A Study on the Preservation on Cryptographic Constant-Time Security in the CompCert Compiler

Alix Trieu[†]

FOSAD 2018, Bertinoro

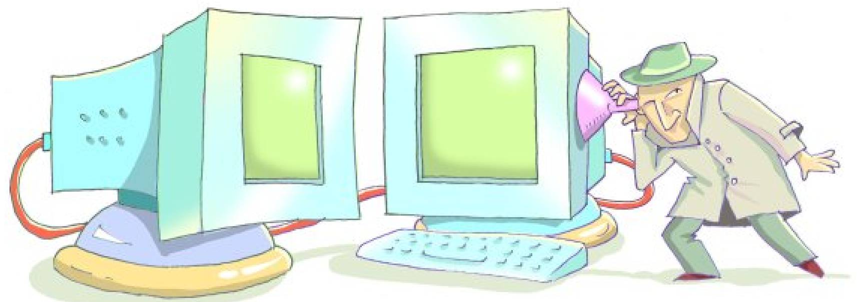
[†]Univ Rennes, Inria, CNRS, IRISA

August 29th, 2018

Side-channels

Crypto algorithms are designed to be *mathematically* sound, but their implementations run in the *physical* world.

Their execution affects the world:



Side-channels

Crypto algorithms are designed to be *mathematically* sound, but their implementations run in the *physical* world.

Their execution affects the world:

- uses power
- makes noise
- takes time to compute

Example of timing side-channel:

```
if (secret) {
  foo();
} else {
  bar();
}
```

Leak information on secret if `foo` and `bar` have different execution time.

Cache Timing Attacks

- Program is run and table is put in cache

table[0]...table[7]
table[8]...table[15]
table[16]...table[23]
table[24]...table[31]
table[32]...table[39]
table[40]...table[47]
table[48]...table[55]
table[56]...table[63]

Cache Timing Attacks

- Program is run and table is put in cache
- The attacker replaces some cache lines

Cache Timing Attacks

- Program is run and table is put in cache
- The attacker replaces some cache lines
- Program continues and loads from table again

table[0]...table[7]
?
table[16]...table[23]
table[24]...table[31]
?
?
table[48]...table[55]
table[56]...table[63]

Cache Timing Attacks

- Program is run and table is put in cache
- The attacker replaces some cache lines
- Program continues and loads from table again
- Attacker loads his data:
  - Fast: program did not load from this line

table[0]...table[7]
?
table[16]...table[23]
table[24]...table[31]
?
Attacker's data
table[48]...table[55]
table[56]...table[63]

Cache Timing Attacks

- Program is run and table is put in cache
- The attacker replaces some cache lines
- Program continues and loads from table again
- Attacker loads his data:
  - Fast: program did not load from this line
  - Slow: program did load from this line

table[0]...table[7]
?
table[16]...table[23]
table[24]...table[31]
?
table[40]...table[47]
table[48]...table[55]
table[56]...table[63]

Constant-Time Programming

"It is important that the pattern of memory accesses and jumps not depend on the values of any of the bits of k."
RFC7748: Elliptic Curves for Security, Section 5.1. Side-Channel Considerations

Constant-time programs ensure that

- branchings are not secret dependent
- memory accesses are not secret dependent

Challenge

Crypto libraries (OpenSSL, NaCl, libsodium, MbedTLS, …) are mostly written in C.

Can we be sure that compilers preserve constant-time security ?

Challenge

Crypto libraries (OpenSSL, NaCl, libsodium, MbedTLS, …) are mostly written in C.

Can we be sure that compilers preserve constant-time security ?

Example

```c
// kind of things found in crypto libraries
// constant-time with regards to b
unsigned ct_select
      (unsigned x, unsigned y, bool b)
{ return x ^ ((y ^ x) & (-(unsigned) b)); }
```

```c
// not constant-time
unsigned not_ct_select
      (unsigned x, unsigned y, bool b)
{
  if (b) {
    return y;
  } else {
    return x;
  }
}
```

```asm
; not constant-time
  mov al, byte ptr [esp + 12]
  mov ecx, dword ptr [esp + 4]
  test al, al
  jne .LBB0_1
  xor eax, eax
  xor eax, ecx
  ret
.LBB0_1:
  mov eax, dword ptr [esp + 8]
  xor eax, ecx
  xor eax, ecx
  ret
```

What does a compiler guarantee ?

Observable behaviors

- Terminates($t_1 t_2 t_3 \cdots t_n$)
- GoesWrong($t_1 t_2 t_3 \cdots t_n$)
- Silent($t_1 t_2 t_3 \cdots t_n$)
- Diverges($t_1 t_2 t_3 \cdots$)

Semantic preservation (Compiler correctness)

$$\forall P, \forall t, P\downarrow \rightsquigarrow t \implies P \rightsquigarrow t$$

What does a compiler guarantee ?

Observable behaviors

- Terminates($t_1 t_2 t_3 \cdots t_n$)
- GoesWrong($t_1 t_2 t_3 \cdots t_n$)
- Silent($t_1 t_2 t_3 \cdots t_n$)
- Diverges($t_1 t_2 t_3 \cdots$)

Semantic preservation (Compiler correctness)

$$\forall P, \forall t, P \downarrow \rightsquigarrow t \implies P \rightsquigarrow t$$

What it doesn't talk about

- side channels
- security properties
- timing, etc

What does a compiler guarantee ?

Observable behaviors

- Terminates($t_1 t_2 t_3 \cdots t_n$)
- GoesWrong($t_1 t_2 t_3 \cdots t_n$)
- Silent($t_1 t_2 t_3 \cdots t_n$)
- Diverges($t_1 t_2 t_3 \cdots$)

Semantic preservation (Compiler correctness)

$$\forall P, \forall t, P \downarrow \rightsquigarrow t \implies P \rightsquigarrow t$$

What it doesn't talk about

- side channels
- security properties
- timing, etc

How to solve this ?

Outline of this talk

- Primer on compiler correctness *à la* CompCert
- How to adapt the previous proofs to preservation of constant-time security
- Intuition on why CompCert preserves constant-time

Semantics of a language

A semantics is represented by:

- a set of states,
- a set of transitions between states,
- a predicates to indicate initial and final states.

$$s \xrightarrow{\quad t \quad} s'$$

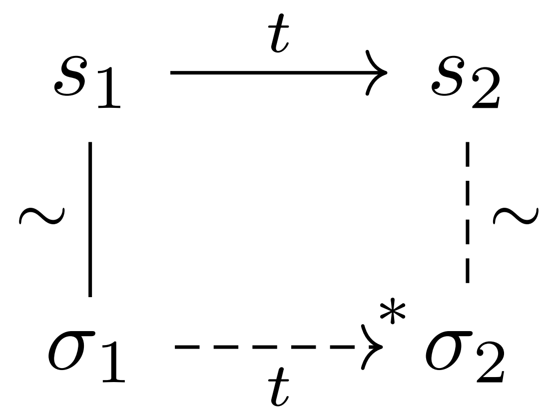
Observable behavior is (in)finite trace $t_1 \cdot t_2 \cdot t_3 \cdots$

$$s_1 \xrightarrow{\quad t_1 \quad} s_2 \xrightarrow{\quad t_2 \quad} s_3 \dashrightarrow \cdots$$

Preservation of observable behavior

Two programs have the same observable behavior if

- their initial states are related by a relation $\sim$
- their related final states have the same return value
- the diagram is satisfied



"Leaky" semantics[†]

Semantics is instrumented with a "leakage"

$$s \xrightarrow{\quad l \quad} s'$$

$$\langle \sigma, \texttt{if}\ e\ \texttt{then}\ s_1\ \texttt{else}\ s_2 \rangle \xrightarrow{\sigma(e)} s'$$

$$\langle \sigma, a[e] \leftarrow e' \rangle \xrightarrow{\sigma(e)} s'$$

A leakage can also be empty, ε.

[†] Verifying Constant-Time Implementations, Almeida, Barbosa, Barthe, Dupressoir, Emmi, USENIX'16

Constant-time security

A program is said to be constant-time if for any two of its executions that initially agree on public values, their leakages are the same.

$$s_1$$

$$s'_1$$

Intuitively, it means that the leaks do not depend on secret.

Constant-time security

A program is said to be constant-time if for any two of its executions that initially agree on public values, their leakages are the same.

$$s_1 \xrightarrow{l_1} s_2$$

$$s'_1 \xrightarrow{l_1} s'_2$$

Intuitively, it means that the leaks do not depend on secret.

Constant-time security

A program is said to be constant-time if for any two of its executions that initially agree on public values, their leakages are the same.

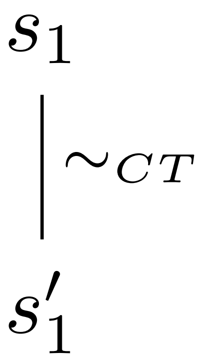
$$s_1 \xrightarrow{l_1} s_2 \xrightarrow{l_2} s_3 \dashrightarrow \cdots$$

$$s'_1 \xrightarrow{l_1} s'_2 \xrightarrow{l_2} s'_3 \dashrightarrow \cdots$$

Intuitively, it means that the leaks do not depend on secret.

Constant-time security (bis)

An equivalent way to state it is to use a simulation diagram.



Constant-time security (bis)

An equivalent way to state it is to use a simulation diagram.

$$\begin{array}{ccc} s_1 & \xrightarrow{l_1} & s_2 \\ \Big| \sim_{CT} & & \Big| \sim_{CT} \\ s'_1 & \xrightarrow{l_1} & s'_2 \end{array}$$

Constant-time security (bis)

An equivalent way to state it is to use a simulation diagram.

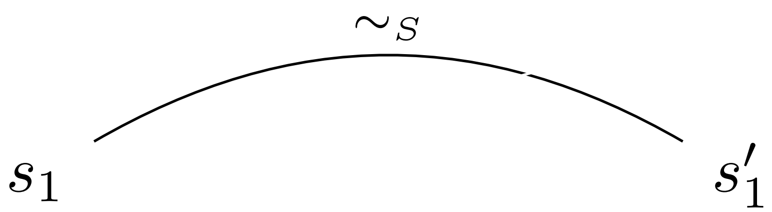
$$
\begin{array}{ccccccc}
s_1 & \xrightarrow{l_1} & s_2 & \xrightarrow{l_2} & s_3 & \dashrightarrow & \cdots \\
\big| \sim_{CT} & & \big| \sim_{CT} & & \big| \sim_{CT} & & \\
s'_1 & \xrightarrow{l_1} & s'_2 & \xrightarrow{l_2} & s'_3 & \dashrightarrow & \cdots
\end{array}
$$

Preservation of constant-time security

Constant-time security talks about two different executions of a program.

Preservation of constant-time security

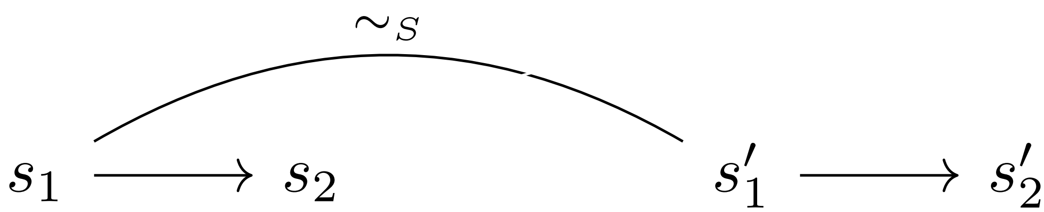
Constant-time security talks about two different executions of a program.



We have two states related by a constant-time simulation relation.

Preservation of constant-time security

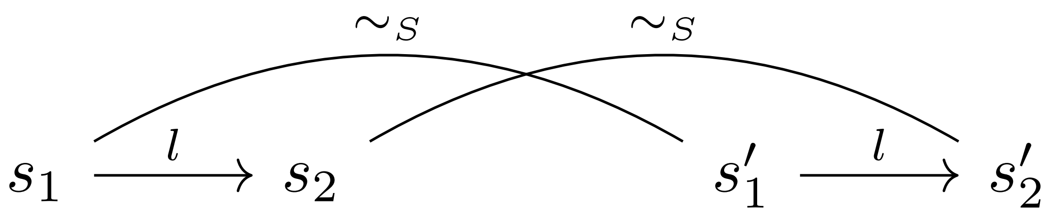
Constant-time security talks about two different executions of a program.



Suppose that both executions now advance a step.

Preservation of constant-time security

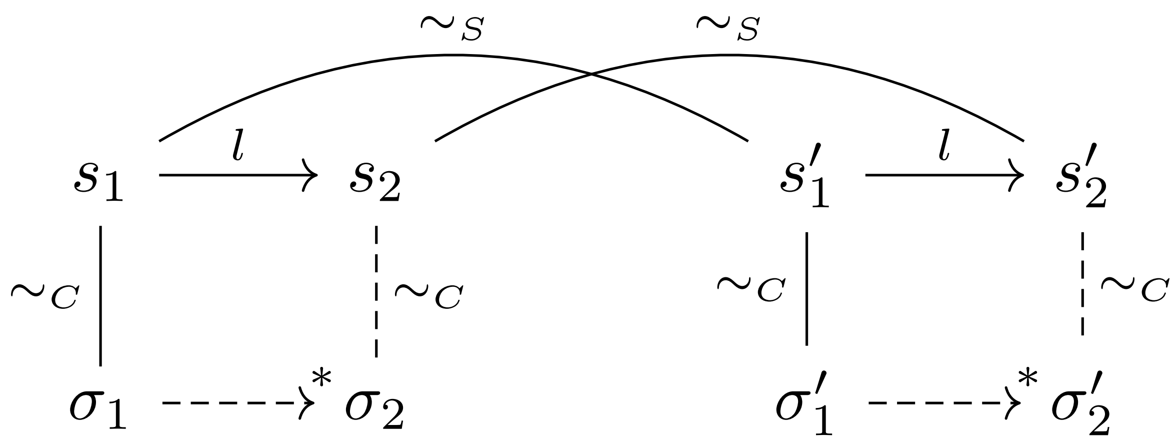
Constant-time security talks about two different executions of a program.



Constant-time simulation tells us that they both leak the same leakage and the reached states are related.

Preservation of constant-time security

Constant-time security talks about two different executions of a program.



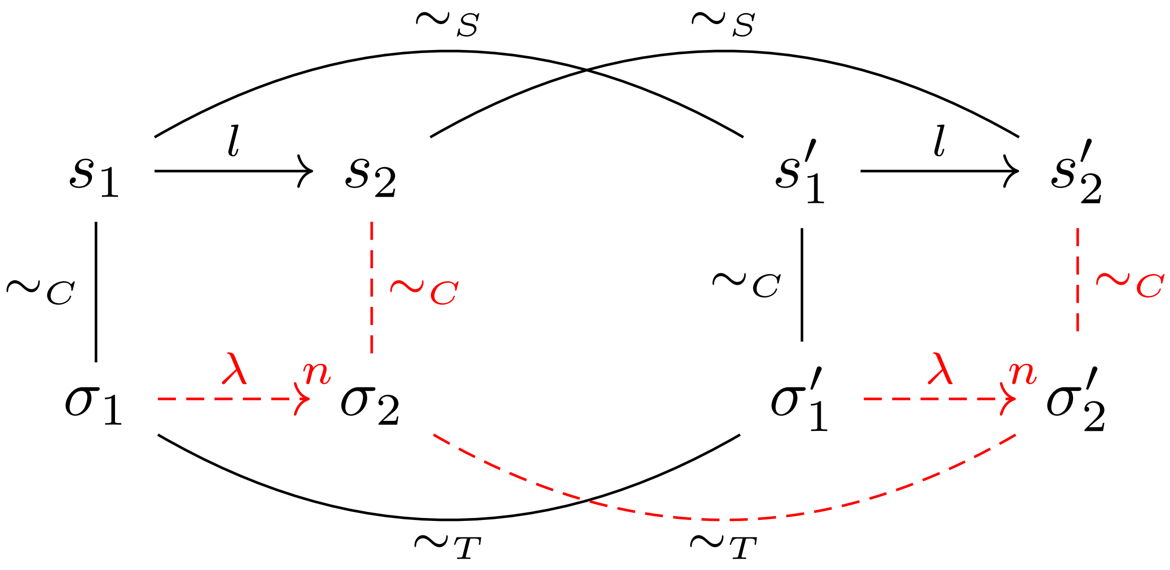
Similarly, we now use the diagram for preservation of traces.

However, it talks about traces, not leakages.

And says nothing about the number of steps.

Preservation of constant-time security

Constant-time security talks about two different executions of a program.

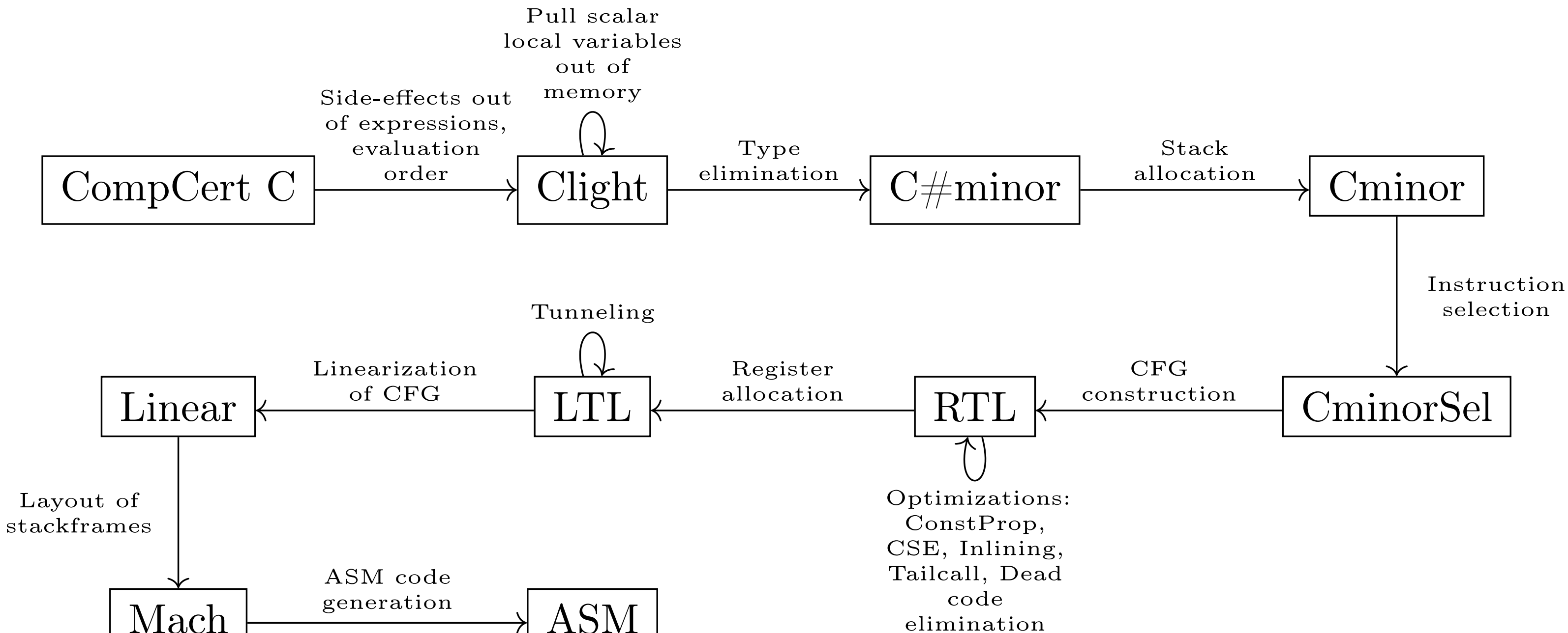


Assuming plain lines, prove dashed lines in the diagram.

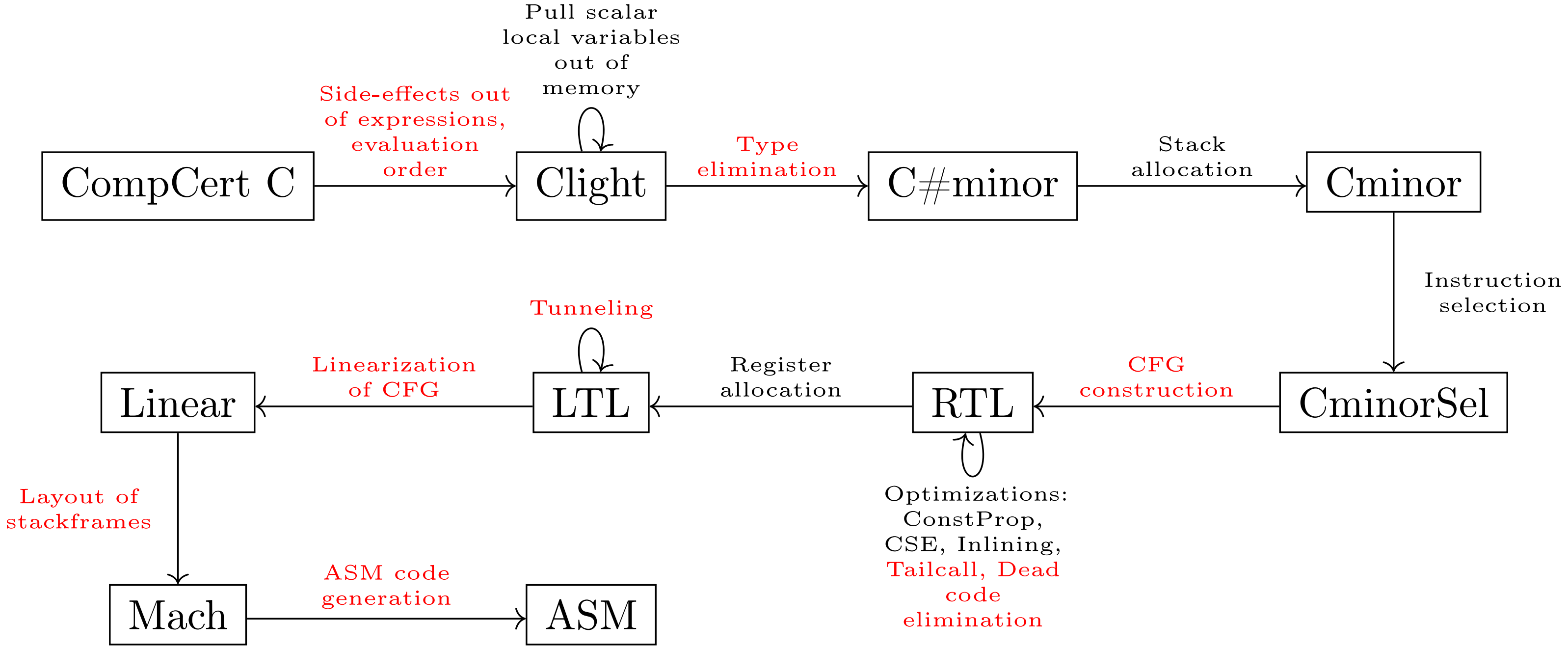
Prove that if there are two equal leaks at source level, then the leaks are the same at target level.

Paper proof that it implies preservation of constant-time security.

CompCert vs Constant-time security ?

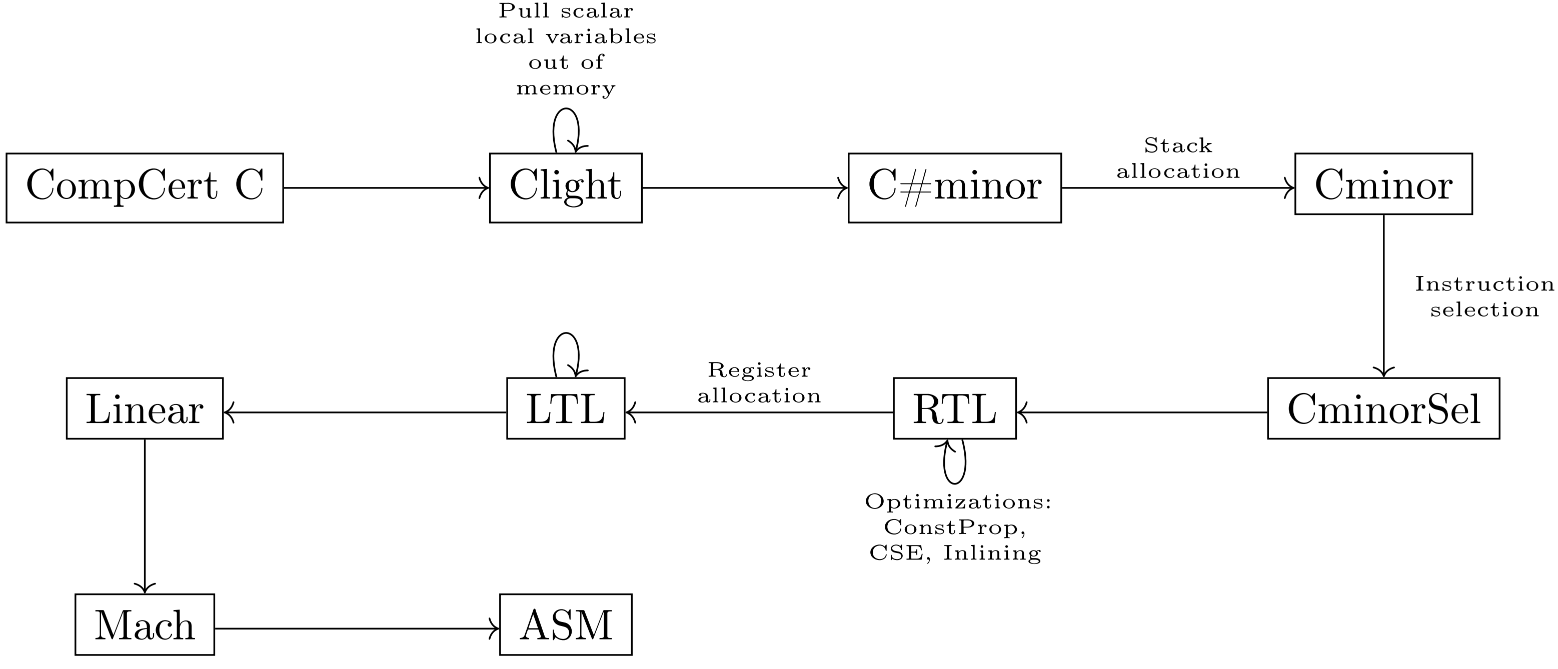


Benign transformations



- Only changing representation, does not introduce control-flow or memory access modifications
- Expliciting execution
- Dead code elimination: remove code that's not executed

What's left

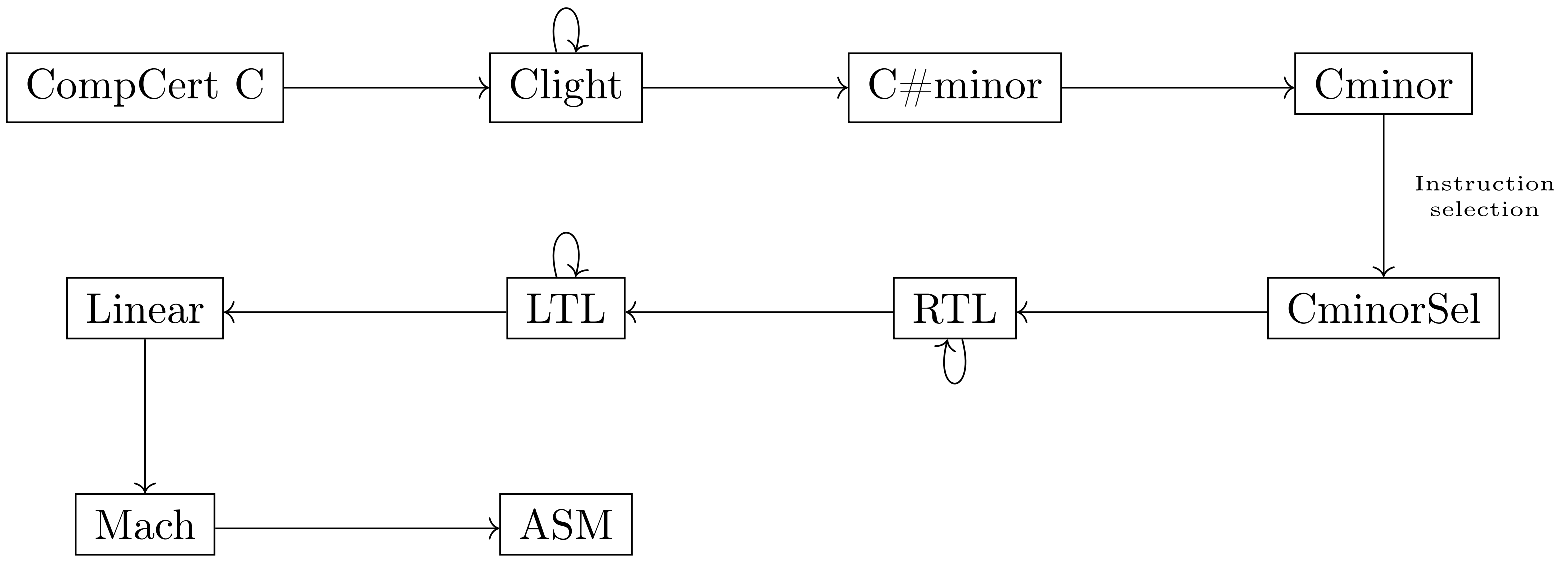


Pull scalar local variables out of memory
CompCert C
Clight
C#minor
Stack allocation
Cminor
Instruction selection
CminorSel
RTL
Optimizations: ConstProp, CSE, Inlining
Register allocation
LTL
Linear
Mach
ASM

Transformations that touch memory

	Remove memory accesses	Shift memory accesses at constant offset	Add memory accesses at constant offset
Pulling scalar local variables out of memory	✓		
Stack Allocation		✓	
CSE	✓		
ConstProp	✓		
Inlining		✓	
Register Allocation			✓

What's left bis



Instruction Selection

- It replaces recognized patterns with platform specific operators or builtins (e.g., `x % 4` replaced by `x & 3`, or long long multiplication on 32 bits architectures).
- Need to be careful that handwritten assembly builtin functions are constant-time.
- Need to verify one by one that each implementation is constant-time.

Conclusion

- Framework to prove preservation of constant-time security.
- Intuition on why CompCert preserves constant-time.
- No mechanization in Coq yet, will start in a week or two after my thesis is sent to reviewers.

Conclusion

- Framework to prove preservation of constant-time security.
- Intuition on why CompCert preserves constant-time.
- No mechanization in Coq yet, will start in a week or two after my thesis is sent to reviewers.

Thank you !