

GCD: Garbled, Corrected, Demonstrandum

Fixing and Proving Go's Extended GCD Implementation

Linard Arquint



Department of
Computer Science
School of Computing

Motivation

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

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

used to compute the
RSA private key exponent

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

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  
}
```

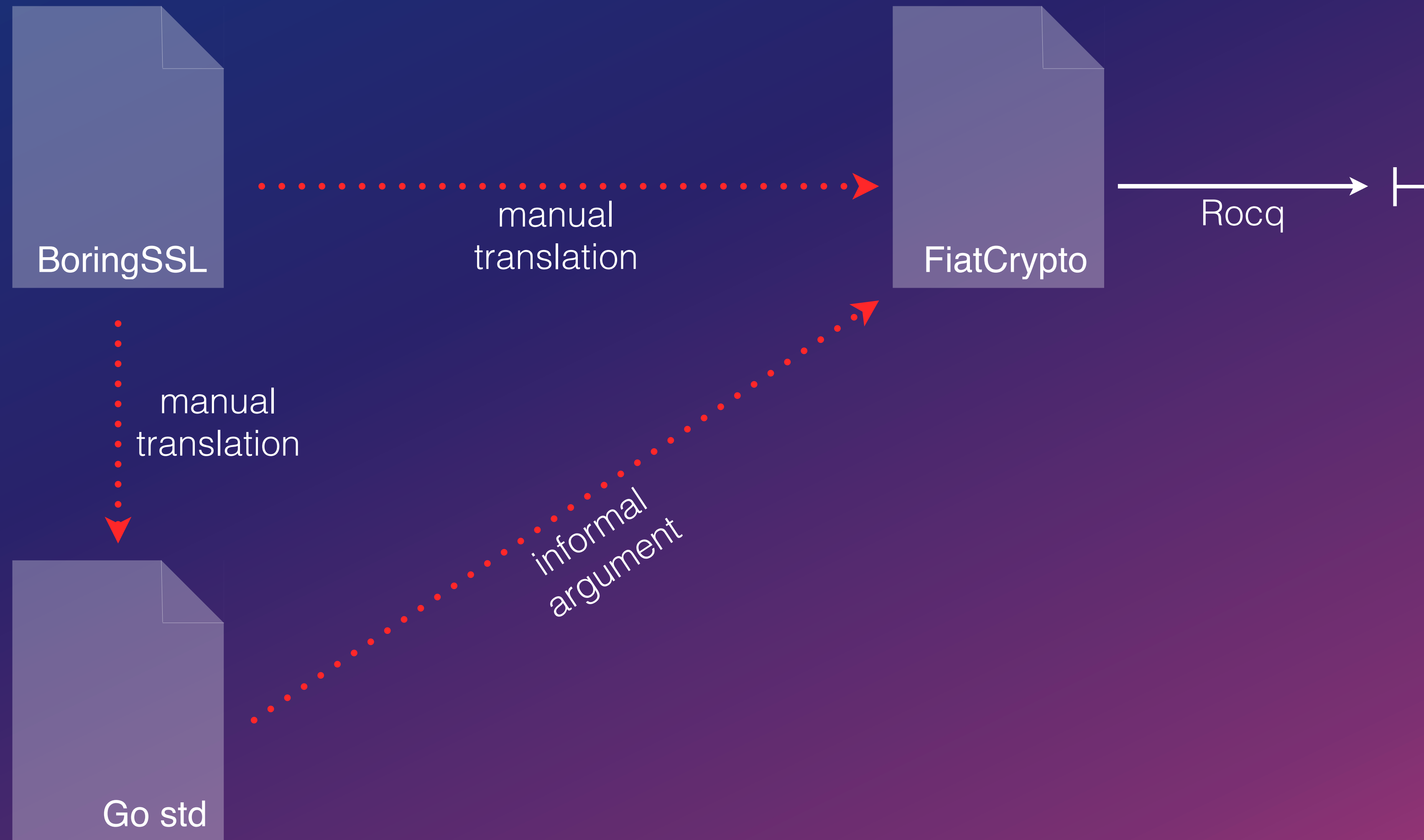
Invariants

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

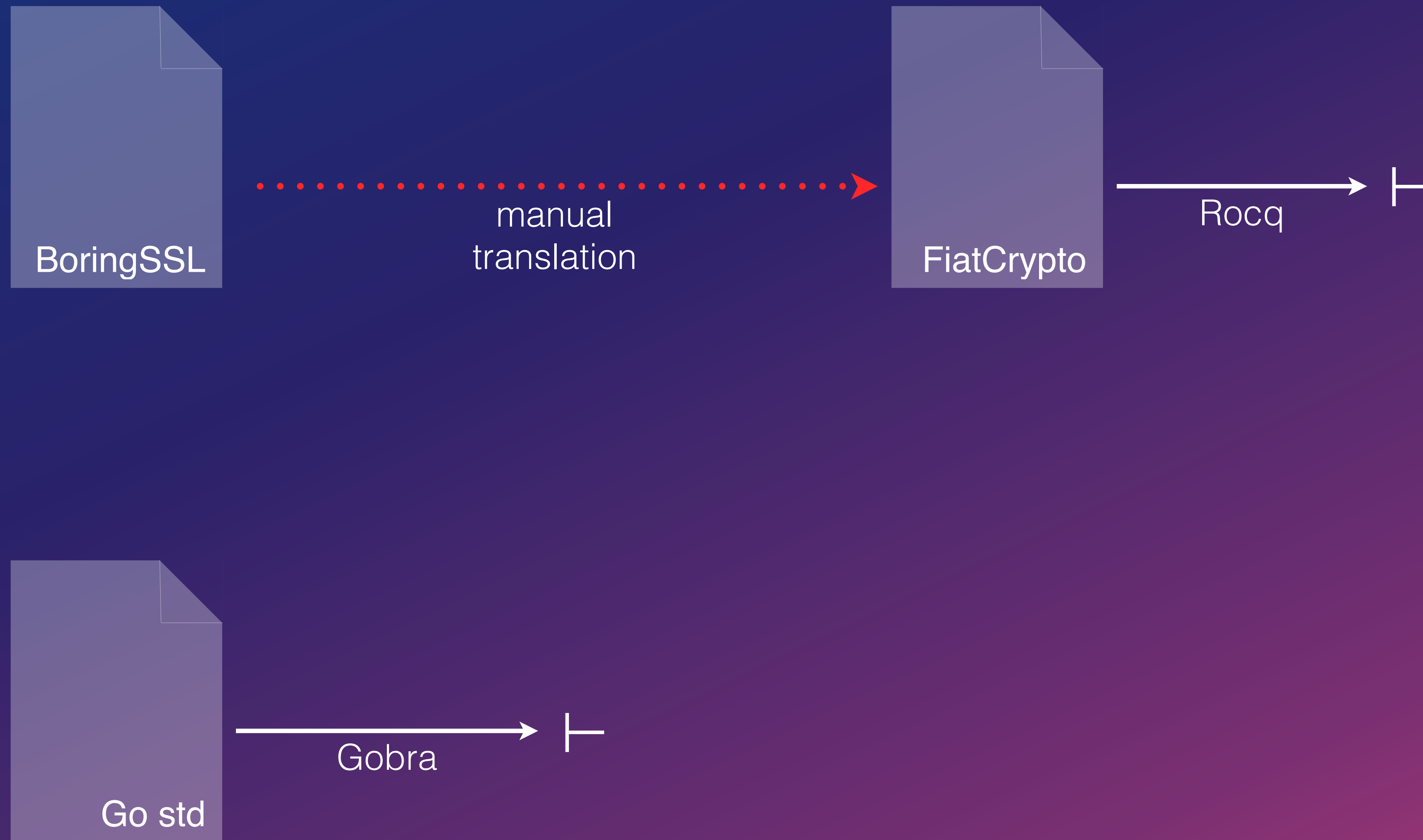
$$u = A * a - B * n$$

$$v = D * n - C * a$$

State of the Art



This Work



Contributions



Identify 2 deviations violating the algorithm's invariants

Fix results in a 24% speedup

Fix these deviations

Verify the fixed extended GCD implementation

A screenshot of the Gobra IDE interface. The window title is 'gcd'. The editor shows a Go file named 'example.go' with the following code:

```
1 package example
2
3
4
5
6
7
8
9
10 func mod(x, y int) (res int) {
11     return x % y
12 }
```

The code is color-coded: 'package' is red, 'example' is green, 'func' is red, 'mod' is purple, 'x' is orange, 'y' is green, 'int' is blue, 'res' is orange, and 'return' is red. The expression 'x % y' is underlined with a red wavy line. The bottom status bar shows a red 'Verification failed in 0.66s with:' message. The interface includes a sidebar with icons for file explorer, search, and settings, and a top bar with window management controls.



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

A screenshot of the goBRA IDE interface. The window title is 'gcd'. The editor shows a Go file 'example.go' with the following code:

```
1 package example
2
3
4
5
6
7
8
9 //@ requires y != 0
10 func mod(x, y int) (res int) {
11     return x % y
12 }
```

The interface includes a sidebar on the left with icons for file explorer, search, and settings. The bottom status bar shows 'Verification succeeded in 0.63s' in green text, along with icons for errors, warnings, and a 'Reload' button.



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

A screenshot of the goobra IDE interface. The top bar shows the file name 'gcd' and a tab for 'example.go 1'. The editor displays Go code with line numbers 1 through 7. Line 4 contains a postcondition assertion that is underlined with a red wavy line. Below the code, a red error banner indicates a problem. The error message states that the postcondition might not hold and the assertion might not hold. The code snippet shows a package declaration, a postcondition, and a function definition. The bottom status bar shows 'Verification failed in 0.66s with:' and a 'Reload' button.

```
1 package example
2
3 //@ requires y != 0
4 //@ ensures res >= 0
5
6 func mod(x, y int) (res int) {
7     return x % y
8 }
```

⊗ example.go 1 of 1 problem

Postcondition might not hold.
Assertion res >= 0 might not hold.

<< ⊗ 1 ⚠ 0 Verification failed in 0.66s with: {} Reload 🔔



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

A screenshot of the goobra IDE interface. The window title is 'gcd'. The editor shows a Go file named 'example.go' with the following code:

```
1 package example
2
3
4
5
6
7
8 //@ requires y != 0
9 //@ ensures x >= 0 ==> res >= 0
10 func mod(x, y int) (res int) {
11     return x % y
12 }
```

The interface includes a sidebar on the left with icons for file explorer, search, and settings. The bottom status bar shows 'Verification succeeded in 0.77s' and a 'Reload' button.

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

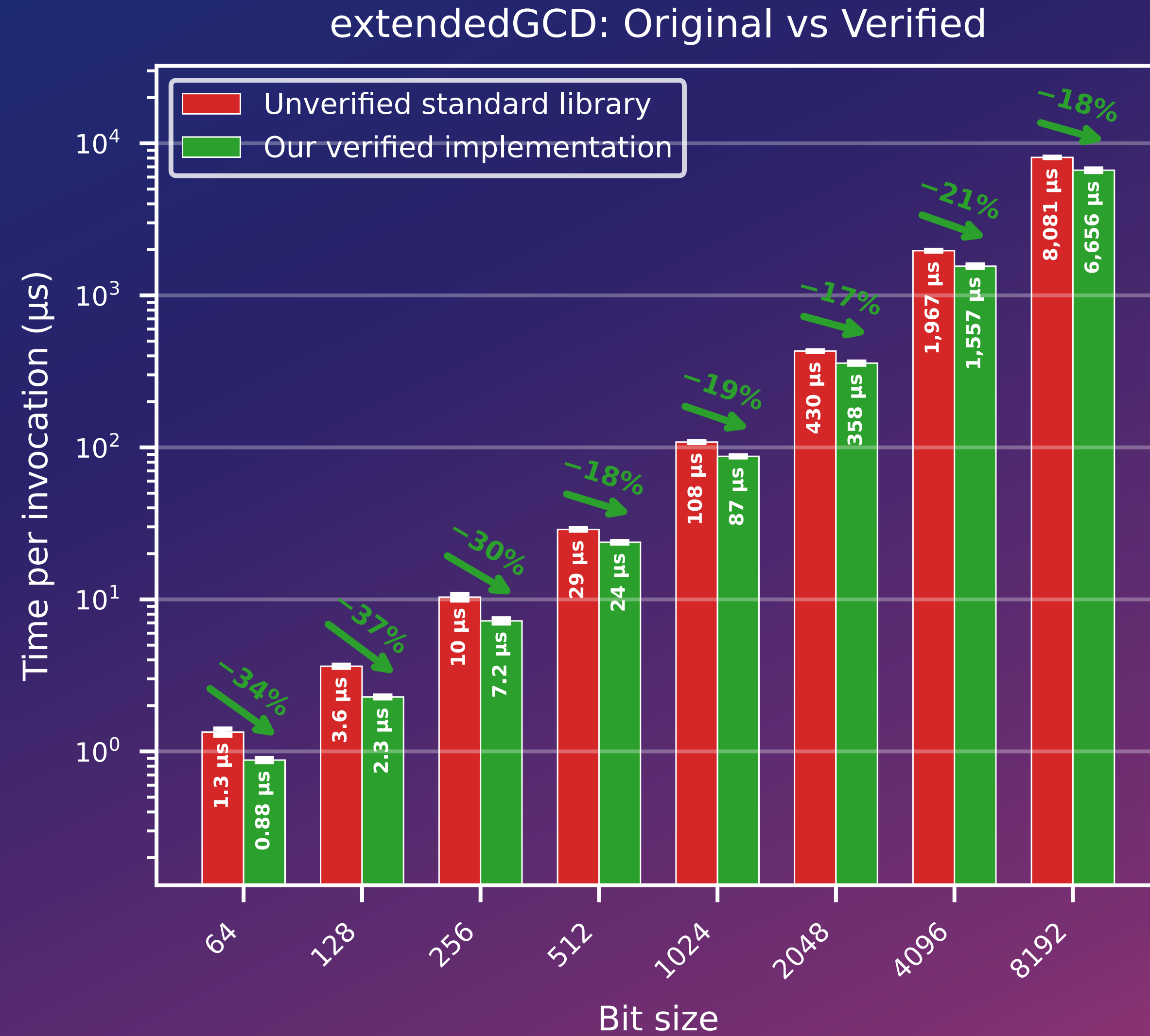
Review axiomatized lemmas &
check their Lean proofs

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

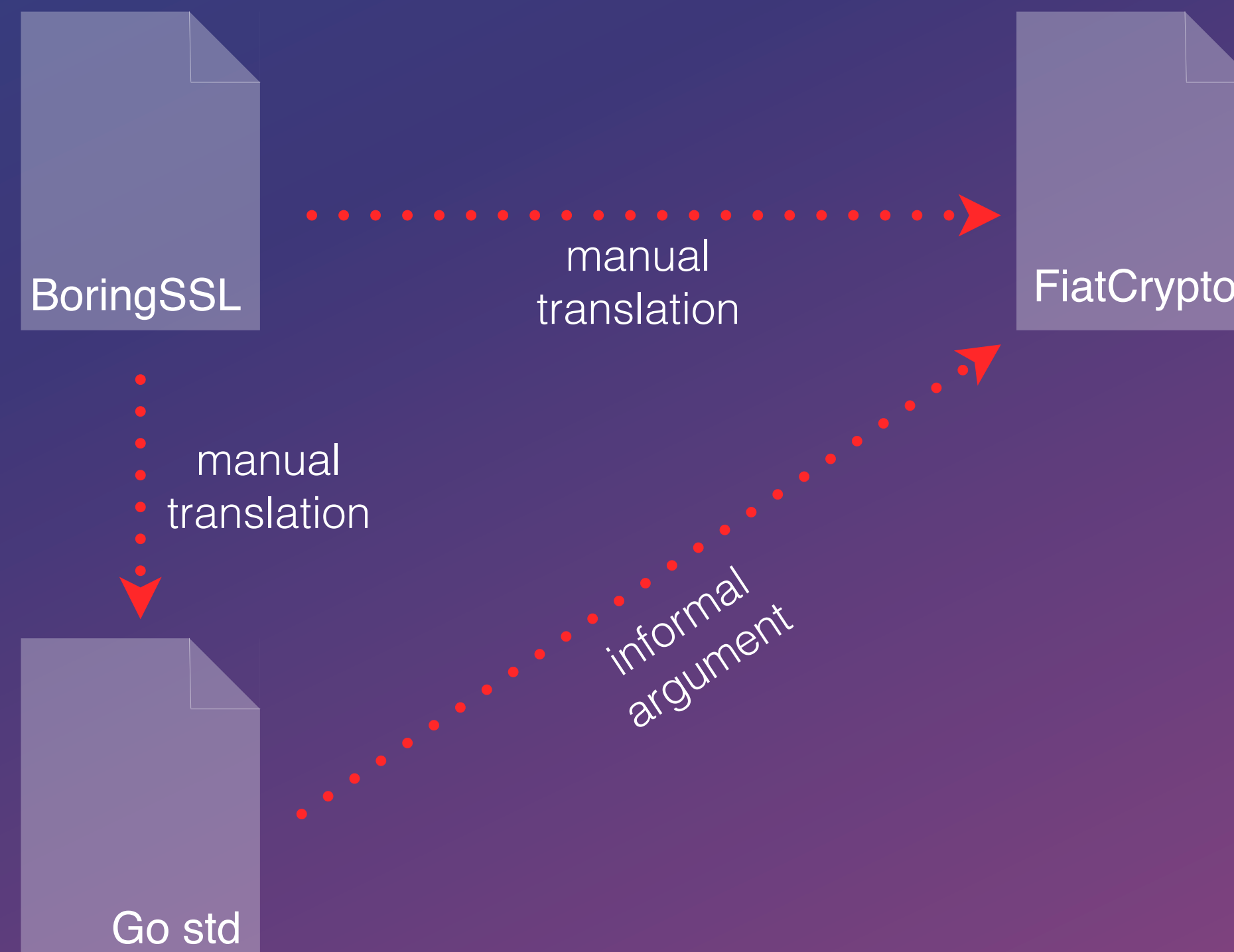
Evaluation

	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

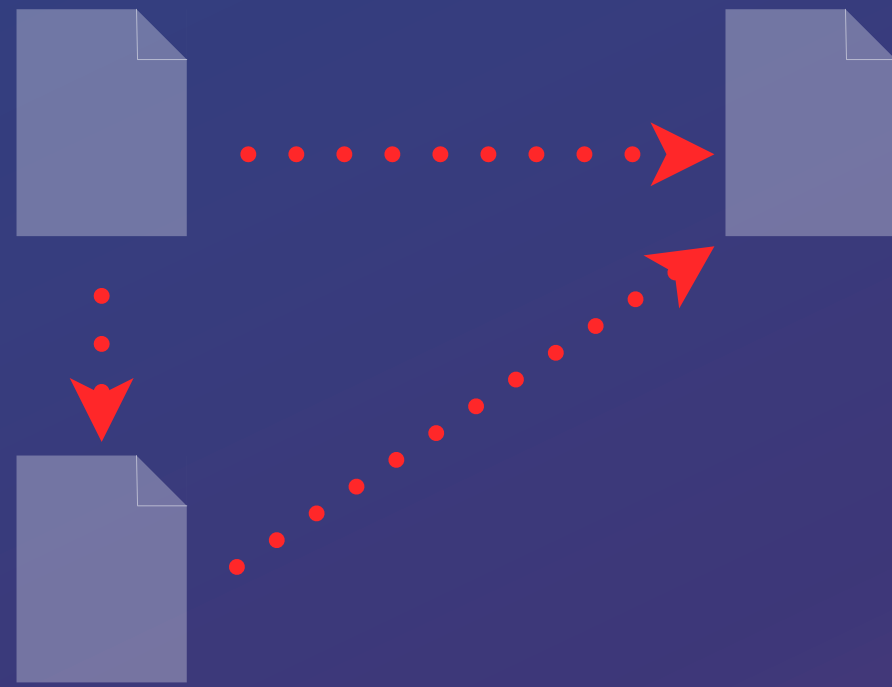


Conclusions



Translation as a soundness threat

Conclusions



Translation as a soundness threat



Upstreamed patch



Verified GCD implementation



Agentic verification