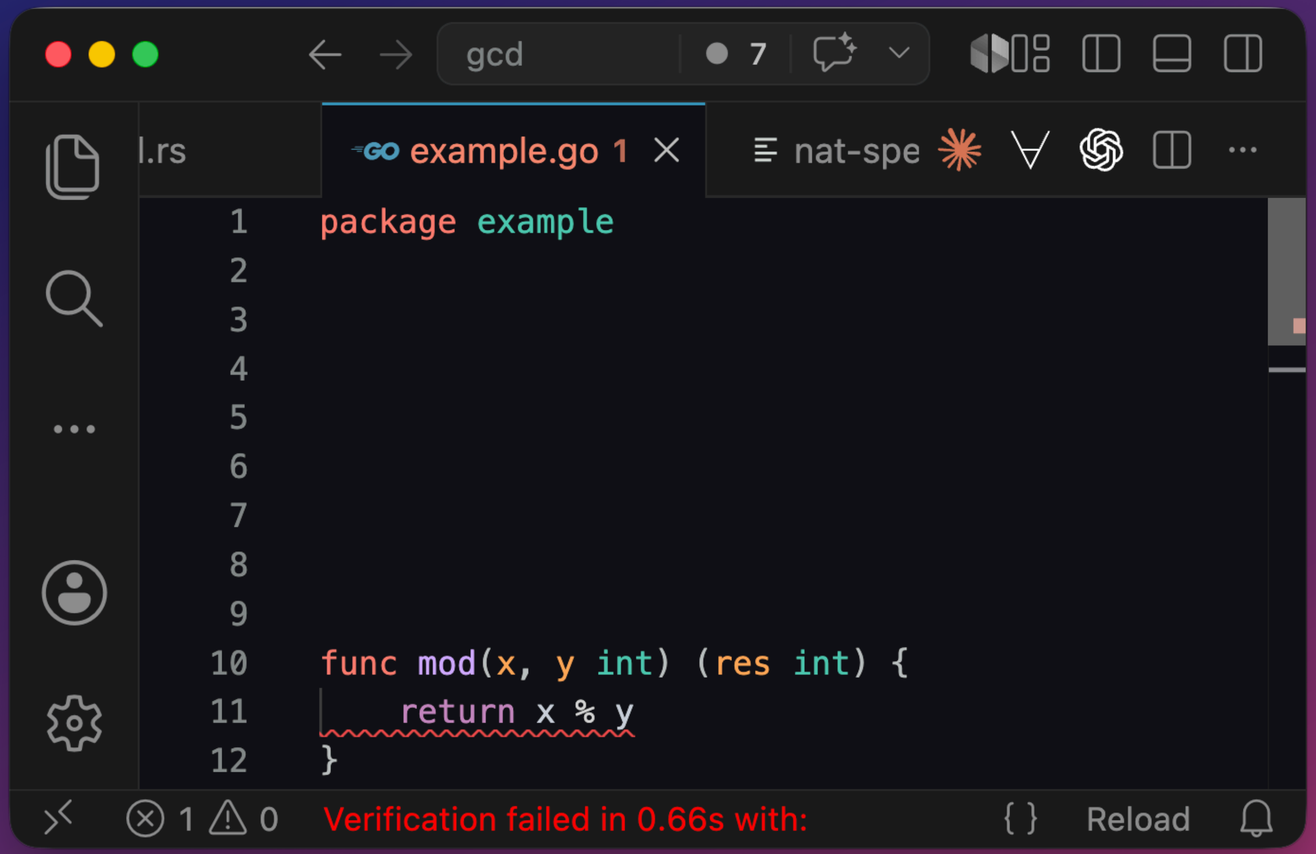
Identify 2 deviations violating the algorithm's invariants

Fix results in a 24% speedup

Fix these deviations

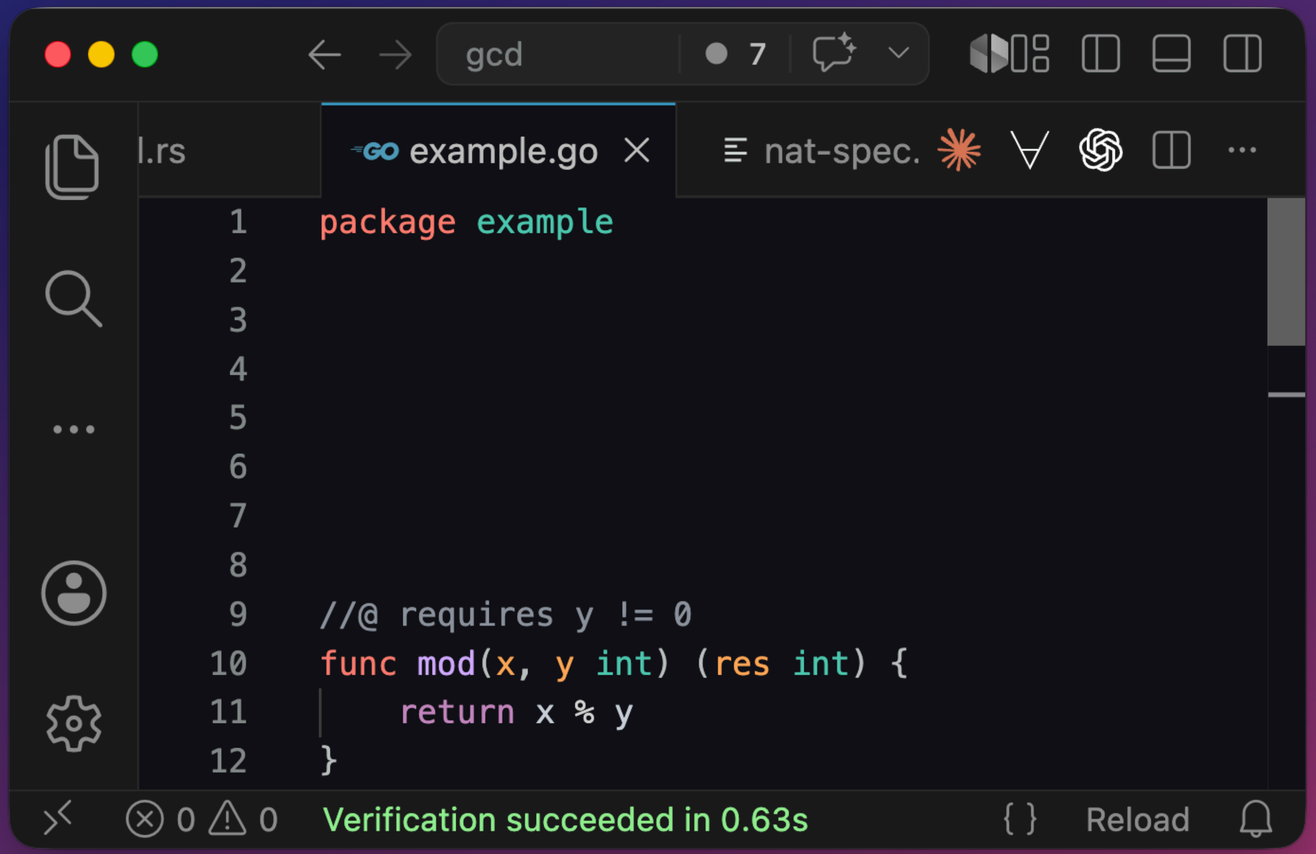
Verify the fixed extended GCD implementation

gobra


gcd
7
l.rs
example.go 1
nat-spe
1 package example
2
3
4
5
6
7
8
9
10 func mod(x, y int) (res int) {
11 return x % y
12 }
1 0
Verification failed in 0.66s with:
{ }
Reload

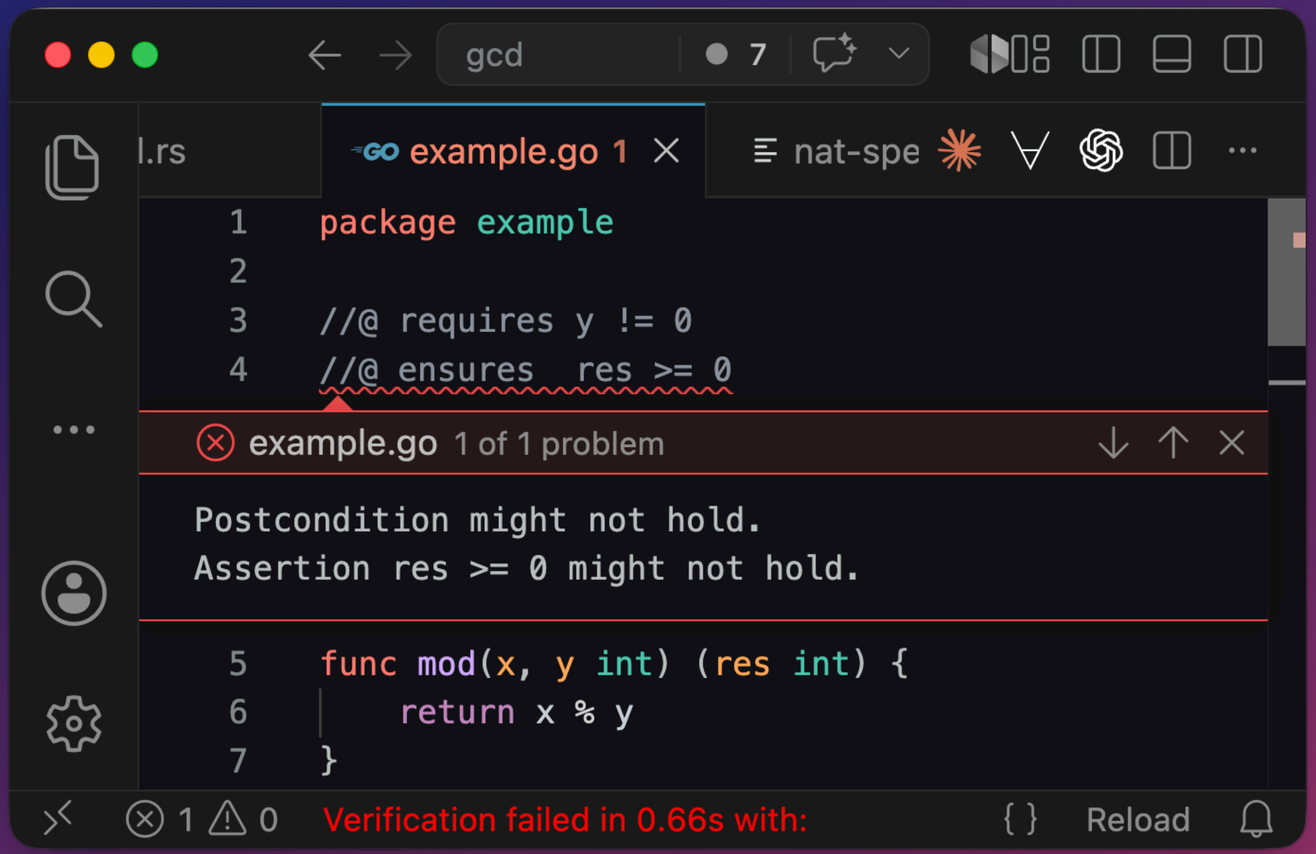
gobra

- Automated program verifier for concurrent Go code
- Modular
- Panic freedom



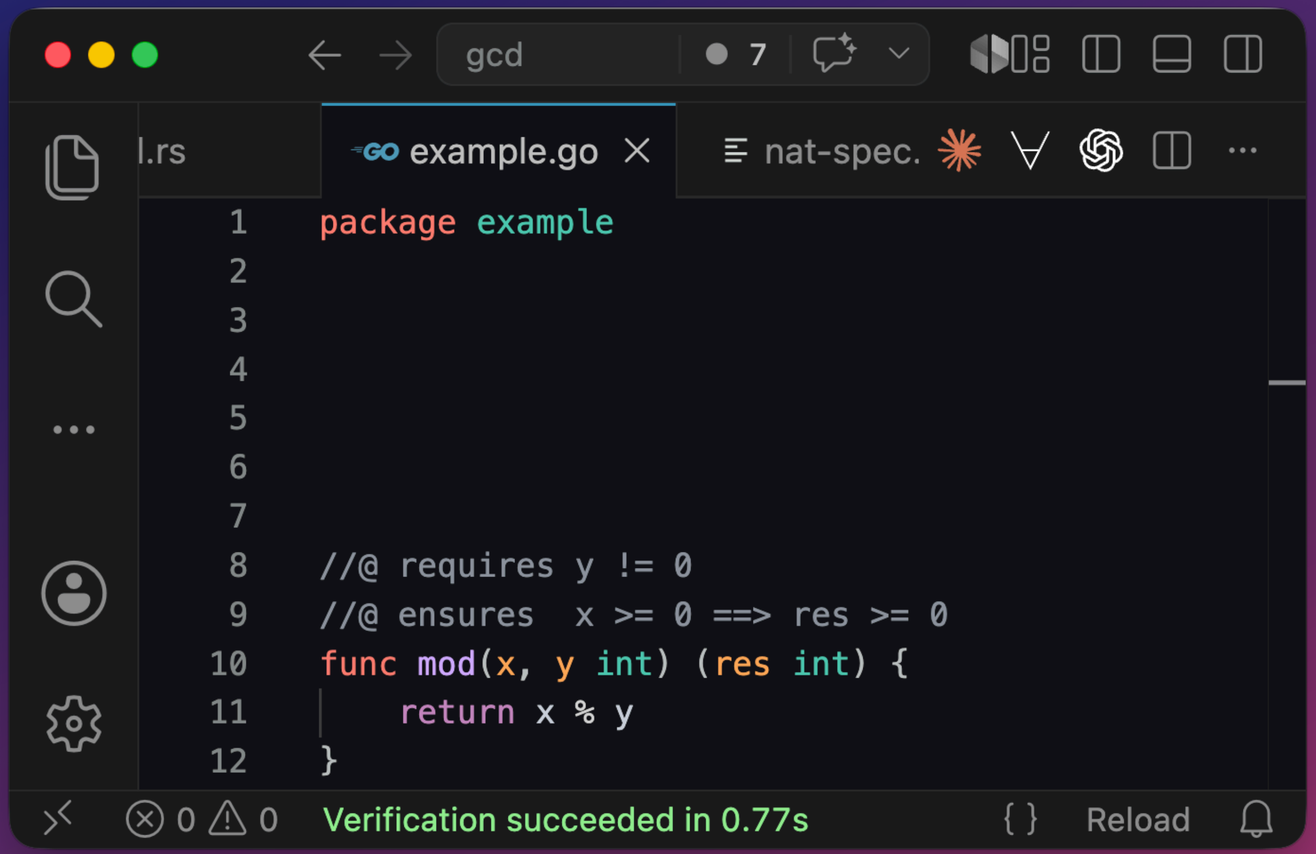
gobra

- Automated program verifier for concurrent Go code
- Modular
- Panic freedom



gobra

- Automated program verifier for concurrent Go code
- Modular
- Panic freedom
- Functional properties
- Memory safety



Verification Process

1. Axiomatize arbitrary-precision natural numbers

```
type Nat struct {
  limbs []uint
}
```

+

Memory footprint

Mathematical abstraction

Verification Process

2 Specify the implementation

Translate the natural language specification

Verification Process

3 Prompt Claude Code

Gobra's readable error messages proved helpful

Failed repeatedly as implementation did not maintain the invariants

Required multiple prompts to convince Claude Code that implementation might be incorrect

Coef Update Deviation

```
if A + C < n {
  A := A + C
  B := B + D
} else {
  A := A + C - n
  B := B + D - a
}
```

BoringSSL

```
if A + C < n {
  A := A + C
} else {
  A := A + C - n
}

if B + D < a {
  B := B + D
} else {
  B := B + D - a
}
```

Go std

Verification Process

4 Complete proof & cleanup

Unify lemmas

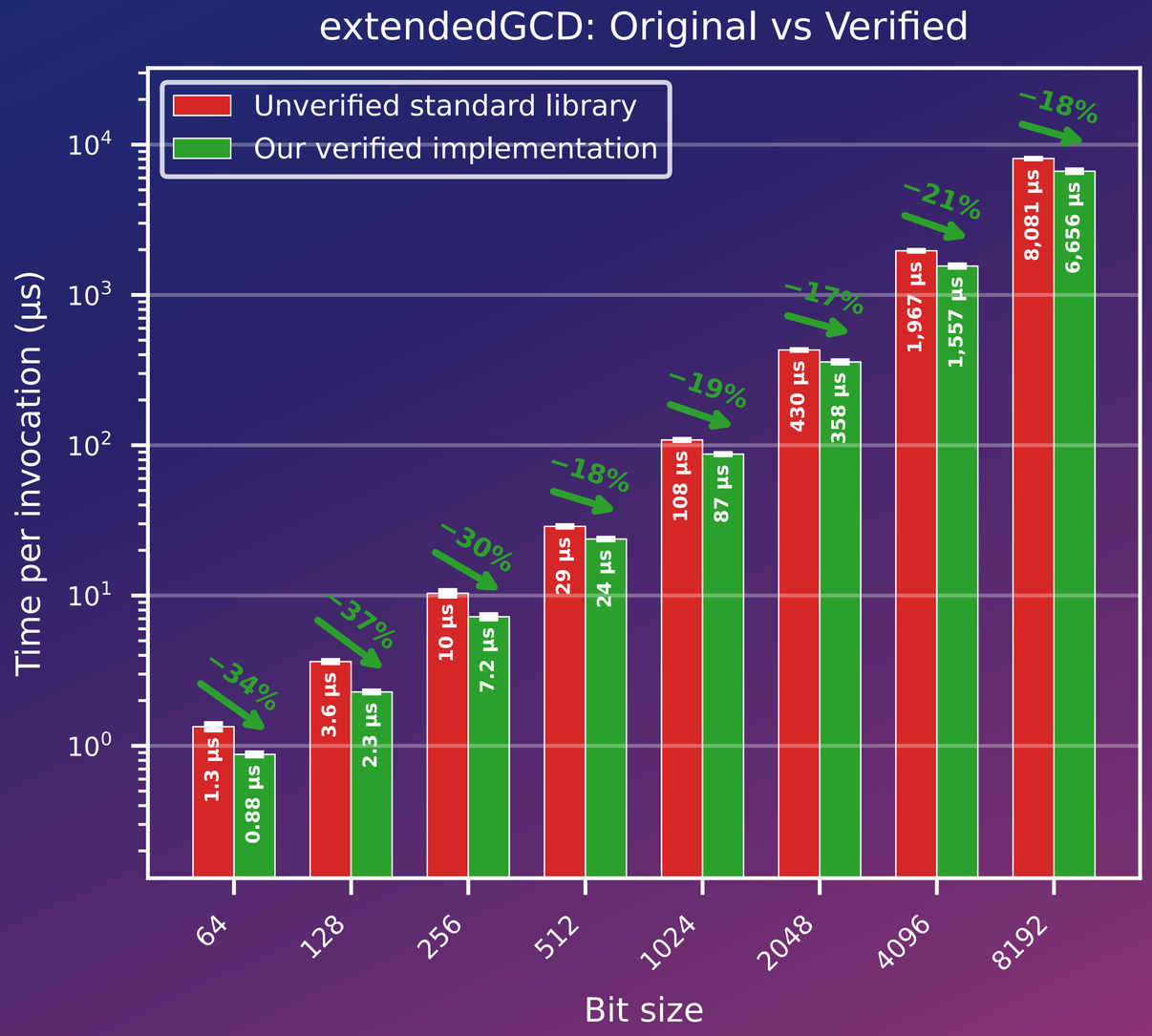
Review axiomatized lemmas & check their Lean proofs

Extend GCD’s domain of inputs to match its prose description

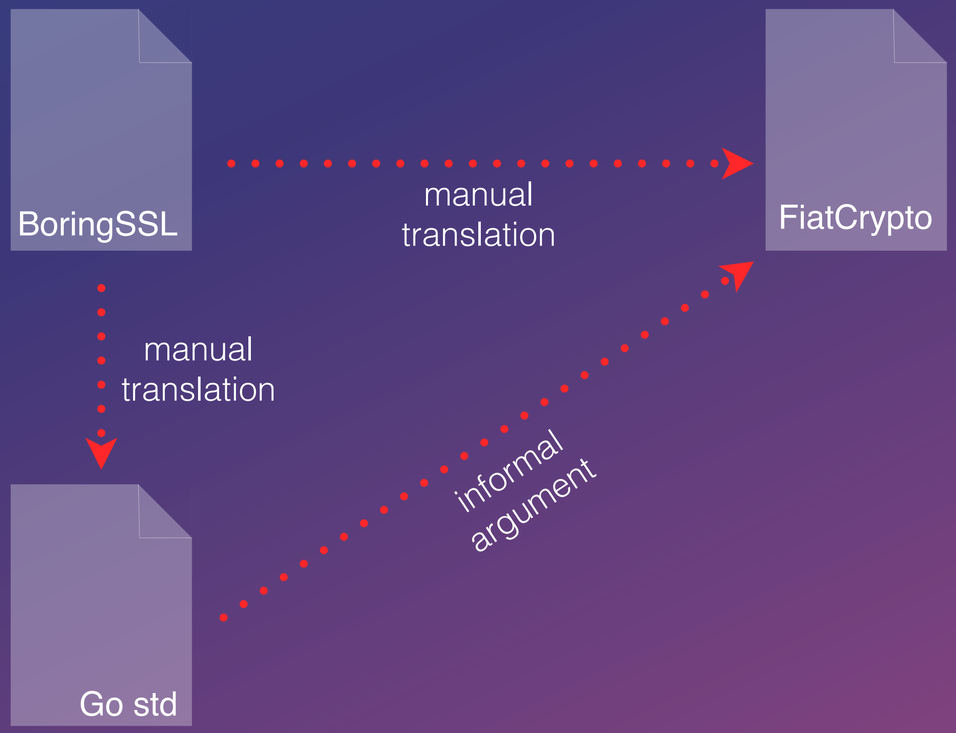
Evaluation

	LOC	LOS
`extendedGCD`	53	99
`InverseVarTime`	10	18
`GCDVarTime`	7	12
`syncAdd`	8	60
Other Go functions	71	314
Auxiliary definitions	—	130
Lemmas	—	349
Total	149	838

Evaluation

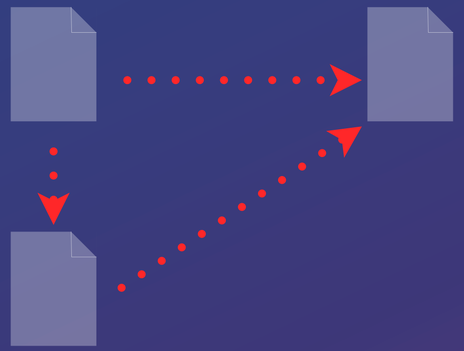

extendedGCD: Original vs Verified
Unverified standard library
Our verified implementation
Time per invocation (µs)
10^0
10^1
10^2
10^3
10^4
1.3 µs
0.88 µs
−34%
3.6 µs
2.3 µs
−37%
10 µs
7.2 µs
−30%
29 µs
24 µs
−18%
108 µs
87 µs
−19%
430 µs
358 µs
−17%
1,967 µs
1,557 µs
−21%
8,081 µs
6,656 µs
−18%
64
128
256
512
1024
2048
4096
8192
Bit size

Conclusions



Translation as a soundness threat

Conclusions



Translation as a soundness threat

Upstreamed patch

Verified GCD implementation

Agentic verification