Synthesizing HW-SW Leakage Contracts for RISC-V Open-Source Processors

(to appear at DATE 2024)

Jan Reineke

Joint work with
Gideon Mohr @ Universität des Saarlandes
Marco Guarnieri @ IMDEA Software

This work has received funding from an Intel Strategic Research Alliance (ISRA) and the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme (grant agreement No 101020415)
The Need for New HW/SW Contracts
Instruction Set Architectures (ISAs): Benefits

High-level language ➔ Instruction set architecture (ISA) ➔ Microarchitecture
Instruction Set Architectures (ISAs): Benefits

High-level language

Instruction set architecture (ISA)

Microarchitecture

Can program independently of microarchitecture
Instruction Set Architectures (ISAs): Benefits

High-level language

Instruction set architecture (ISA)

Microarchitecture

Can program independently of microarchitecture

Can implement arbitrary optimizations as long as ISA semantics are obeyed
Inadequacy of ISAs: Side Channels

High-level language

Instruction set architecture (ISA)

Microarchitecture
Inadequacy of ISAs: Side Channels

- High-level language
- Instruction set architecture (ISA)
- Microarchitecture

No guarantees about side channels
Inadequacy of ISAs: Side Channels

High-level language

>-

Instruction set architecture (ISA)

No guarantees about side channels

Can implement arbitrary insecure optimizations as long as ISA is implemented correctly

Microarchitecture
Inadequacy of ISAs: Side Channels

High-level language

Instruction set architecture (ISA)

Microarchitecture

No guarantees about side channels

Can implement arbitrary insecure optimizations as long as ISA is implemented correctly
Inadequacy of ISAs: Side Channels

Impossible to program securely cryptographic algorithms? sandboxing untrusted code?

No guarantees about side channels

Can implement arbitrary insecure optimizations as long as ISA is implemented correctly

Instruction set architecture (ISA)

High-level language

Microarchitecture
A Way Forward: HW/SW Leakage Contracts

HW/SW Leakage contract = ISA + Leakage specification
A Way Forward: HW/SW Leakage Contracts

Can program **securely** on top of contract **independently** of microarchitecture

**HW/SW Leakage contract = ISA + Leakage specification**
**A Way Forward: HW/SW Leakage Contracts**

Can program **securely** on top of contract **independently** of microarchitecture.

**HW/SW Leakage contract = ISA + Leakage specification**

Can implement **arbitrary insecure optimizations** as long as contract is obeyed.
A Way Forward: HW/SW Leakage Contracts

Can program securely on top of contract independently of microarchitecture

HW/SW Leakage contract = ISA + Leakage specification

Captures possible leakage at ISA level

Can implement arbitrary insecure optimizations as long as contract is obeyed
Our prior work in this context

M. Guarnieri, B. Köpf, J. Reineke, and P. Vila
Hardware-Software Contracts for Secure Speculation
S&P (Oakland) 2021

Specification and Verification of Side-channel Security for Open-source Processors via Leakage Contracts
CCS (2023)
Synthesizing Leakage Contracts

Can program securely on top of contract independently of microarchitecture

HW/SW leakage contract = ISA + Leakage specification

Captures possible leakage at ISA level
Synthesizing Leakage Contracts

Can program securely on top of contract independently of microarchitecture

HW/SW leakage contract = ISA + Leakage specification

Can implement arbitrary insecure optimizations as long as contract is obeyed

Captures possible leakage at ISA level
Synthesizing Leakage Contracts

Can program securely on top of contract independently of microarchitecture

HW/SW leakage contract = ISA + Leakage specification

Captures possible leakage at ISA level

Can implement arbitrary insecure optimizations as long as contract is obeyed

RISC-V Open-Source Cores

Ibex

CVA6
Synthesizing Leakage Contracts

Can program **securely** on top of contract **independently** of microarchitecture

\[
\text{HW/SW leakage contract} = \text{ISA} + \text{Leakage specification}
\]

**Captures possible leakage at ISA level**

RISC-V Open-Source Cores

- Ibex
- CVA6
Synthesizing Leakage Contracts

Can program **securely** on top of contract **independently** of microarchitecture

**HW/SW leakage contract = ISA + Leakage specification**

Captures possible leakage at ISA level

**Contract Synthesis**

RISC-V Open-Source Cores

- Ibex
- CVA6
1. Contracts and Contract Templates
2. Synthesis Goals and Methodology
3. Some Experimental Results
Outline

1. Contracts and Contract Templates
2. Synthesis Goals and Methodology
3. Some Experimental Results
An Example Contract

“For every multiplication the second operand leaks and for every memory access the memory address leaks.”
An Example Contract

“For every multiplication the second operand leaks and for every memory access the memory address leaks.”
An Example Contract

“For every multiplication the second operand leaks and for every memory access the memory address leaks.”
An Example Contract

“For every multiplication the second operand leaks and for every memory access the memory address leaks.”

Composition of Contract Atoms

Condition on ISA State

Leakage of Part of ISA State
Contract Atoms -
What could possibly leak (non-speculatively)?

\[
\text{mul } a0, t0, t1 \quad a0 \leftarrow t0 \times t1
\]
Contract Atoms - What could possibly leak (non-speculatively)?

```
mul a0, t0, t1  # a0 ← t0 * t1
```

Leakage sources:
- Instruction type
Contract Atoms - What could possibly leak (non-speculatively)?

\[
mul \ a0, \ t0, \ t1 \\
\]

\[
a0 \leftarrow t0 * t1
\]

Leakage sources:
- Instruction type
- Register names
- Register values
Contract Atoms - What could possibly leak (non-speculatively)?

\[ \text{mul } a0, t0, t1 \quad a0 \leftarrow t0 \times t1 \]

\[ \text{lw } a1, -4(t2) \quad a1 \leftarrow \text{mem}[t2 + (-4)] \]

Leakage sources:
- Instruction type
- Register names
- Register values
Contract Atoms - What could possibly leak (non-speculatively)?

\begin{align*}
\text{mul} & \quad a0, t0, t1 \quad a0 \leftarrow t0 \times t1 \\
\text{lw} & \quad a1, [\text{-}4(t2)] \quad a1 \leftarrow \text{mem}[t2 + (\text{-}4)]
\end{align*}

Leakage sources:
- Instruction type
- Register names
- Register values
- Immediate values
Contract Atoms - What could possibly leak (non-speculatively)?

```
mul a0, t0, t1  a0 ← t0 * t1
lw  a1, -4(t2)  a1 ← mem[t2 + (-4)]
```

Leakage sources:
- Instruction type
- Register names
- Register values
- Immediate values
- Memory addresses
- Memory values

Memory addresses and values
Contract Atoms -
What could possibly leak (non-speculatively)?

\[ \text{mul a0, t0, t1} \quad \text{a0} \leftarrow \text{t0} \ast \text{t1} \]

\[ \text{lw a1, -4(t2)} \quad \text{a1} \leftarrow \text{mem[t2 + (-4)]} \]

Leakage sources:
- Instruction type
- Register names
- Register values
- Immediate values
- Memory addresses
- Memory values
Contract Atoms -
What could possibly leak (non-speculatively)?

Contract Atom =
Contract Atoms - What could possibly leak (non-speculatively)?

Contract Atom = Condition based on Instruction Type
Contract Atoms - What could possibly leak (non-speculatively)?

Contract Atom =  
Condition based on Instruction Type × Applicable Leakage Sources
Contract Atoms - What could possibly leak (non-speculatively)?

Contract Atom = Condition based on Instruction Type \times \text{Applicable Leakage Sources}

Several hundred contract atoms for RISC-V I+M
1. Contracts and Contract Templates

2. Synthesis Goals and Methodology

3. Some Experimental Results
Contract Synthesis: Goals
Contract Synthesis: Goals

For a given processor, find contract from template s.t.:
Contract Synthesis: Goals

For a given processor, find contract from template s.t.:

1. Processor satisfies contract
Contract Synthesis: Goals

For a given processor, find contract from template s.t.:

1. Processor satisfies contract

2. Contract is as precise as possible
Contract Synthesis: Goals

For a given processor, find contract from template s.t.:

1. Processor satisfies contract

For any pair of programs and inputs \((P_1, I_1)\) and \((P_2, I_2)\):

2. Contract is as precise as possible
Contract Synthesis: Goals

For a given processor, find contract from template s.t.:

1. Processor satisfies contract

For any pair of programs and inputs \((P_1, I_1)\) and \((P_2, I_2)\):

\[
(P_1, I_1) \neq (P_2, I_2)
\]

2. Contract is as precise as possible
Contract Synthesis: Goals

For a given processor, find contract from template s.t.:

1. Processor satisfies contract

For any pair of programs and inputs \((P_1, I_1)\) and \((P_2, I_2)\):

\[
(P_1, I_1) \neq (P_2, I_2) \implies \boxed{(P_1, I_1)} \neq \boxed{(P_2, I_2)}
\]

2. Contract is as precise as possible
Contract Synthesis: Goals

For a given processor, find contract from template s.t.:

1. Processor satisfies contract

For any pair of programs and inputs \((P_1, I_1)\) and \((P_2, I_2)\):

\[
(P_1, I_1) \neq (P_2, I_2) \implies (P_1, I_1) \neq (P_2, I_2)
\]

2. Contract is as precise as possible

As few programs and inputs \((P, I)\) as possible s.t.:
Contract Synthesis: Goals

For a given processor, find contract from template s.t.:

1. Processor satisfies contract

For any pair of programs and inputs \((P_1, I_1)\) and \((P_2, I_2)\):

\[ \text{\hat{\sigma}} (P_1, I_1) \neq \text{\hat{\sigma}} (P_2, I_2) \implies (P_1, I_1) \neq (P_2, I_2) \]

2. Contract is as precise as possible

As few programs and inputs \((P, I)\) as possible s.t.:

\[ (P_1, I_1) \neq (P_2, I_2) \]
Contract Synthesis: Goals

For a given processor, find contract from template s.t.:

1. Processor satisfies contract

For any pair of programs and inputs \((P_1, I_1)\) and \((P_2, I_2)\):

\[
(P_1, I_1) \neq (P_2, I_2) \implies (P_1, I_1) \neq (P_2, I_2)
\]

2. Contract is as precise as possible

As few programs and inputs \((P, I)\) as possible s.t.:

\[
(P_1, I_1) \neq (P_2, I_2) \land (P_1, I_1) = (P_2, I_2)
\]
Contract Synthesis: Methodology (1/2)
Contract Synthesis: Methodology (1/2)

Use test cases as proxy for contract satisfaction

Test case = Pair \((P_1, l_1)\), \((P_2, l_2)\)
Contract Synthesis: Methodology (1/2)

Use test cases as proxy for contract satisfaction

Test case = Pair \((P_1, I_1), (P_2, I_2)\)

Simulation

Contract template
Contract Synthesis: Methodology (1/2)

Use test cases as proxy for contract satisfaction

Test case = Pair \((P_1, I_1), (P_2, I_2)\)

Simulation

Attacker distinguishability

\((P_1, I_1) \neq (P_2, I_2)\)
Contract Synthesis: Methodology (1/2)

Use test cases as proxy for contract satisfaction

Test case = Pair \((P_1, I_1), (P_2, I_2)\)

Simulation

Attacker distinguishability

\((\mathcal{A}, P_1, I_1) \neq (\mathcal{A}, P_2, I_2)\)

For each atom \(X\):

Atom distinguishability

\(X_{P_1, I_1} \neq X_{P_2, I_2}\)
Simulation

For each atom $X$:

$\phi(P_1, I_1) \neq \phi(P_2, I_2)$

$X(P_1, I_1) \neq X(P_1, I_1)$
Simulation

\[ (P_1, I_1) \neq (P_2, I_2) \]

For each atom \( X \):

\[ (P, I_1) \neq (P, I_1) \]

**Integer Linear Program**

**Contract**
For each atom $X$: 

$$X(P_1, I_1) \neq X(P_2, I_2)$$

**Simulation**

**Integer Linear Program**

**False-positive test cases + Responsible Atoms**
Simulation

For each atom $X$:

$\neg (P_1, I_1) \neq (P_2, I_2)$

$\neg X (P_1, I_1) \neq X (P_1, I_1)$

False-positive test cases + Responsible Atoms

Refinement of Template

Integer Linear Program

Contract Synthesis: Methodology (2/2)
Outline

1. Contracts and Contract Templates
2. Synthesis Goals and Methodology
3. Some Experimental Results
Experimental Evaluation

Implemented using Icarus Verilog, Google OR-Tools, and RISC-V Formal Interface
Experimental Evaluation

Implemented using Icarus Verilog, Google OR-Tools, and RISC-V Formal Interface

“Ibex is a production-quality open-source 32-bit RISC-V CPU core”
Experimental Evaluation

Implemented using Icarus Verilog, Google OR-Tools, and RISC-V Formal Interface

"Ibex is a production-quality open-source 32-bit RISC-V CPU core"

"CVA6 is an application-class 6-stage RISC-V CPU capable of booting Linux"
Performance on Ibex - Sensitivity

Sensitivity = \frac{True\ Positives}{True\ Positives + False\ Negatives}

Sensitivity w.r.t. a set of 2,000,000 test cases
Performance on Ibex - Sensitivity

$Sensitivity = \frac{\text{True Positives}}{\text{True Positives} + \text{False Negatives}}$

Final sensitivity: 99.94%
Performance on Ibex - Precision

\[ \text{Precision} = \frac{\text{True Positives}}{\text{True Positives} + \text{False Positives}} \]

Precision w.r.t. a set of 2,000,000 test cases
Performance on Ibex - Precision

\[ \text{Precision} = \frac{\text{True Positives}}{\text{True Positives} + \text{False Positives}} \]

Final precision: 18.95%
Refining Contract Template - Ibex Memory Interface

Program 1

\[\text{li a0, 0x100} \quad \text{a0 ← 0x100}\]
\[\text{lw a1, 0(a0)} \quad \text{a1 ← mem[a0]}\]

Program 2

\[\text{li a0, 0x102} \quad \text{a0 ← 0x102}\]
\[\text{lw a1, 0(a0)} \quad \text{a1 ← mem[a0]}\]
Refining Contract Template - Ibex Memory Interface

Program 1

li a0, 0x100  
a0 ← 0x100
lw a1, 0(a0)  
a1 ← mem[a0]

Program 2

li a0, 0x102  
a0 ← 0x102
lw a1, 0(a0)  
a1 ← mem[a0]

Slower!
Refining Contract Template - Ibex Memory Interface

<table>
<thead>
<tr>
<th>Program 1</th>
<th>Program 2</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>li a0, 0x100</code> a0 ← 0x100</td>
<td><code>li a0, 0x102</code> a0 ← 0x102</td>
</tr>
<tr>
<td><code>lw a1, 0(a0)</code> a1 ← mem[a0]</td>
<td><code>lw a1, 0(a0)</code> a1 ← mem[a0]</td>
</tr>
</tbody>
</table>

Slower!
Address is not aligned
Refining Contract Template - Ibex Memory Interface

Program 1

```
li a0, 0x100
lw a1, 0(a0)
```

Program 2

```
li a0, 0x102
lw a1, 0(a0)
```

Refinement of template to capture:
- alignment of accesses
- branch outcomes
- data dependencies

**Slower!**

Address is not aligned
Refined Contract Template - Precision

\[ \text{Precision} = \frac{\text{True Positives}}{\text{True Positives} + \text{False Positives}} \]

Final precision: 72.11%
Thank you for your attention!


Play with the artifact: https://github.com/hw-sw-contracts/riscv-contract-synthesis
https://zenodo.org/records/10491534