# CryptOpt: Verified Compilation with Randomized Program Search for Cryptographic Primitives

Chitchanok Chuengsatiansup

The University of Melbourne, Australia

Future Communications Workshop

Joint work with A. Chlipala, O. Conoly, A. Erbsen, D. Genkin, J. Gross, J. Kuepper, C. Sun, S. Tian, M. Wagner, D. Wu and Y. Yarom





# Motivation

# Motivation

- Correct: produce expected output

# Motivation

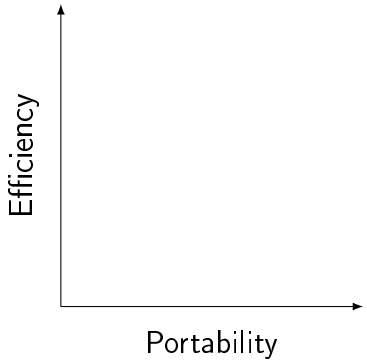
- Correct: produce expected output
- Efficient: high-speed high-security

# Motivation

- Correct: produce expected output
- Efficient: high-speed high-security
- Portable: applicable for different devices

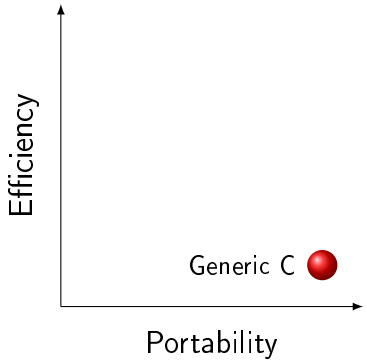
# Motivation

- Correct: produce expected output
- Efficient: high-speed high-security
- Portable: applicable for different devices



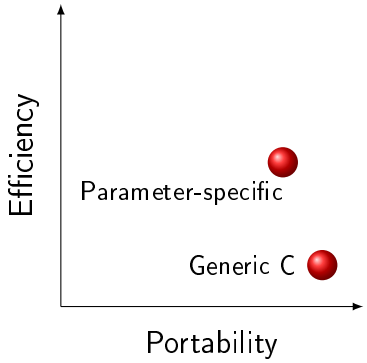
# Motivation

- Correct: produce expected output
- Efficient: high-speed high-security
- Portable: applicable for different devices



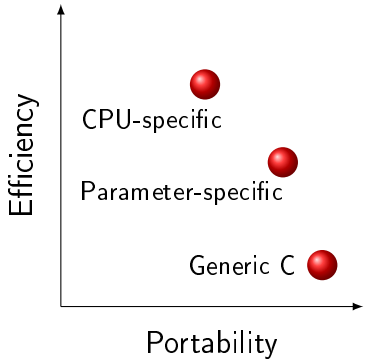
# Motivation

- Correct: produce expected output
- Efficient: high-speed high-security
- Portable: applicable for different devices



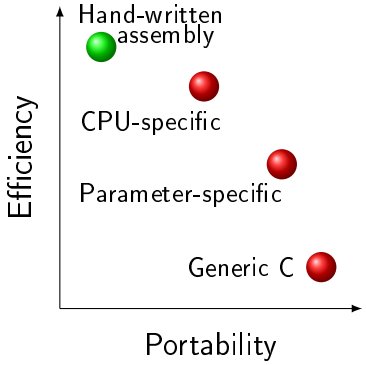
# Motivation

- Correct: produce expected output
- Efficient: high-speed high-security
- Portable: applicable for different devices



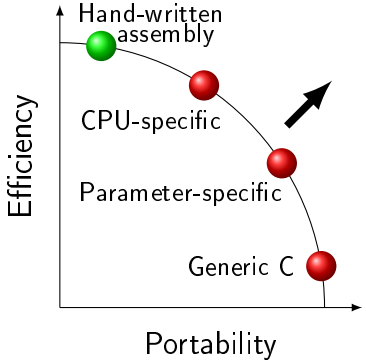
# Motivation

- Correct: produce expected output
- Efficient: high-speed high-security
- Portable: applicable for different devices



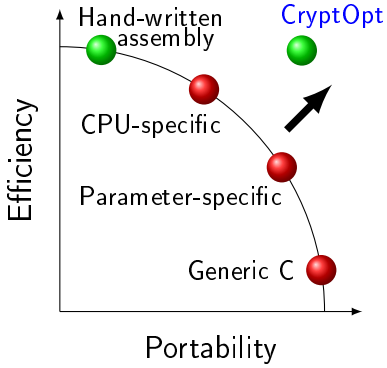
# Motivation

- Correct: produce expected output
- Efficient: high-speed high-security
- Portable: applicable for different devices



# Motivation

- Correct: produce expected output
- Efficient: high-speed high-security
- Portable: applicable for different devices



# Observation

# Observation

- Compilers are general-purpose

# Observation

- Compilers are general-purpose
- Cryptographic code has “special” structures

# Constant-Time Programming

# Constant-Time Programming

- No *secret-dependent* control flow

# Constant-Time Programming

- No *secret-dependent* control flow

- No *secret-dependent* memory access

# Constant-Time Programming

- No *secret-dependent* control flow
- No *secret-dependent* memory access
- No *secret-dependent* variable-time instruction

# Constant-Time Programming

- No *secret-dependent* control flow
  - CryptOpt: straight-line code
- No *secret-dependent* memory access
- No *secret-dependent* variable-time instruction

# Constant-Time Programming

- No *secret-dependent* control flow
  - CryptOpt: straight-line code
- No *secret-dependent* memory access
  - CryptOpt: fixed memory offset
- No *secret-dependent* variable-time instruction

# Constant-Time Programming

- No *secret-dependent* control flow
  - CryptOpt: straight-line code
- No *secret-dependent* memory access
  - CryptOpt: fixed memory offset
- No *secret-dependent* variable-time instruction
  - CryptOpt: constant-time instruction

# Optimization Strategies

# Optimization Strategies

- Straight-line code in static single assignment (SSA)

# Optimization Strategies

- Straight-line code in static single assignment (SSA)
  - ensure constant-time code

# Optimization Strategies

- Straight-line code in static single assignment (SSA)
  - ensure constant-time code
- Combinatorial optimization

# Optimization Strategies

- Straight-line code in static single assignment (SSA)
  - ensure constant-time code
- Combinatorial optimization
  - search for best-performing implementation

# Optimization Strategies

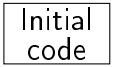
- Straight-line code in static single assignment (SSA)
  - ensure constant-time code
- Combinatorial optimization
  - search for best-performing implementation
- Random local search (RLS) with bet-and-run heuristic

# Optimization Strategies

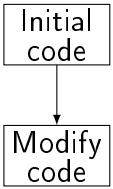
- Straight-line code in static single assignment (SSA)
  - ensure constant-time code
- Combinatorial optimization
  - search for best-performing implementation
- Random local search (RLS) with bet-and-run heuristic
  - "bet" explores up to budget then "run" continues from the best

# Search for Fast Implementation

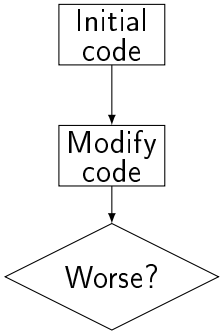
# Search for Fast Implementation



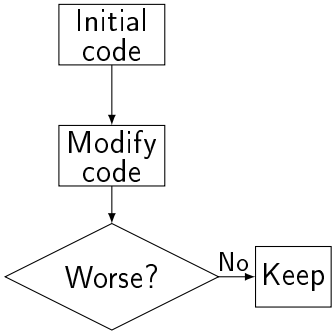
# Search for Fast Implementation



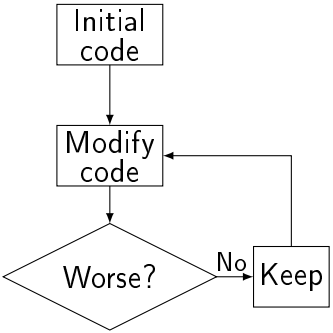
# Search for Fast Implementation



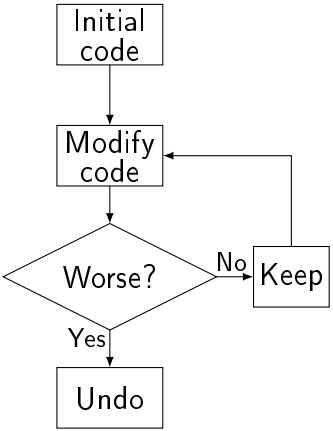
# Search for Fast Implementation



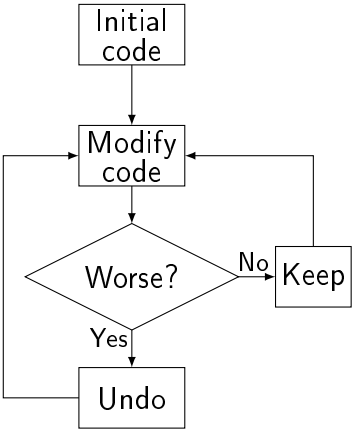
# Search for Fast Implementation



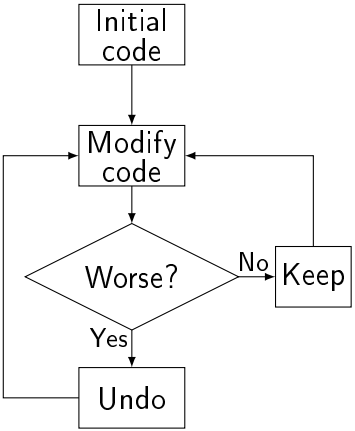
# Search for Fast Implementation



# Search for Fast Implementation

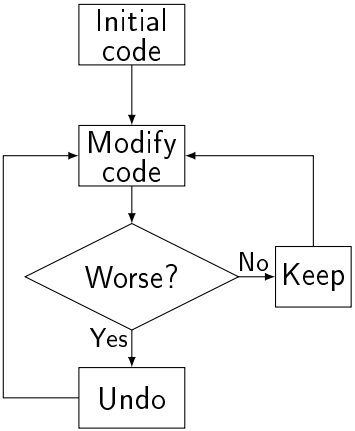


# Search for Fast Implementation

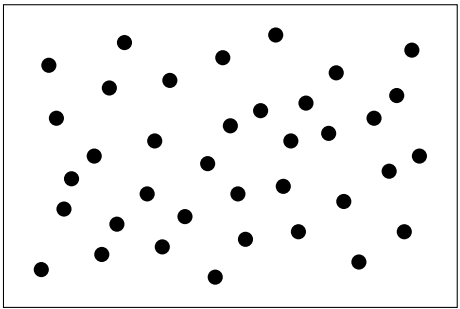



Random Local Search

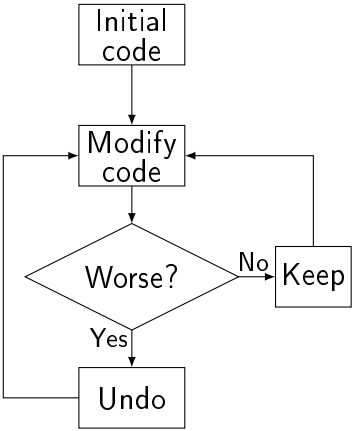
# Search for Fast Implementation



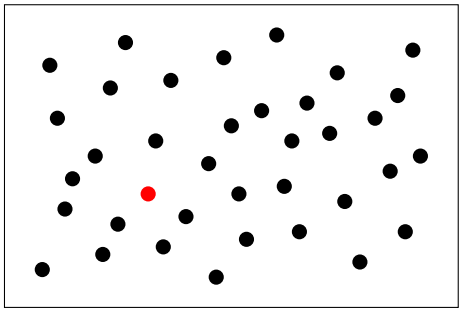

Random Local Search



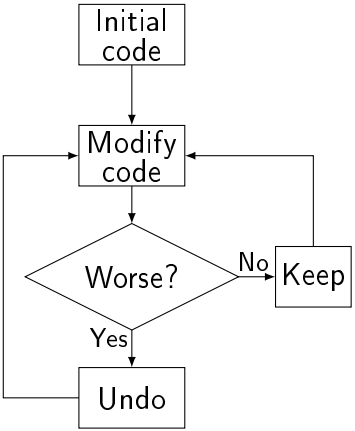
# Search for Fast Implementation



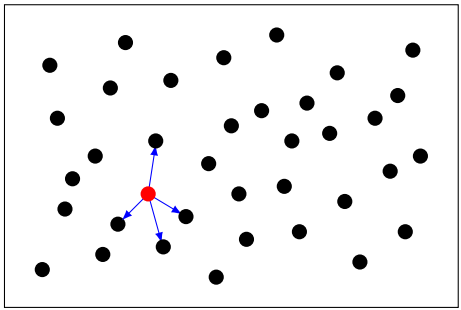

Random Local Search



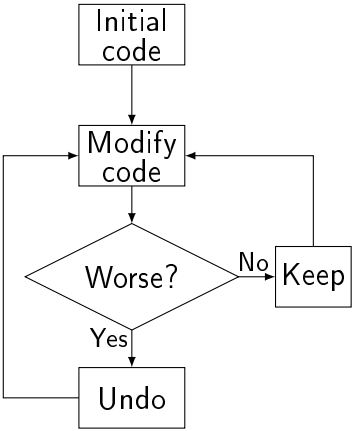
# Search for Fast Implementation



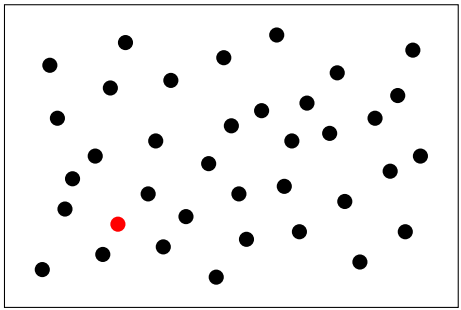

Random Local Search



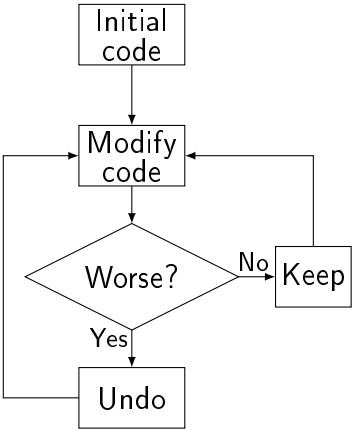
# Search for Fast Implementation



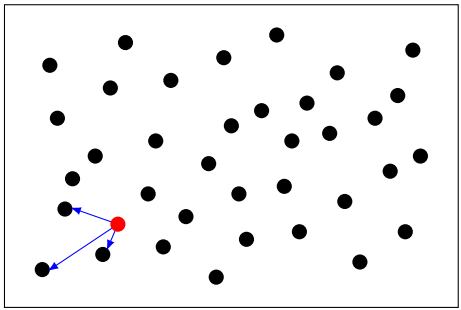

Random Local Search



# Search for Fast Implementation

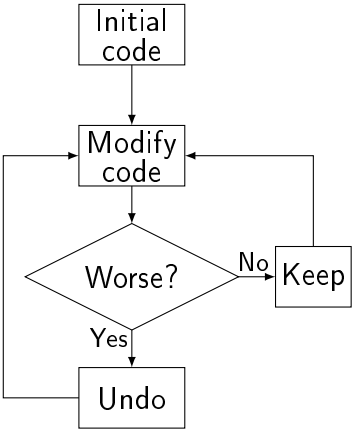



Random Local Search

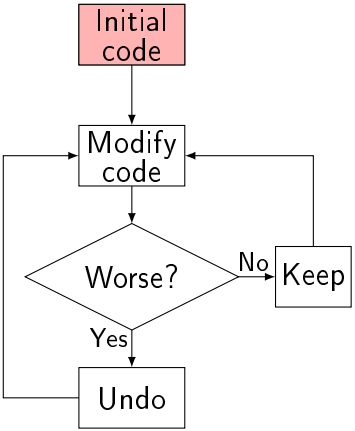


# Example Function: $(X + Y) \cdot Z + Z^2$

# Example Function: $(X + Y) \cdot Z + Z^2$

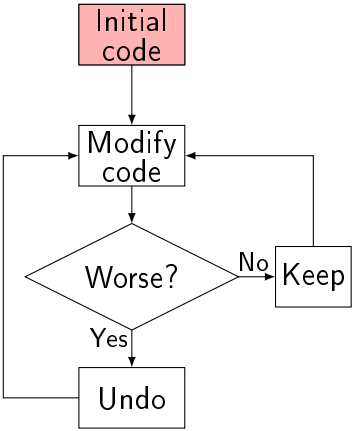


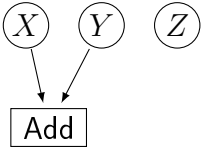
# Example Function: $(X + Y) \cdot Z + Z^2$



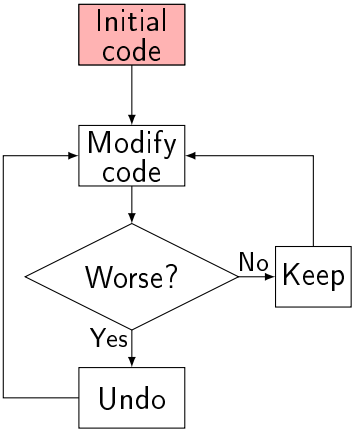

$X$ $Y$ $Z$

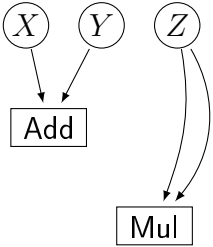
# Example Function: $(X + Y) \cdot Z + Z^2$



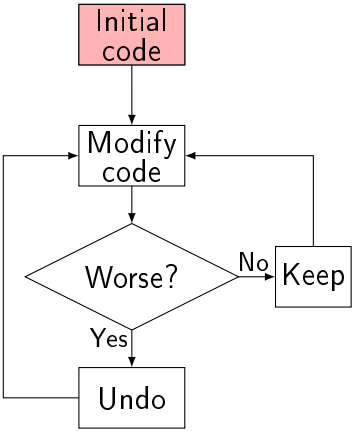



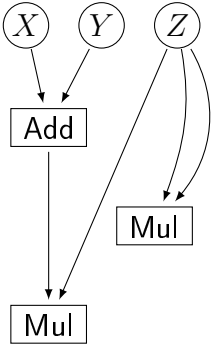
# Example Function: $(X + Y) \cdot Z + Z^2$



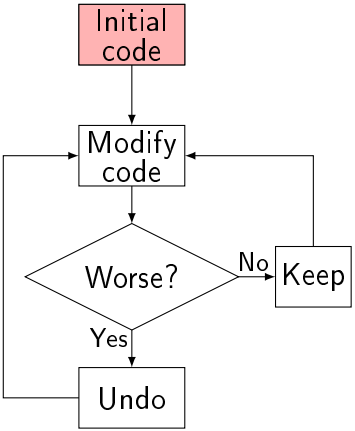



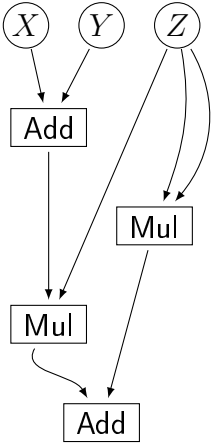
# Example Function: $(X + Y) \cdot Z + Z^2$



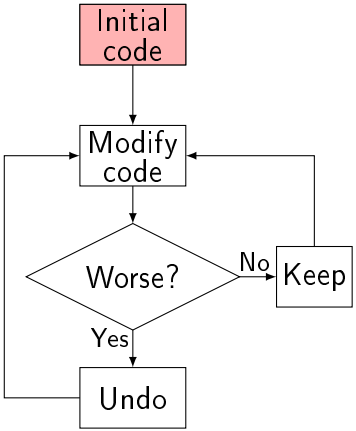



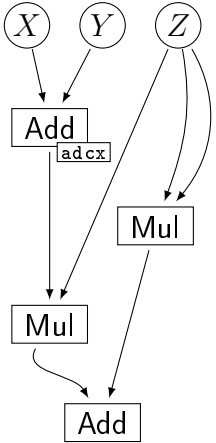
# Example Function: $(X + Y) \cdot Z + Z^2$



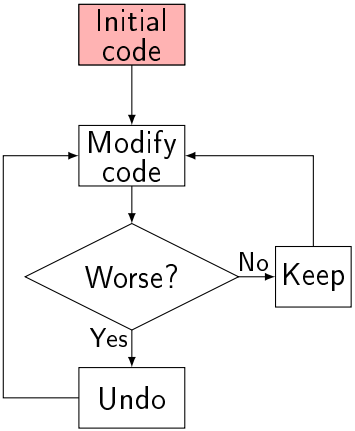
# Example Function: $(X + Y) \cdot Z + Z^2$

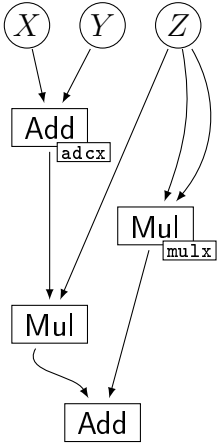



```
mov rax, [X]
clc
adcx rax, [Y]
```

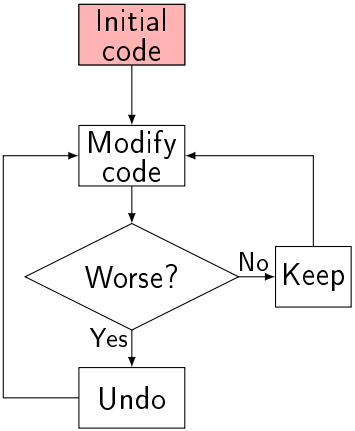
# Example Function: $(X + Y) \cdot Z + Z^2$

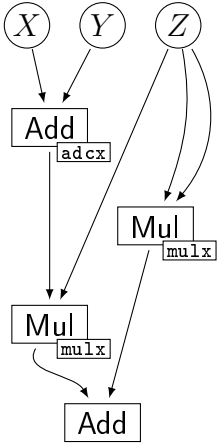



```
mov rax, [X]
clc
adcx rax, [Y]

mov rdx, [Z]
mulx r8, r9, rax
```

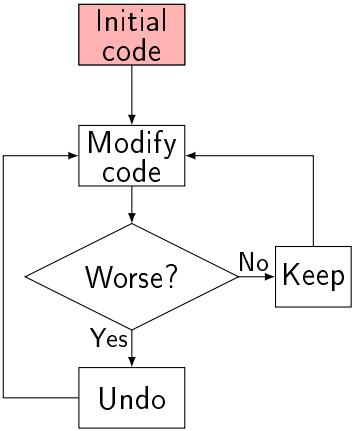
# Example Function: $(X + Y) \cdot Z + Z^2$

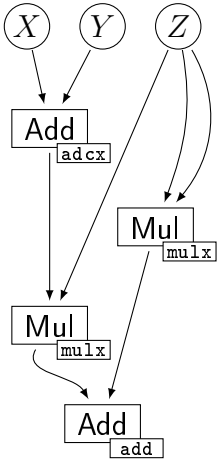



```
mov rax, [X]
clc
adcx rax, [Y]

mov rdx, [Z]
mulx r8, r9, rax


mulx r10, r11, [Z]
```

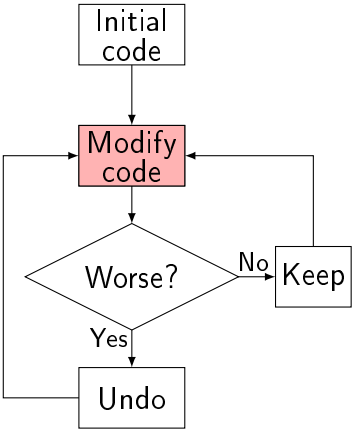
# Example Function: $(X + Y) \cdot Z + Z^2$

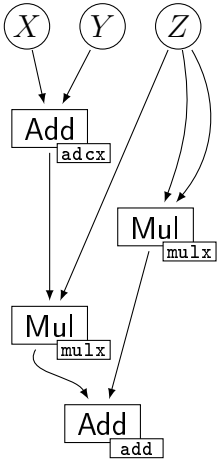



```
mov rax, [X]
clc
adcx rax, [Y]

mov rdx, [Z]
mulx r8, r9, rax

mulx r10, r11, [Z]

add r11, r9
mov [out], r11
```

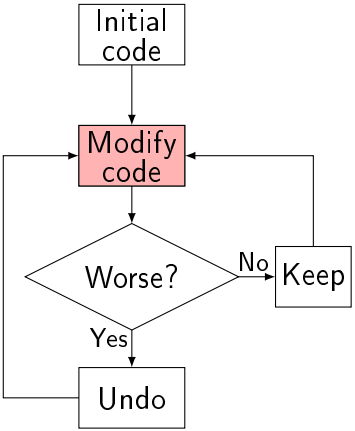
# Example Function: $(X + Y) \cdot Z + Z^2$

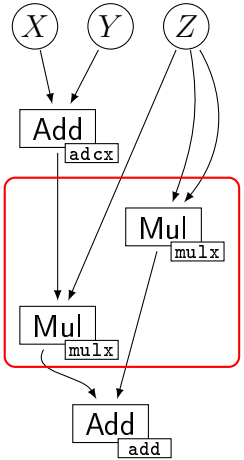



```
mov rax, [X]
clc
adcx rax, [Y]

mov rdx, [Z]
mulx r8, r9, rax

mulx r10, r11, [Z]

add r11, r9
mov [out], r11
```

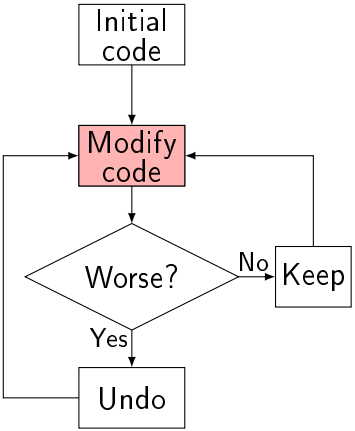
# Example Function: $(X + Y) \cdot Z + Z^2$

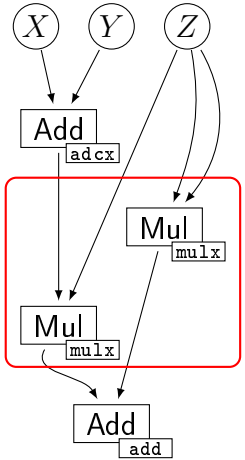



```
mov rax, [X]
clc
adcx rax, [Y]

mov rdx, [Z]
mulx r8, r9, rax

mulx r10, r11, [Z]

add r11, r9
mov [out], r11
```

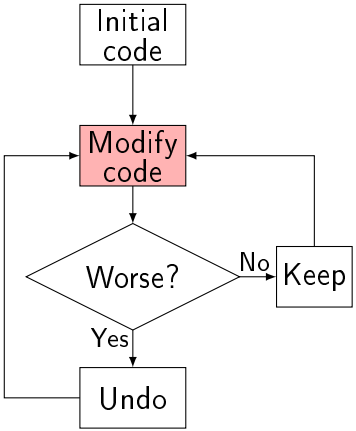
# Example Function: $(X + Y) \cdot Z + Z^2$

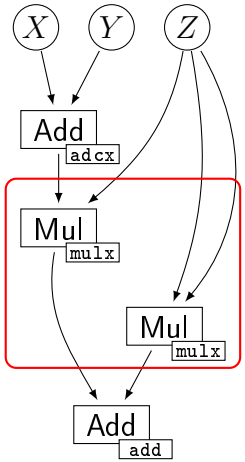



```
mov rax, [X]
clc
adcx rax, [Y]

mov rdx, [Z]
mulx r8, r9, rax


mulx r10, r11, [Z]

add r11, r9
mov [out], r11
```

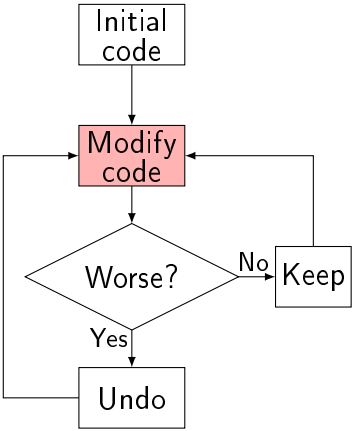
# Example Function: $(X + Y) \cdot Z + Z^2$ [reorder]

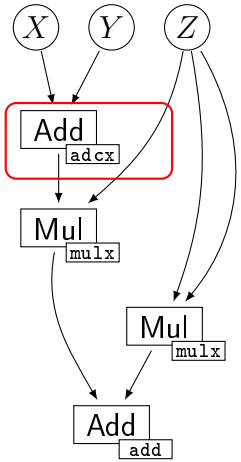



```
mov rax, [X]
clc
adcx rax, [Y]

mov rdx, [Z]
mulx r10, r11, [Z]


mulx r8, r9, rax


add r11, r9
mov [out], r11
```

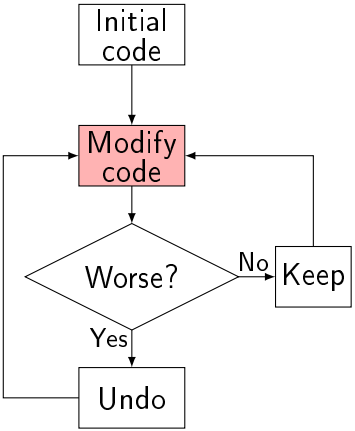
# Example Function: $(X + Y) \cdot Z + Z^2$ [template]

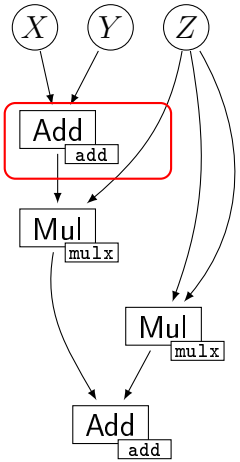



```
mov rax, [X]
clc
adcx rax, [Y]

mov rdx, [Z]
mulx r10, r11, [Z]

mulx r8, r9, rax

add r11, r9
mov [out], r11
```

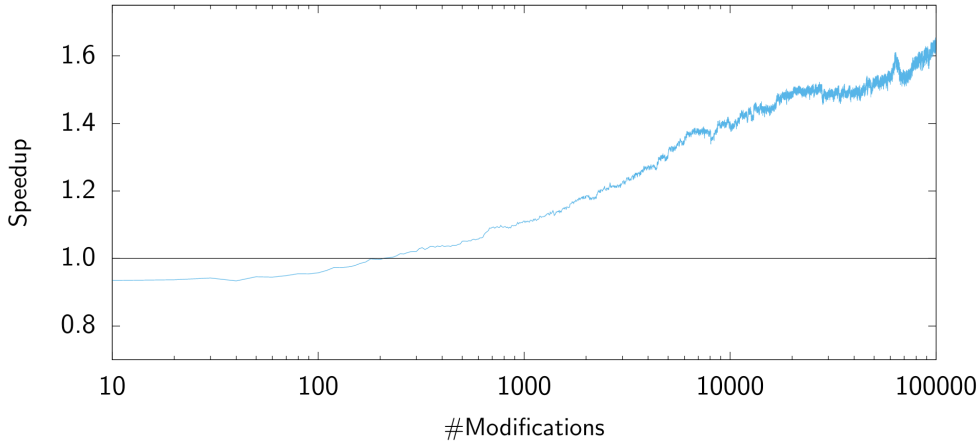
# Example Function: $(X + Y) \cdot Z + Z^2$ [template]



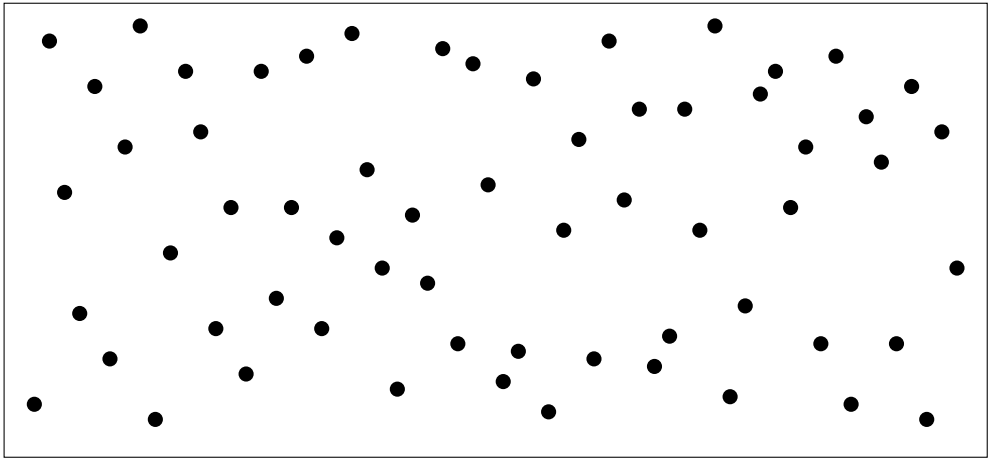




```
mov rax, [X]
clc
add rax, [Y]

mov rdx, [Z]
mulx r10, r11, [Z]

mulx r8, r9, rax

add r11, r9
mov [out], r11
```

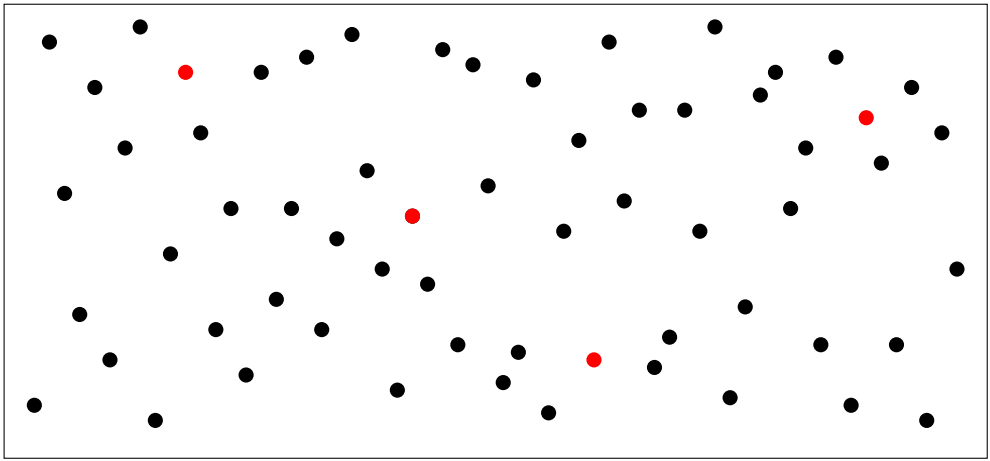
# Optimization Progress



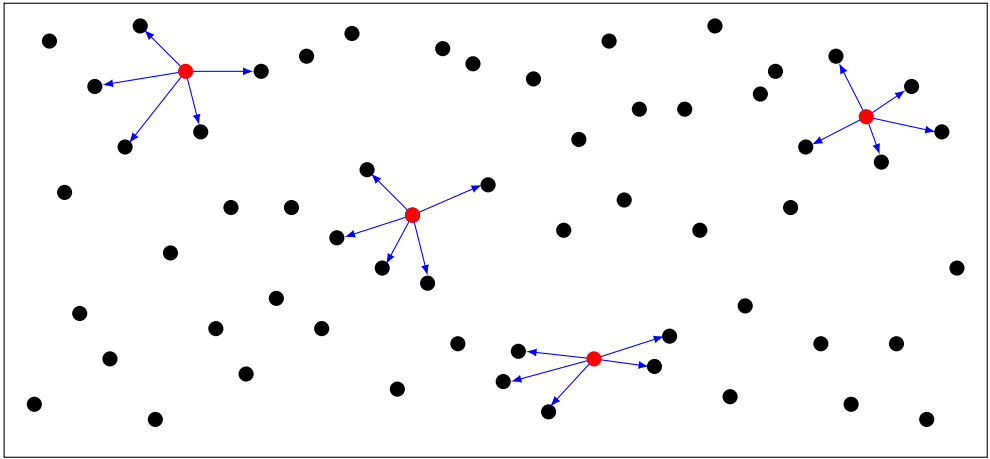
# Random Local Search with Bet-and-Run



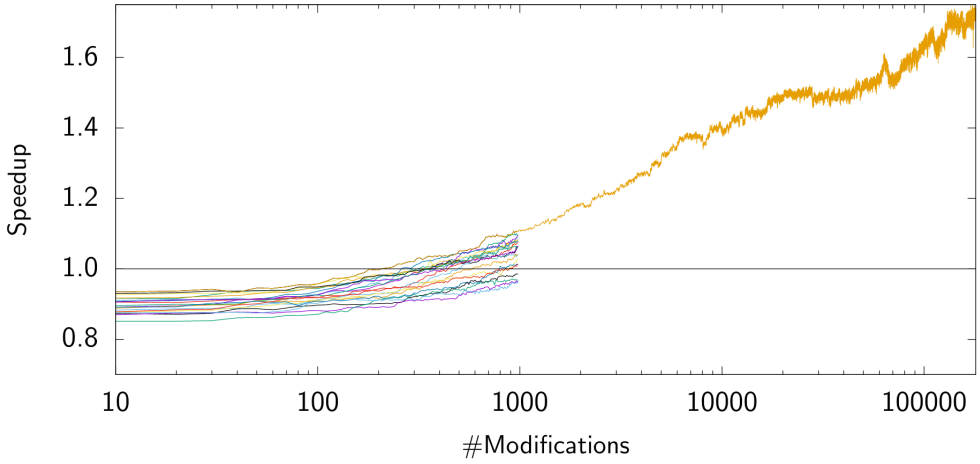
# Random Local Search with Bet-and-Run



# Random Local Search with Bet-and-Run



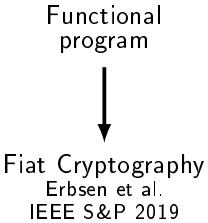
# Bet-and-Run in Action



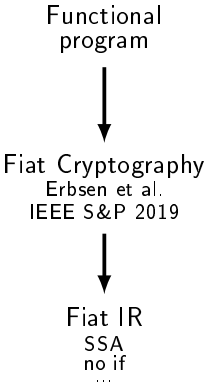
# Fiat Cryptography

Fiat Cryptography
Erbsen et al.
IEEE S&P 2019

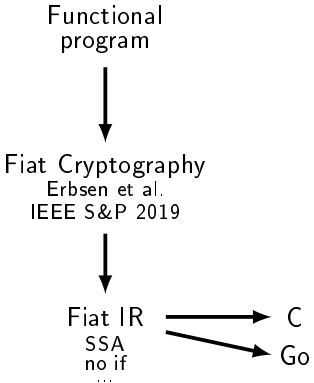
# Fiat Cryptography



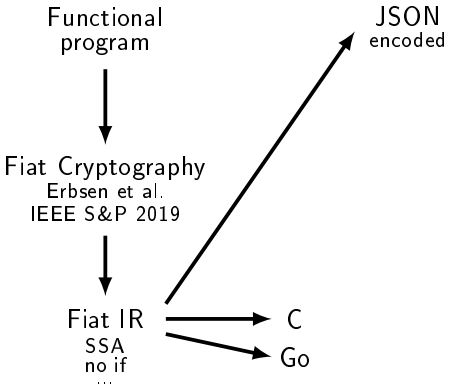
# Fiat Cryptography



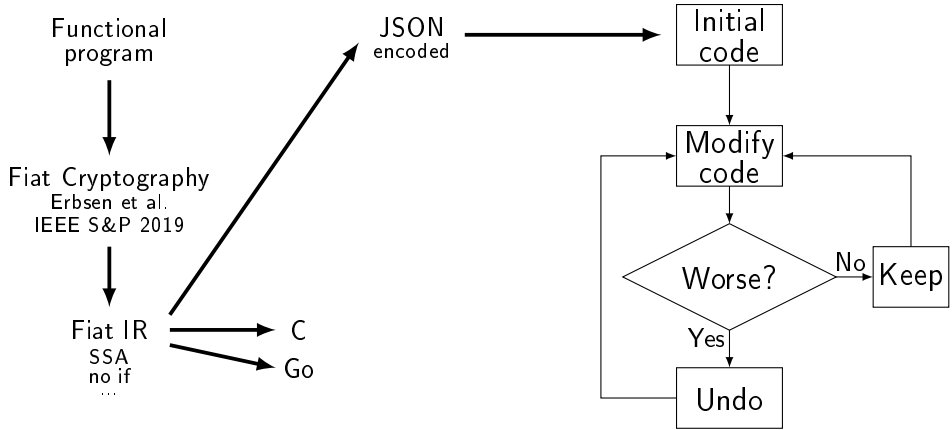
# Fiat Cryptography



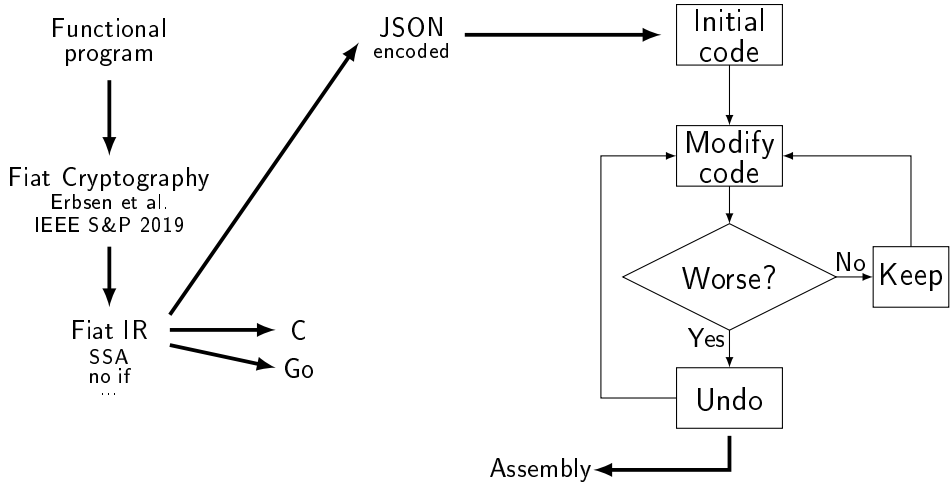
# Fiat Cryptography



# Fiat Cryptography



# Fiat Cryptography



# Performance: Field Arithmetic

Geometric Mean (4x AMD, 6x Intel)

| | Multiply | | Square | |
|---|---|---|---|---|
| Curve | Clang | GCC | Clang | GCC |
| Curve25519 | | | | |
| P-224 | | | | |
| P-256 | | | | |
| P-384 | | | | |
| SIKEp434 | | | | |
| Curve448 | | | | |
| P-521 | | | | |
| Poly1305 | | | | |
| secp256k1 | | | | |

# Performance: Field Arithmetic

Geometric Mean (4x AMD, 6x Intel)

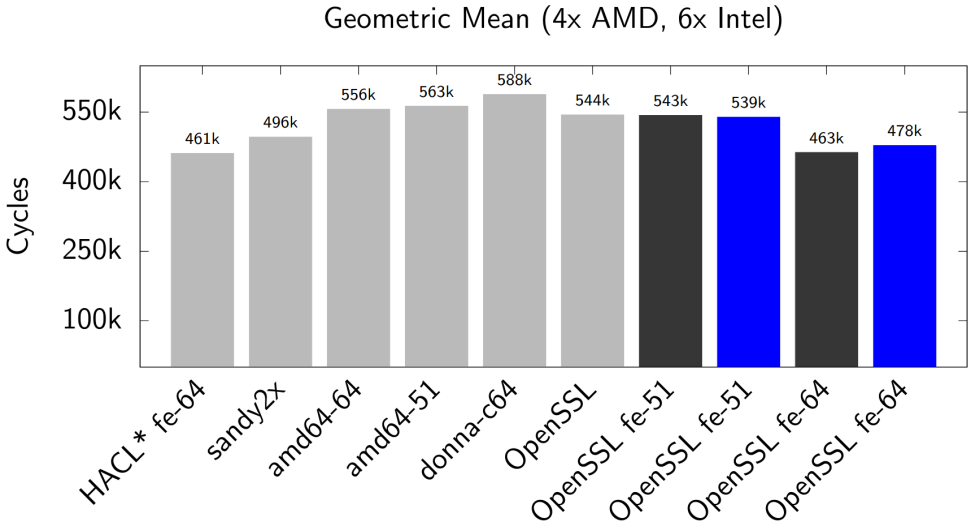
| | Multiply | | Square | |
|---|---|---|---|---|
| Curve | Clang | GCC | Clang | GCC |
| Curve25519 | 1.19 | 1.14 | 1.14 | 1.18 |
| P-224 | | | | |
| P-256 | | | | |
| P-384 | | | | |
| SIKEp434 | | | | |
| Curve448 | | | | |
| P-521 | | | | |
| Poly1305 | | | | |
| secp256k1 | | | | |

# Performance: Field Arithmetic

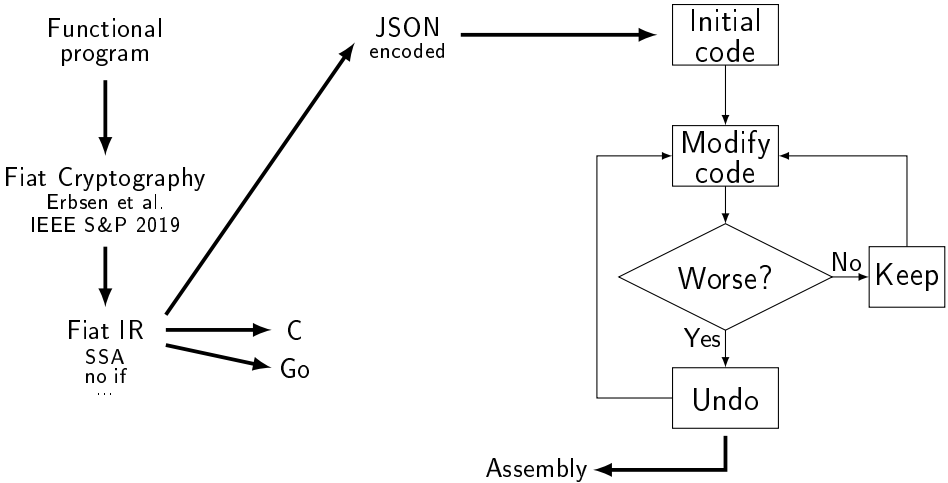
Geometric Mean (4x AMD, 6x Intel)

| | Multiply | | Square | |
|---|---|---|---|---|
| Curve | Clang | GCC | Clang | GCC |
| Curve25519 | 1.19 | 1.14 | 1.14 | 1.18 |
| P-224 | 1.31 | 1.87 | 1.24 | 1.84 |
| P-256 | 1.27 | 1.79 | 1.30 | 1.85 |
| P-384 | 1.12 | 1.66 | 1.08 | 1.60 |
| SIKEp434 | 1.30 | 1.70 | 1.29 | 1.83 |
| Curve448 | 1.02 | 0.95 | 1.00 | 0.99 |
| P-521 | 1.20 | 1.06 | 1.25 | 1.11 |
| Poly1305 | 1.10 | 1.15 | 1.09 | 1.16 |
| secp256k1 | 1.34 | 1.73 | 1.32 | 1.74 |

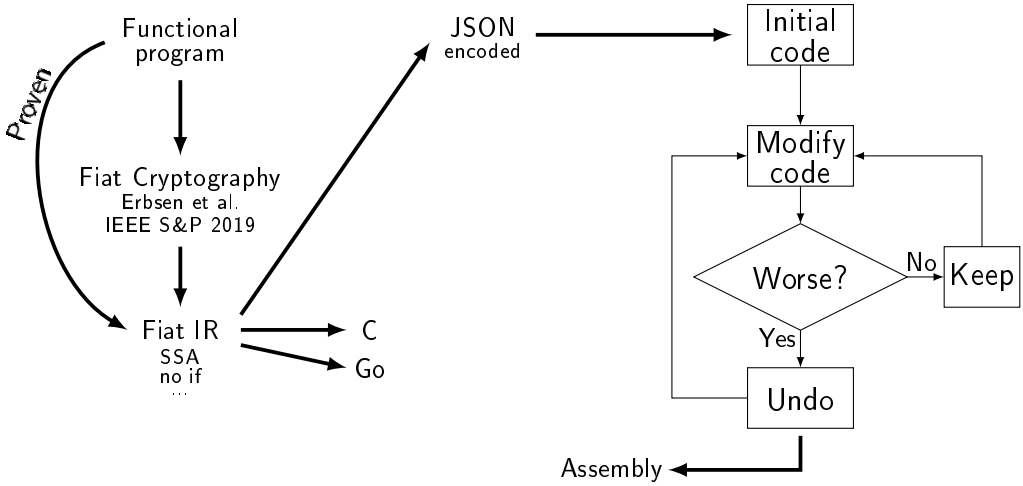
# Performance: Scalar Multiplication



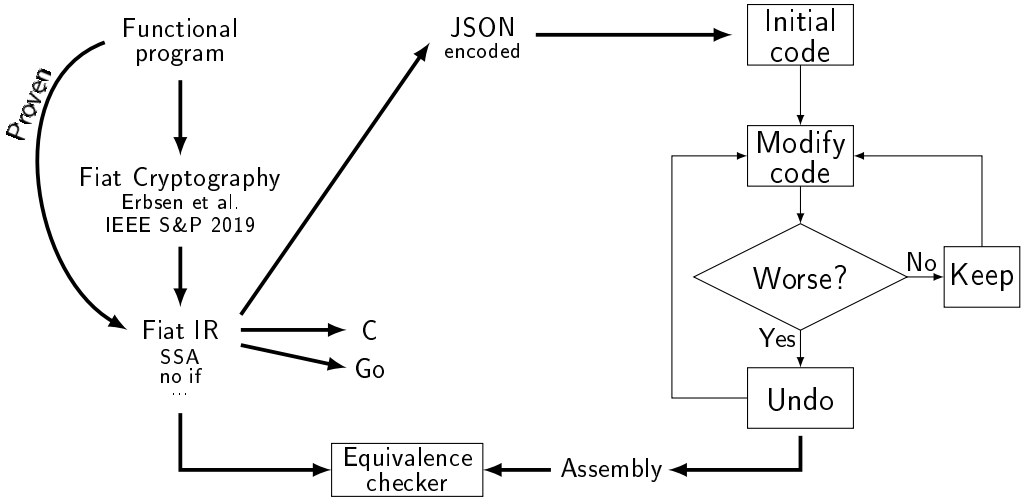
# Correctness



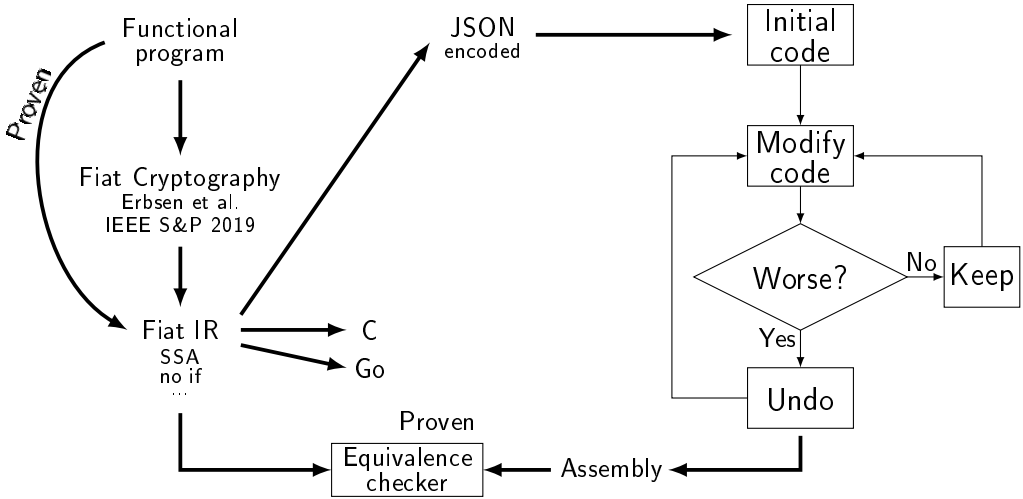
# Correctness



# Correctness



# Correctness



# Achievements

# Achievements

- CryptOpt: automatic cryptographic code optimizer

# Achievements

- CryptOpt: automatic cryptographic code optimizer
- Competitive performance with hand-written assembly

# Achievements

- CryptOpt: automatic cryptographic code optimizer
- Competitive performance with hand-written assembly
- Verified compilation of the generated code

# Achievements

- CryptOpt: automatic cryptographic code optimizer
- Competitive performance with hand-written assembly
- Verified compilation of the generated code
- Resistant to timing side channels

# Achievements

- CryptOpt: automatic cryptographic code optimizer
- Competitive performance with hand-written assembly
- Verified compilation of the generated code
- Resistant to timing side channels
- https://0xade1a1de.github.io/CryptOpt

GitHub Project

# Achievements

- CryptOpt: automatic cryptographic code optimizer
- Competitive performance with hand-written assembly
- Verified compilation of the generated code
- Resistant to timing side channels
- https://0xade1a1de.github.io/CryptOpt
- Distinguished Paper (PLDI 2023) & Humies GOLD Award (GECCO 2023)

GitHub Project

# Achievements

- CryptOpt: automatic cryptographic code optimizer
- Competitive performance with hand-written assembly
- Verified compilation of the generated code
- Resistant to timing side channels
- https://0xade1a1de.github.io/CryptOpt

GitHub Project

- Distinguished Paper (PLDI 2023) & Humies GOLD Award (GECCO 2023)
- Integrated in Google's products including Chromium-base browsers