CryptOpt

Verified Compilation with Randomized Program Search for Cryptographic Primitives

Joel Kuepper, Andres Erbsen, Jason Gross, Owen Conoly, Chuyue Sun, Samuel Tian, David Wu, Adam Chlipala, Chitchanok Chuengsatiansup, Daniel Genkin, Markus Wagner, Yuval Yarom

Motivation

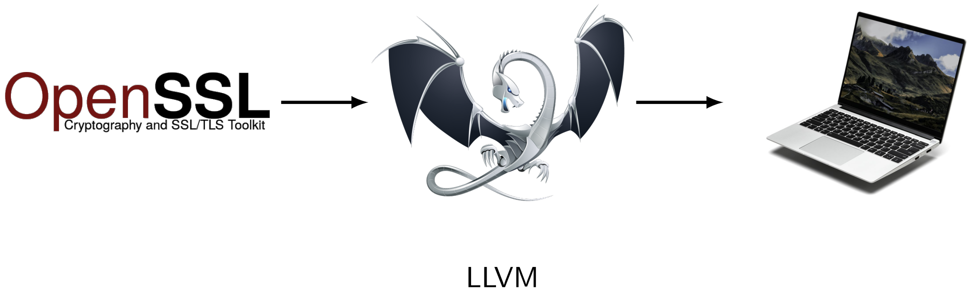
LLVM

Motivation

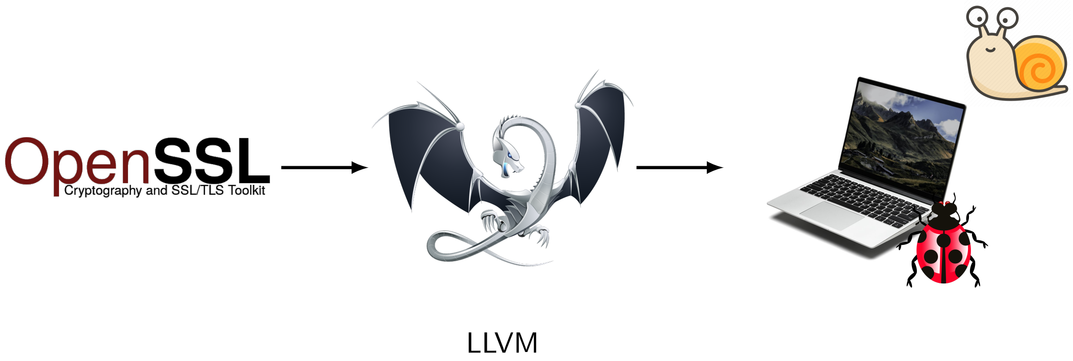
OpenSSL
Cryptography and SSL/TLS Toolkit
$\longrightarrow$

LLVM

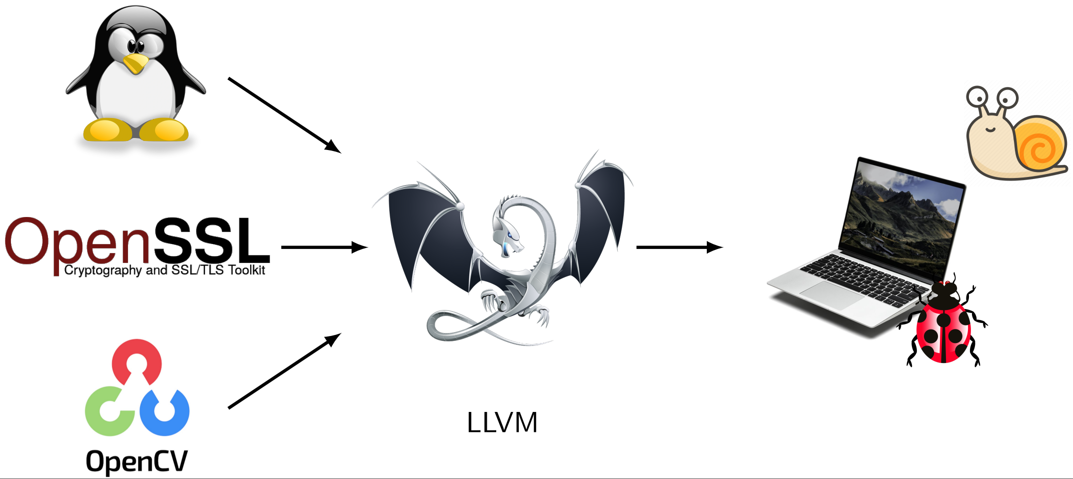
Motivation



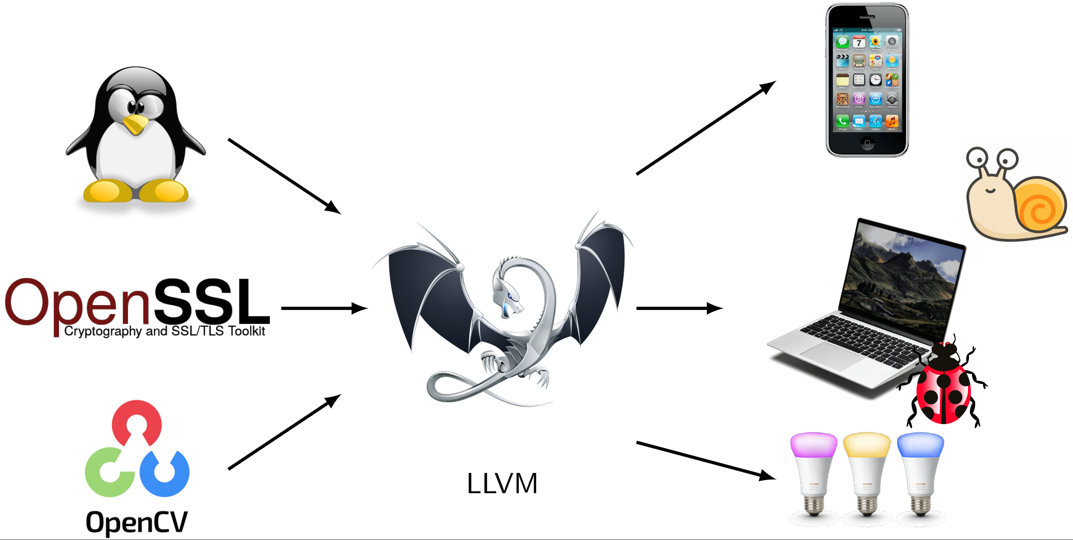
Motivation



Motivation

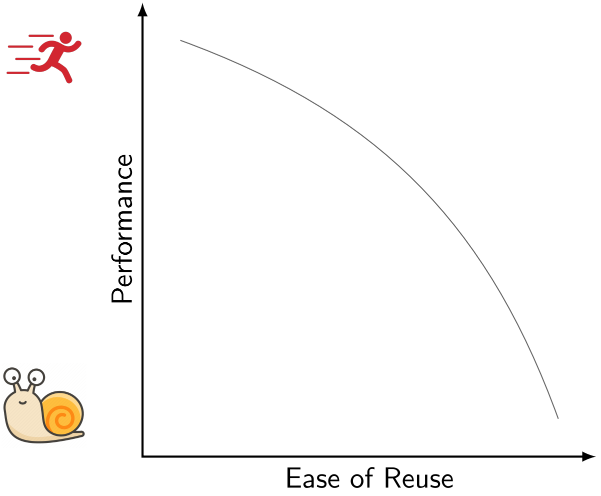


Motivation

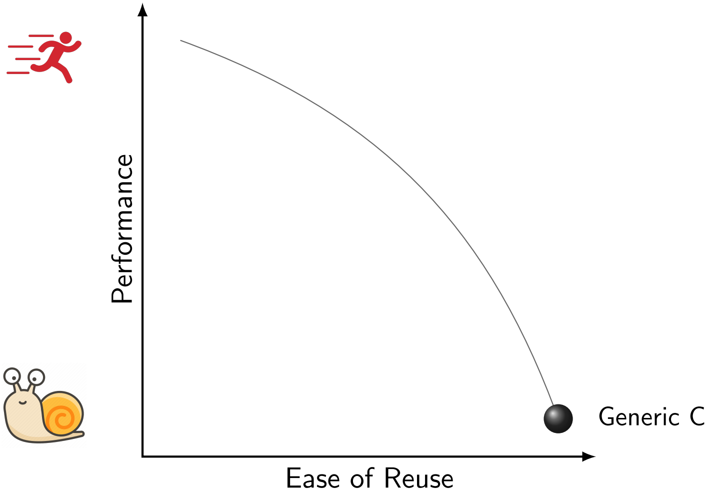


Code

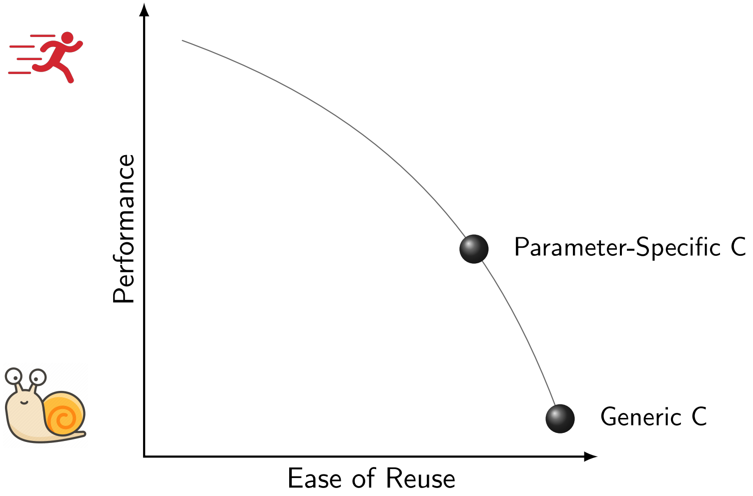
Code



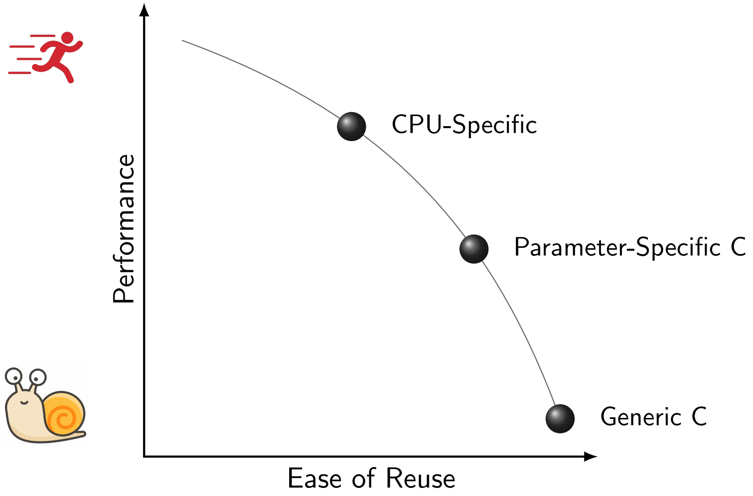
Code



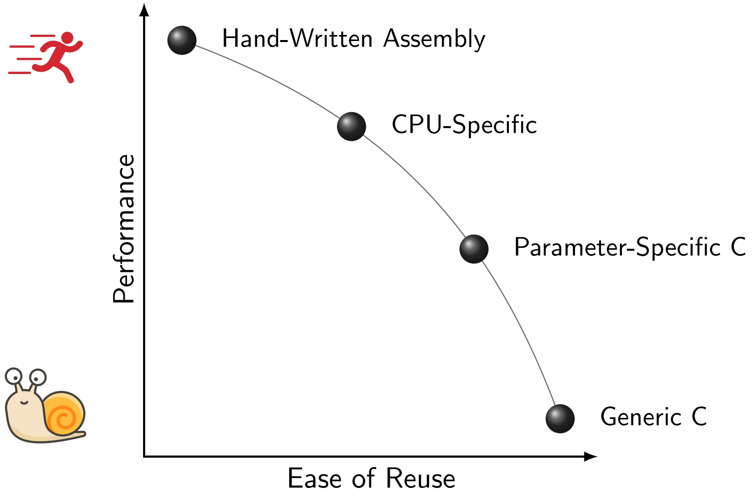
Code



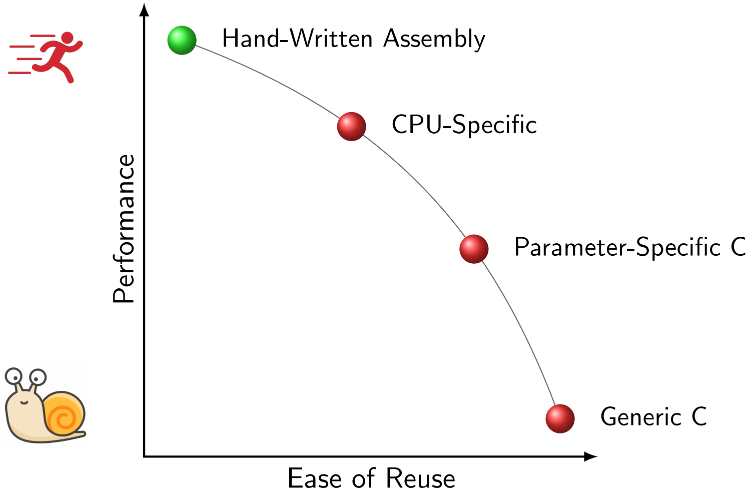
Code


Performance
Ease of Reuse
CPU-Specific
Parameter-Specific C
Generic C

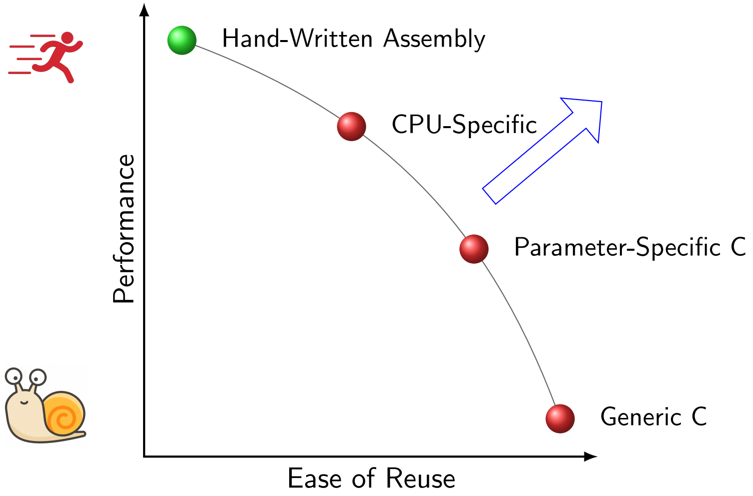
Code


Hand-Written Assembly
CPU-Specific
Parameter-Specific C
Generic C
Performance
Ease of Reuse

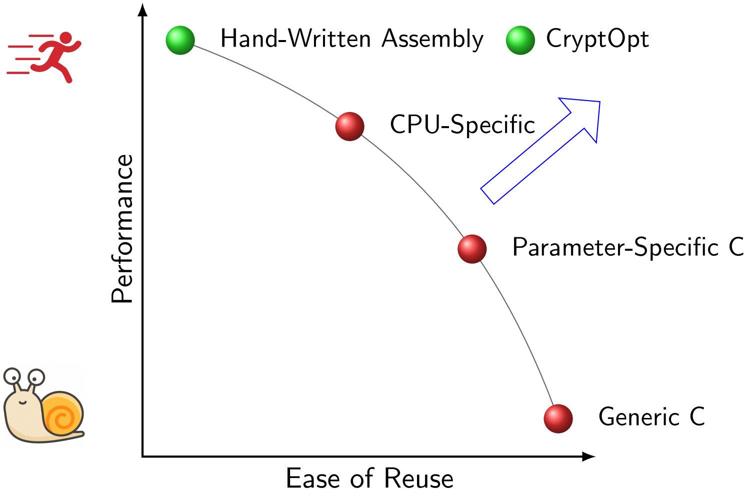
Code


Hand-Written Assembly
CPU-Specific
Parameter-Specific C
Generic C
Performance
Ease of Reuse

Code


Hand-Written Assembly
CPU-Specific
Parameter-Specific C
Generic C
Performance
Ease of Reuse

Code


Hand-Written Assembly
CryptOpt
CPU-Specific
Parameter-Specific C
Generic C
Performance
Ease of Reuse

Idea

Observations:

1. Compilers are general-purpose.

Idea

Observations:

1. Compilers are general-purpose.
2. Some code is “special”.

Idea

Observations:

1. Compilers are general-purpose.
2. Some code is "~~special~~ simpler" $(X + Y) \cdot Z + Z^2$.

Idea

Observations:

1. Compilers are general-purpose.
2. Some code is "~~special~~ simpler" $(X + Y) \cdot Z + Z^2$.

(How) can we exploit this?

Idea

Observations:

1. Compilers are general-purpose.
2. Some code is "~~special~~ simpler" $(X + Y) \cdot Z + Z^2$.

(How) can we exploit this?

So. . .

1. ~~Compiling to~~ $\rightarrow$ search for a fast implementation.

Idea

Observations:

1. Compilers are general-purpose.
2. Some code is "~~special~~ simpler" $(X + Y) \cdot Z + Z^2$.

(How) can we exploit this?

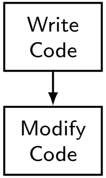
So...

1. ~~Compiling to~~ $\rightarrow$ search for a fast implementation.
2. Prove it correct.

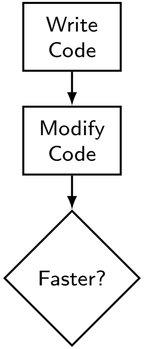
Search for Fast Implementation

Search for Fast Implementation

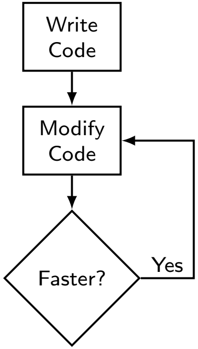
Search for Fast Implementation



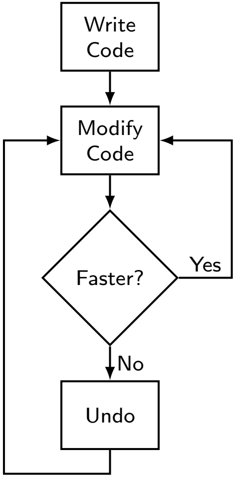
Search for Fast Implementation



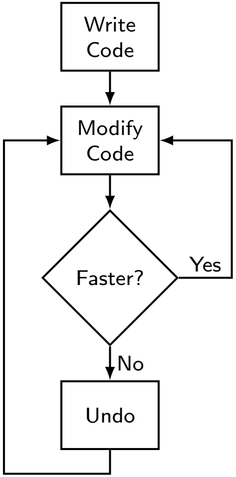
Search for Fast Implementation



Search for Fast Implementation



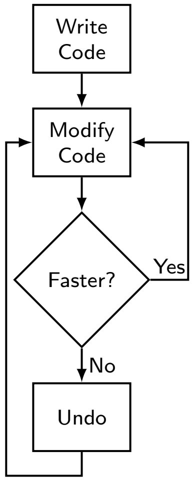
Search for Fast Implementation



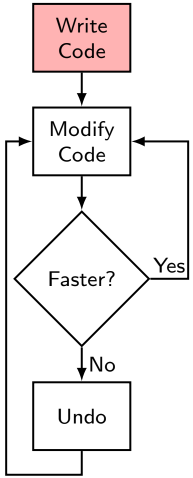
“Random Local Search”

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

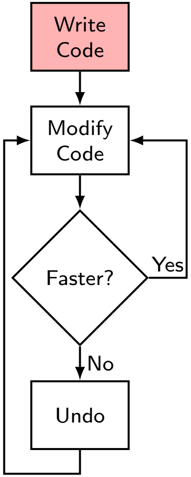
Example: $(X + Y) \cdot Z + Z^2$



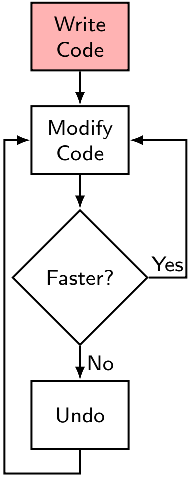
Example: $(X + Y) \cdot Z + Z^2$

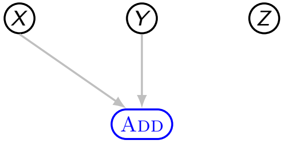


Example: $(X + Y) \cdot Z + Z^2$

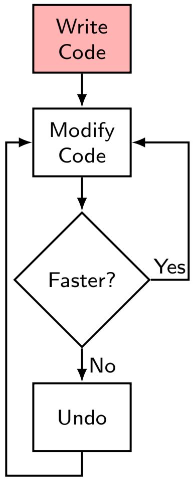


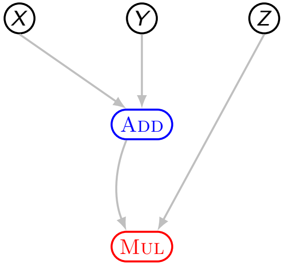
Example: $(X+Y)\cdot Z+Z^2$



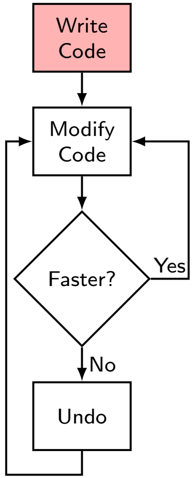


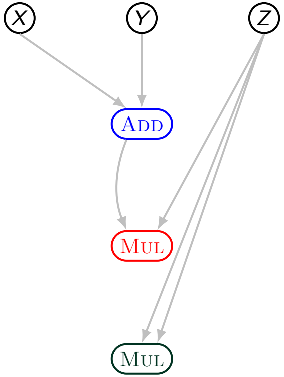
Example: $(X + Y) \cdot Z + Z^2$



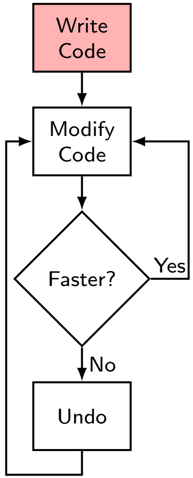


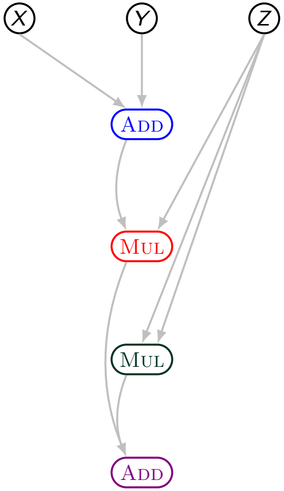
Example: $(X + Y) \cdot Z + Z^2$



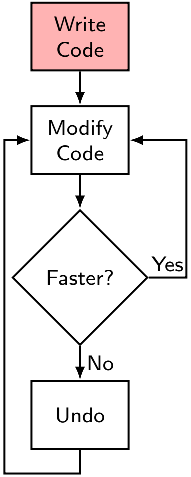


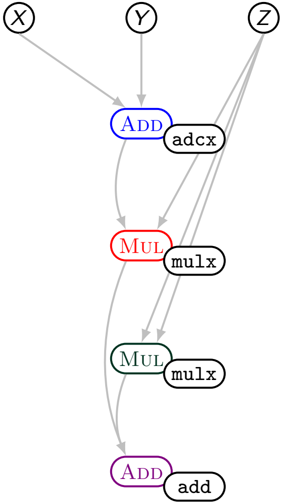
Example: $(X + Y) \cdot Z + Z^2$



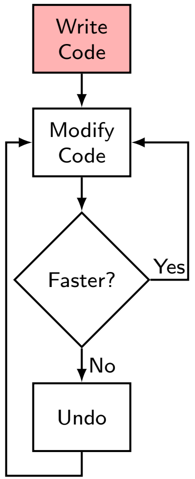


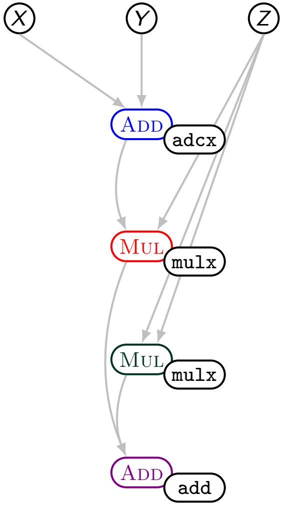
Example: $(X + Y) \cdot Z + Z^2$





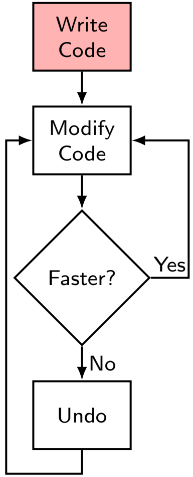
Example: $(X + Y) \cdot Z + Z^2$

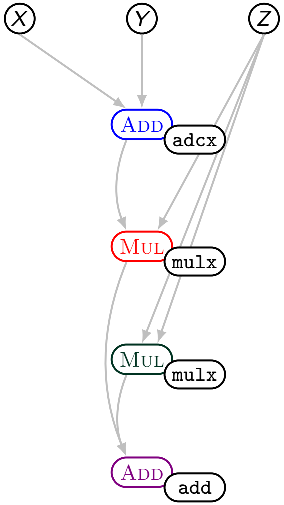




```
mov rax, [X]
clc
adcx rax, [Y]
```

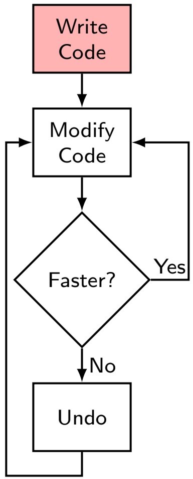
Example: $(X+Y)\cdot Z+Z^2$

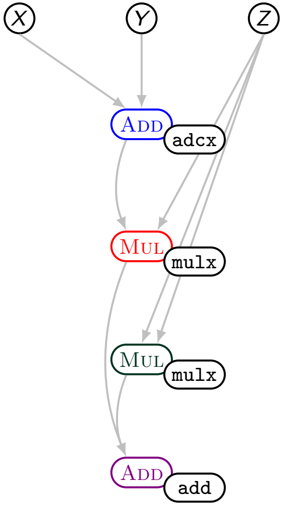




```
mov rax, [X]
clc
adcx rax, [Y]

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

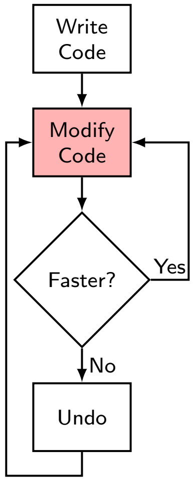
Example: $(X + Y) \cdot Z + Z^2$

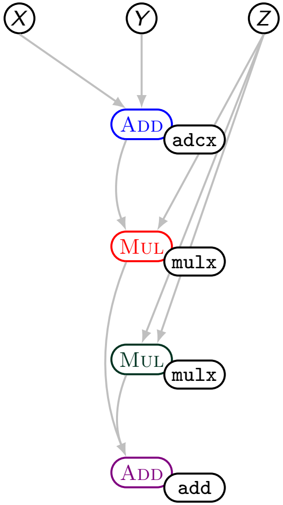




```
mov rax, [X]
clc
adcx rax, [Y]

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


mulx r10, r11, [Z]


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

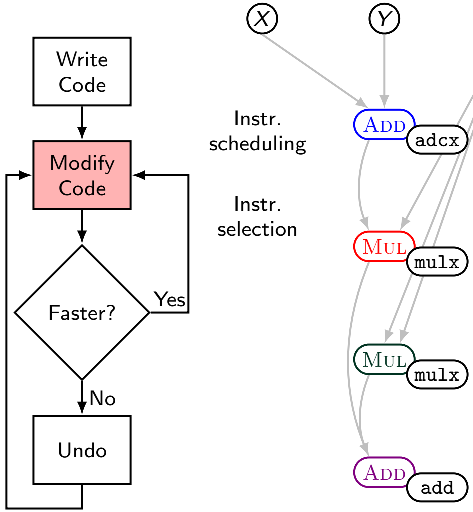
Example: $(X + Y) \cdot Z + Z^2$





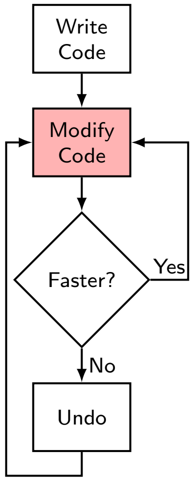
```
mov rax, [X]
clc
adcx rax, [Y]

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


mulx r10, r11, [Z]


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

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



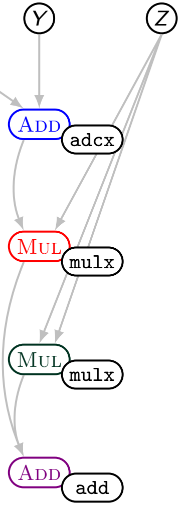
```
mov rax, [X]
clc
adcx rax, [Y]

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

mulx r10, r11, [Z]

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

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



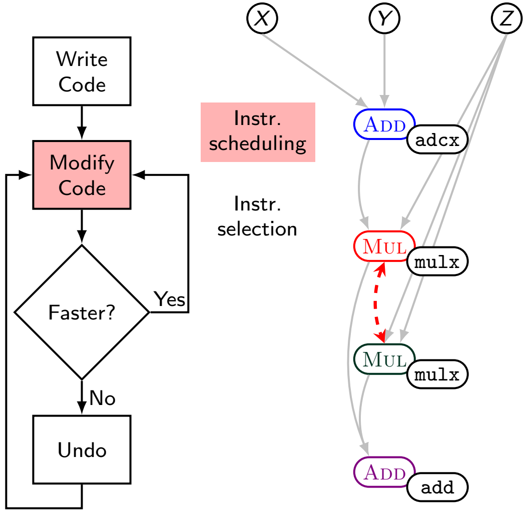
Instr. scheduling

Instr. selection



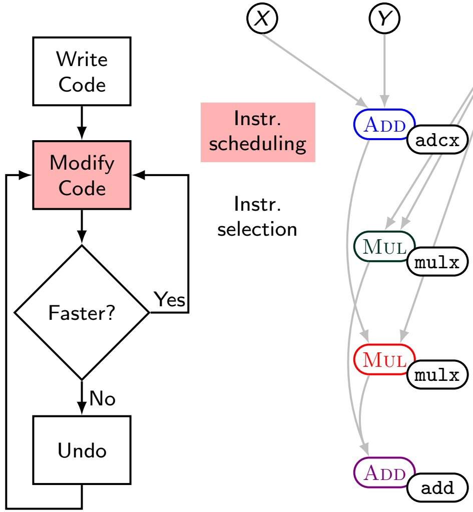
```
mov rax, [X]
clc
adcx rax, [Y]

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


mulx r10, r11, [Z]


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

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



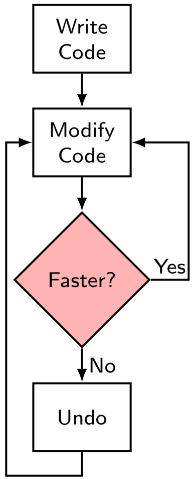
```
mov rax, [X]
clc
adcx rax, [Y]

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

mulx r10, r11, [Z]

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

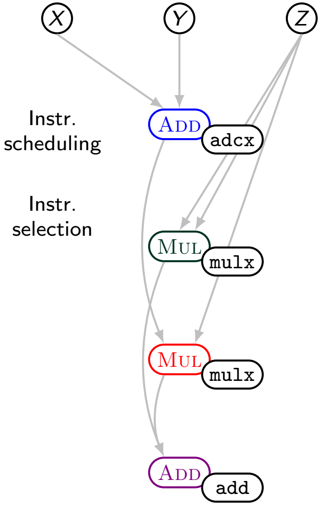
Example: $(X + Y) \cdot Z + Z^2$



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

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


mulx r8, r9, rax


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

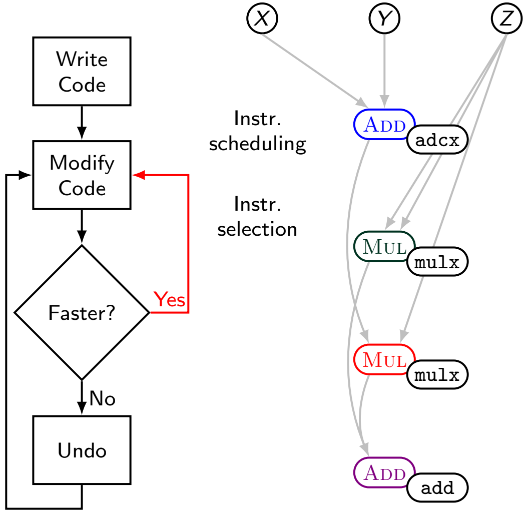
Example: $(X + Y) \cdot Z + Z^2$





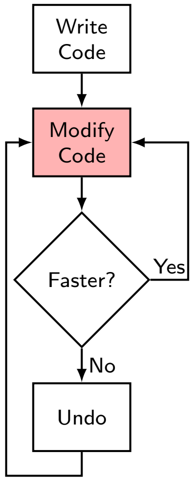
```
mov rax, [X]
clc
adcx rax, [Y]

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

mulx r8, r9, rax

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

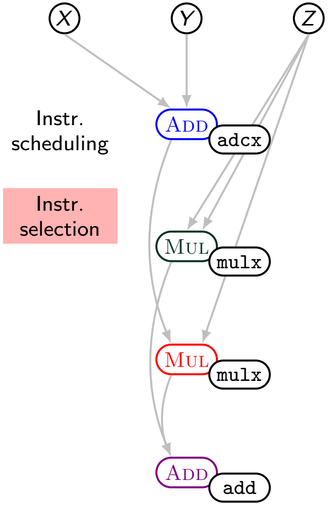
Example: $(X+Y) \cdot Z + Z^2$



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

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

mulx r8, r9, rax

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

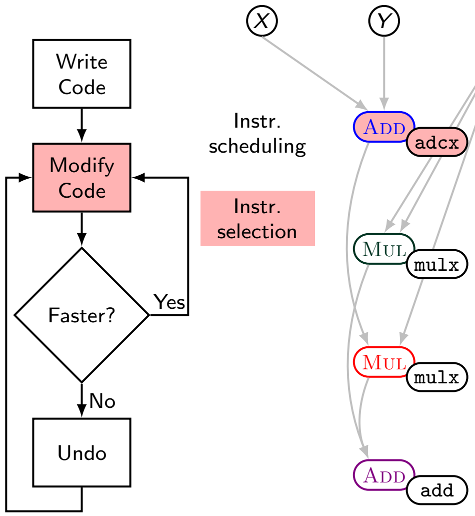
Example: $(X + Y) \cdot Z + Z^2$





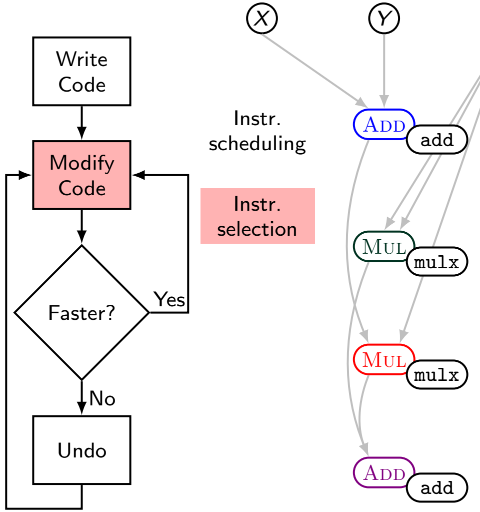
```
mov rax, [X]
clc
adcx rax, [Y]

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

mulx r8, r9, rax

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

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



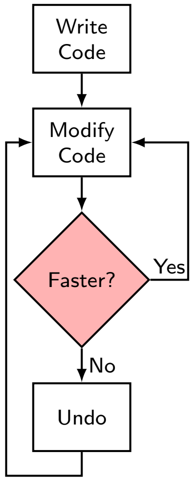
```
mov rax, [X]
clc
adcx rax, [Y]

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


mulx r8, r9, rax


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

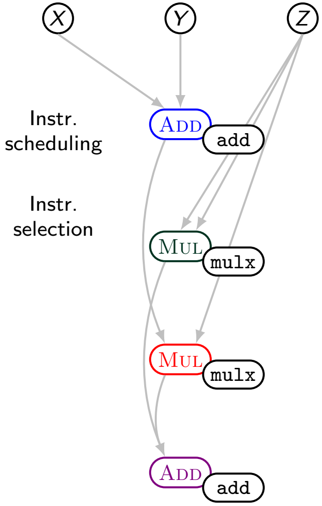
Example: $(X + Y) \cdot Z + Z^2$



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

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

mulx r8, r9, rax

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

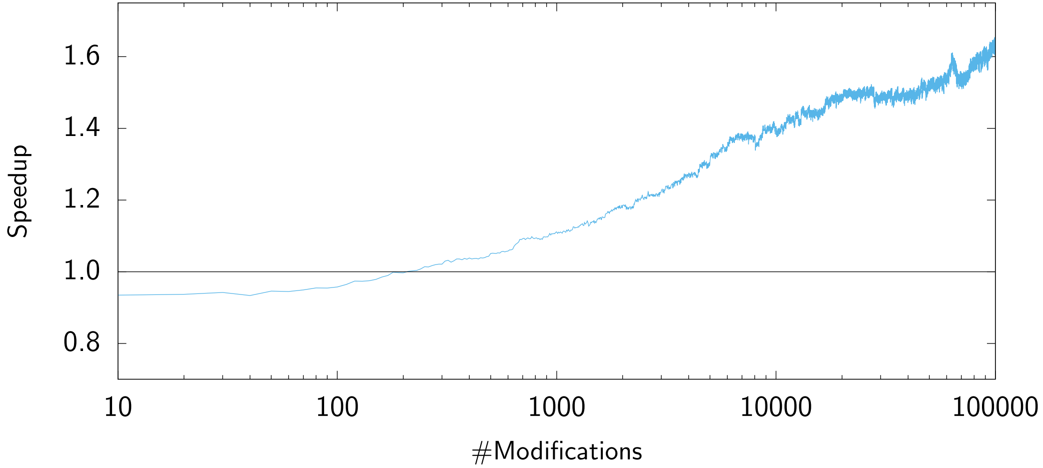
Example: $(X + Y) \cdot Z + Z^2$





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

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

mulx r8, r9, rax

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

Performance



Fiat Cryptography

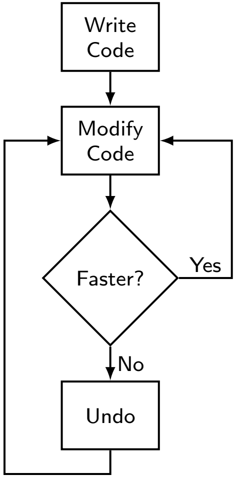
Fiat Cryptography
[Erbsen et al. 2019, IEEE S&P]

Fiat Cryptography

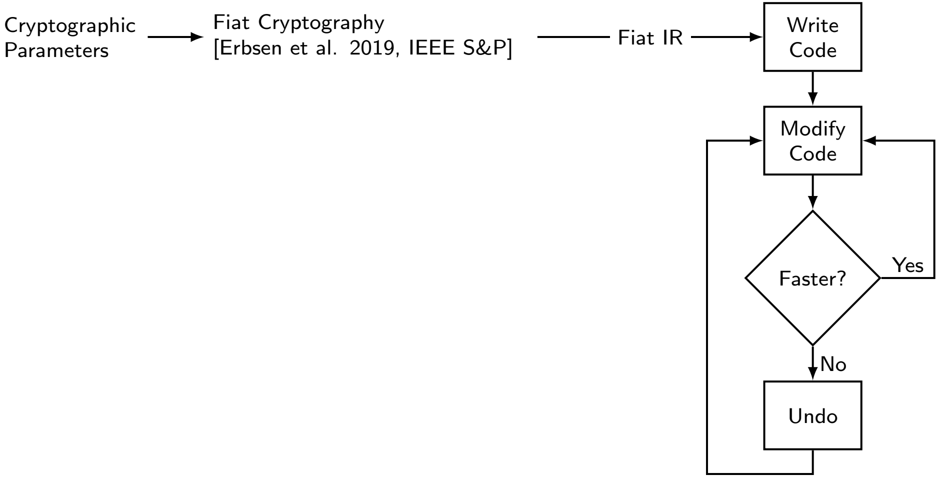
$$\text{Cryptographic Parameters} \longrightarrow \text{Fiat Cryptography [Erbsen et al. 2019, IEEE S\&P]}$$

Fiat Cryptography

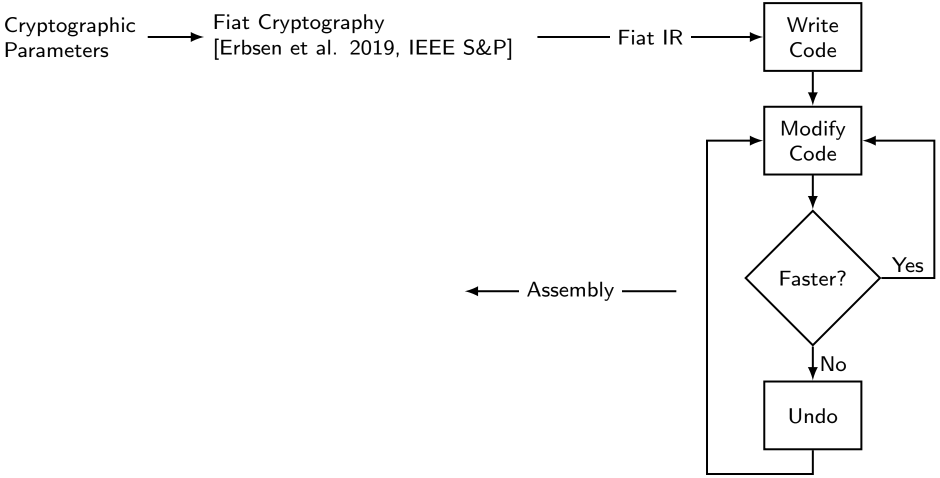
Cryptographic Parameters ⟶ Fiat Cryptography [Erbsen et al. 2019, IEEE S&P]



Fiat Cryptography



Fiat Cryptography



Performance: Field Arithmetic

Geometric Mean (10 μ-Architectures)

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

Performance: Field Arithmetic

Geometric Mean (10 μ-Architectures)

	Multiply		Square	
	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				

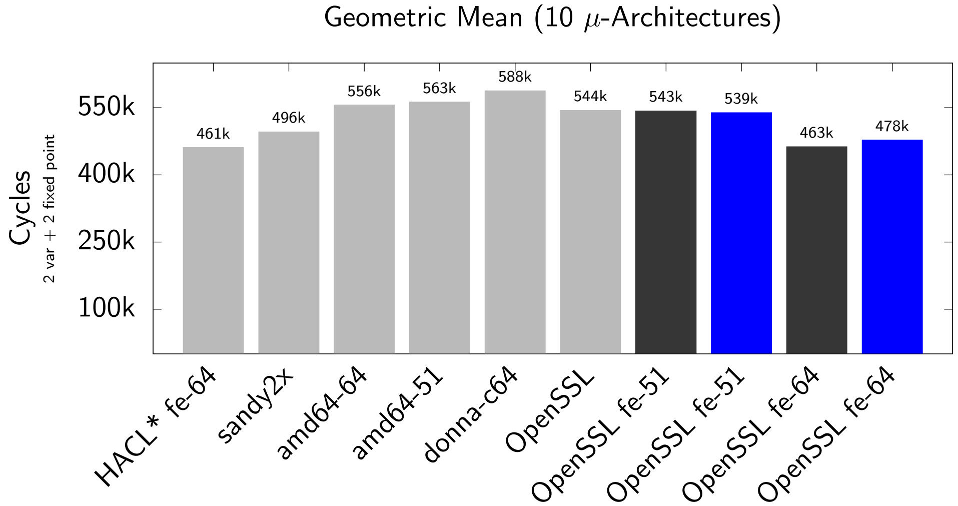
Performance: Field Arithmetic

Geometric Mean (10 μ-Architectures)

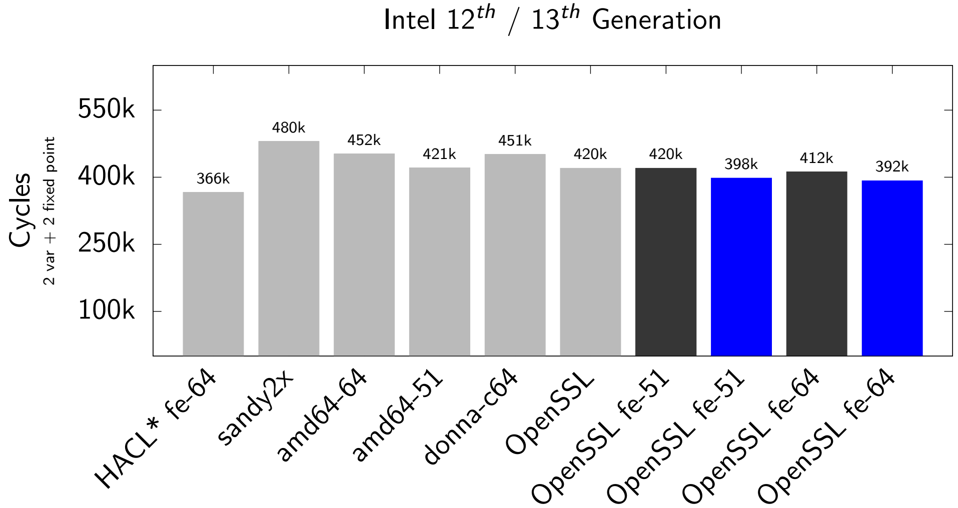
	Multiply		Square	
	Clang	GCC	Clang	GCC
Curve25519	1.19	1.14	1.14	1.18
P-224	1.31	1.87	1.24	1.84
P-256	1.27	1.79	1.30	1.85
P-384	1.12	1.66	1.08	1.60
SIKEp434	1.30	1.70	1.29	1.83
Curve448	1.02	0.95	1.00	0.99
P-521	1.20	1.06	1.25	1.11
Poly1305	1.10	1.15	1.09	1.16
secp256k1	1.34	1.73	1.32	1.74

Performance: Scalar Multiplication

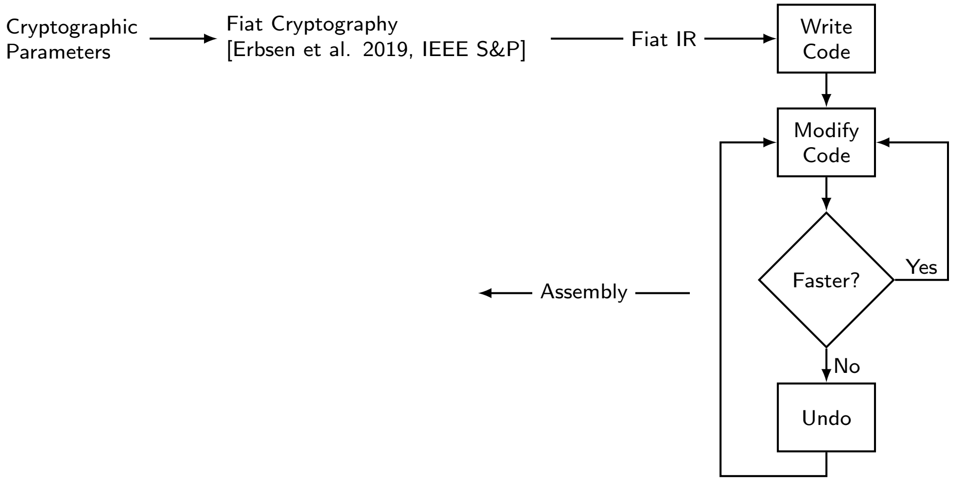
Performance: Scalar Multiplication



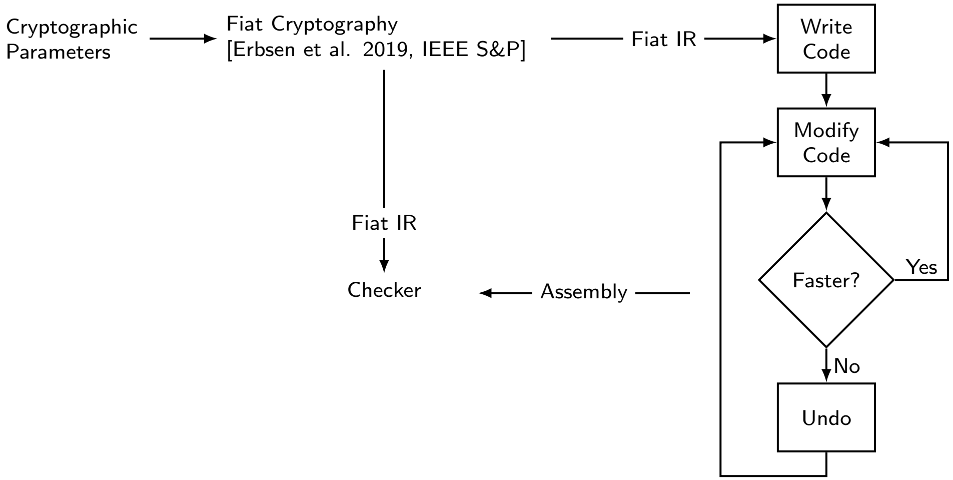
Performance: Scalar Multiplication



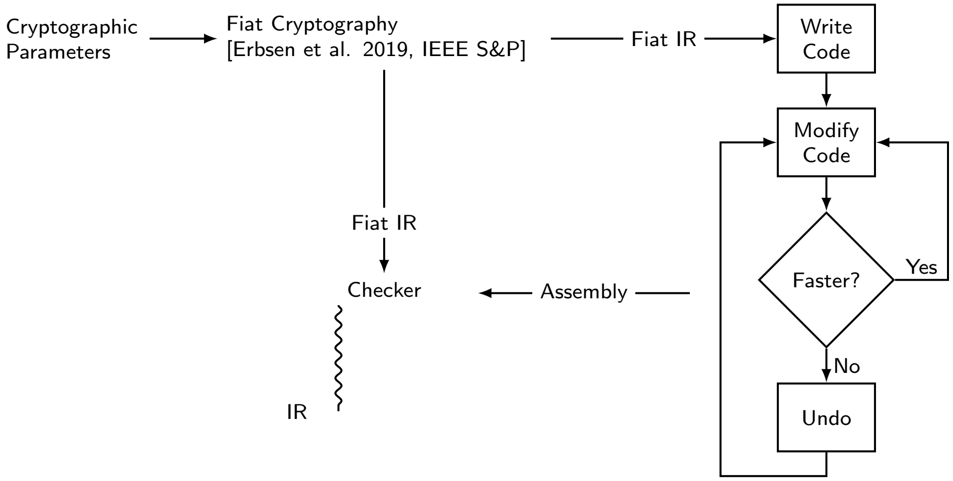
Correctness



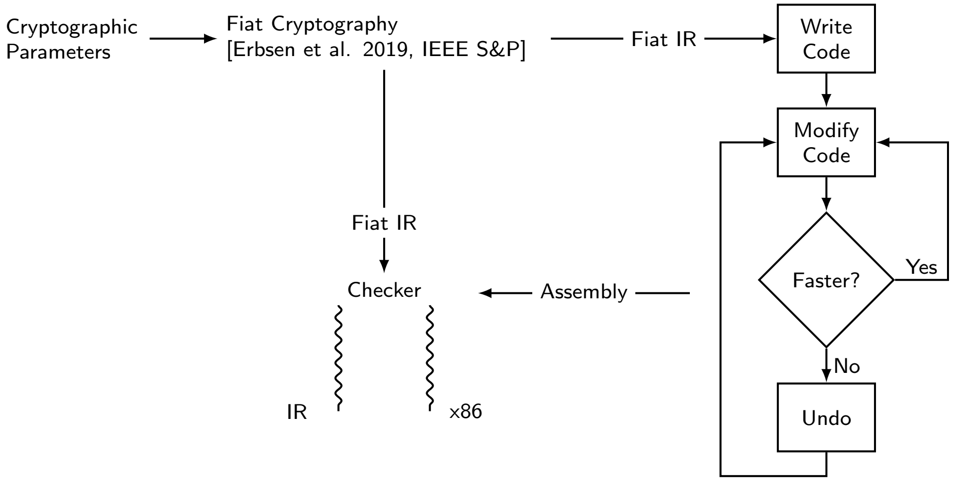
Correctness



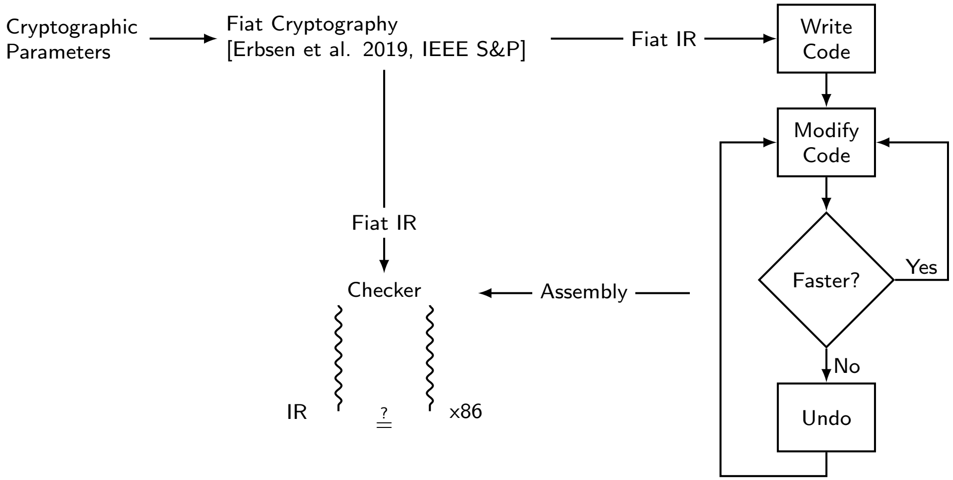
Correctness



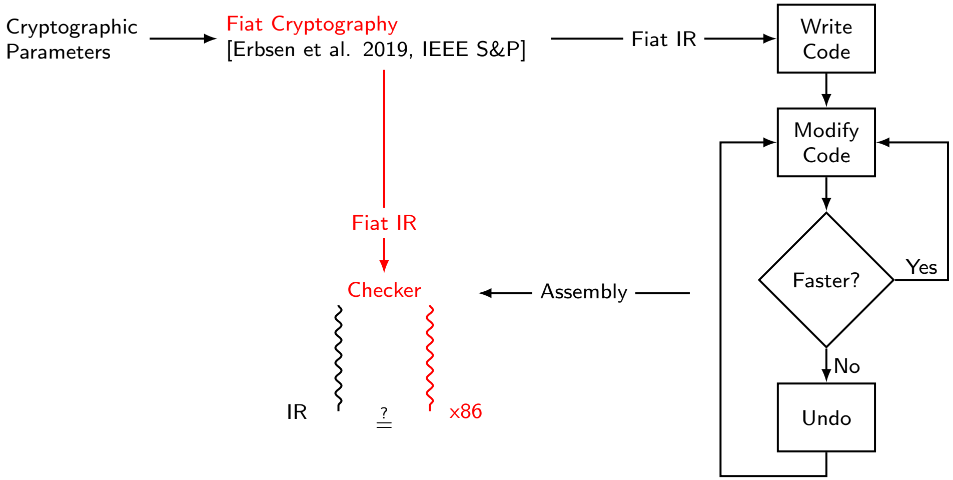
Correctness



Correctness



Correctness



Summary

Summary

Compilation of cryptographic code
$\Longrightarrow$ Search

Summary

Compilation of cryptographic code
$\Longrightarrow$ Search

Random Local Search $+$ Runtime

Summary

Compilation of cryptographic code
$\implies$ Search

Random Local Search $+$ Runtime

Proven-correct assembly for field arithmetic by Fiat Cryptography

Summary

Compilation of cryptographic code
$\Longrightarrow$ Search

Random Local Search + Runtime

Proven-correct assembly for field arithmetic by Fiat Cryptography with on-par performance to hand-optimized assembly.

Summary

Compilation of cryptographic code
$\implies$ Search

Random Local Search + Runtime

Proven-correct assembly for field arithmetic by Fiat Cryptography with on-par performance to hand-optimized assembly.

Curve25519 assembly in BoringSSL.
PR #1329 at bitcoin-core/secp256k1.

Summary

Compilation of cryptographic code
$\Longrightarrow$ Search

Random Local Search + Runtime

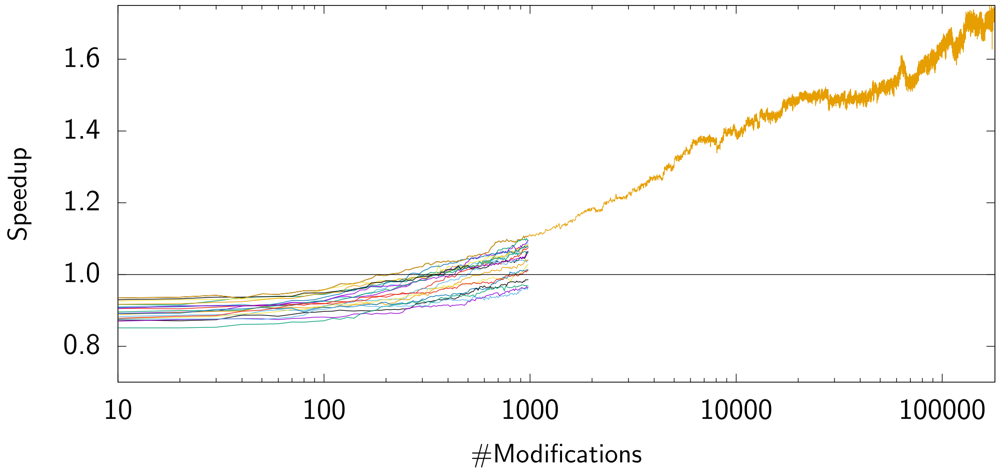
Proven-correct assembly for field arithmetic by Fiat Cryptography with on-par performance to hand-optimized assembly.

Curve25519 assembly in BoringSSL.
PR #1329 at bitcoin-core/secp256k1.

GitHub Project

`https://0xade1a1de.github.io/CryptOpt`

Bet and Run


1.6
1.4
1.2
1.0
0.8
Speedup
10
100
1000
10000
100000
#Modifications