Verifying the Security of Microarchitectures based on Hardware-Software Contracts

Jan Reineke @ Saarland University

Joint work with
Zilong Wang, Marco Guarnieri @ IMDEA Software, Madrid
Klaus v. Gleissenthall @ VU Amsterdam
Gideon Mohr, Valentin Touzeau @ Saarland University

This project has received funding from the European Research Council (ERC) under the European Union’s Horizon 2020 research and innovation programme (grant agreement No 101020415)”
An Introduction to HW/SW Contracts
ISA: Benefits

High-level language

Instruction set architecture (ISA)

Microarchitecture
ISA: Benefits

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

Can program independently of microarchitecture
ISA: 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 the ISA: Side channels

High-level language

Instruction set architecture (ISA)

Microarchitecture
Inadequacy of the ISA: Side channels

High-level language

Instruction set architecture (ISA)

No guarantees about side channels

Microarchitecture
Inadequacy of the ISA: Side channels

- Instruction set architecture (ISA)
- Microarchitecture

No guarantees about side channels

Can implement arbitrary **insecure** optimizations as long as ISA is implemented correctly
Inadequacy of the ISA: Side channels

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

No guarantees about side channels

Can implement arbitrary \textit{insecure} optimizations as long as ISA is implemented correctly.
Inadequacy of the ISA: Side channels

High-level language

Instruction set architecture (ISA)

Microarchitecture

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
A way forward: HW/SW security contracts

**HW/SW contract = ISA + X**

Succinctly captures possible information leakage in a mechanism-independent way
A way forward: HW/SW security contracts

Can program securely on top of contract independently of microarchitecture

HW/SW contract = ISA + X

Succinctly captures possible information leakage in a mechanism-independent way
A way forward: HW/SW security contracts

Can program securely on top of contract independently of microarchitecture

\[ \text{HW/SW contract} = \text{ISA} + \text{X} \]

Succinctly captures possible information leakage in a mechanism-independent way

Can implement arbitrary insecure optimizations as long as contract is obeyed
HW/SW contracts

Contracts specify which program executions a side-channel adversary cannot distinguish.

Contract

ISA
+
Observations
HW/SW contracts

Contracts specify which program executions a side-channel adversary cannot distinguish

Contract
ISA + Observations

Captures how program is executed
HW/SW contracts

Contracts specify which program executions a side-channel adversary cannot distinguish.

Contract

ISA + Observations

Captures how program is executed.

What leaks about an execution.
Contracts
Contracts

Contract
ISA +
Observations
Contracts

Contract traces: $\mathcal{C}(p, \sigma)$
Contracts

**Contract**
ISA +
Observations

**Contract traces:** $(p, \sigma)$

**Hardware**
µArch design +
Attacker observations
Contracts

Contract
ISA +
Observations

Contract traces: \((p, \sigma)\)

Hardware
\(\mu\)Arch design +
Attacker observations

Hardware traces: \((p, \sigma)\)
Contracts

Contract traces: $\mathcal{C}(p, \sigma)\]

Hardware traces: $\mathcal{H}(p, \sigma)$

Contract satisfaction

Hardware satisfies contract if for all programs $p$ and arch. states $\sigma$, $\sigma'$: if $\mathcal{C}(p, \sigma) = \mathcal{C}(p, \sigma')$ then $\mathcal{H}(p, \sigma) = \mathcal{H}(p, \sigma')$
Contracts

Contract
ISA +
Observations
Contract traces: \((p, \sigma)\)

Hardware
\(\mu\)Arch design +
Attacker observations
Hardware traces: \((p, \sigma)\)

Contract satisfaction
Hardware satisfies contract if for all programs \(p\) and arch. states \(\sigma, \sigma'\): if \((p, \sigma) = (p, \sigma')\) then \((p, \sigma) = (p, \sigma')\)
Contracts

Contract
ISA +
Observations

Contract traces: $\square(p, \sigma)$

Hardware
$\mu$Arch design +
Attacker observations

Hardware traces: $\blacksquare(p, \sigma)$

Contract satisfaction
Hardware $\blacksquare$ satisfies contract $\square$ if for all programs $p$ and arch.
states $\sigma, \sigma'$: if $\square(p, \sigma) = \square(p, \sigma')$ then $\blacksquare(p, \sigma) = \blacksquare(p, \sigma')$
Verifying Contract Satisfaction
Contracts for Real ISAs + Real CPUs

Microarchitecture ⊨ Contract
Contracts for Real ISAs + Real CPUs

Microarchitecture

⊨

Register transfer level designs

Contract
Contracts for Real ISAs + Real CPUs

Microarchitecture \(\models\) Contract

Register transfer level designs

Open-source RISC-V cores
Contracts for Real ISAs + Real CPUs

Microarchitecture

<table>
<thead>
<tr>
<th>Register transfer level designs</th>
</tr>
</thead>
<tbody>
<tr>
<td>Open-source RISC-V cores</td>
</tr>
</tbody>
</table>

Contract

| ISA spec. + Observations       |
Contracts for Real ISAs + Real CPUs

Microarchitecture
- Register transfer level designs
- Open-source RISC-V cores

Contract
- ISA spec. + Observations

1. Sail ([https://github.com/rems-project/sail](https://github.com/rems-project/sail))
Contracts for Real ISAs + Real CPUs

Microarchitecture

Register transfer level designs
Open-source RISC-V cores

Contract

ISA spec. + Observations

1. Sail (https://github.com/rems-project/sail)
2. Single-cycle RISC-V reference implementation + Combinatorial observer
Contracts for Real ISAs + Real CPUs

Microarchitecture

- Register transfer level designs
- Open-source RISC-V cores

Contract

- Automatic proof
- ISA spec. + Observations

1. Sail (https://github.com/rems-project/sail)
2. Single-cycle RISC-V reference implementation + Combinatorial observer
Contracts for Real ISAs + Real CPUs

Microarchitecture

- Register transfer level designs
- Open-source RISC-V cores

Contract

- Automatic proof
- SMT solvers + Invariant inference
- ISA spec. + Observations

1. Sail (https://github.com/rem-s-project/sail)
2. Single-cycle RISC-V reference implementation + Combinatorial observer
Verification: 4-way product circuit + Induction

Contract satisfaction

Hardware satisfies contract if for all programs $p$ and arch. states $\sigma$, $\sigma'$: if $\square(p, \sigma) = \square(p, \sigma')$ then $\square(p, \sigma) = \square(p, \sigma')$
Verification: 4-way product circuit + Induction

Contract satisfaction
Hardware $\Box$ satisfies contract $\square$ if for all programs $p$ and arch. states $\sigma, \sigma'$: if $\square(p, \sigma) = \square(p, \sigma')$ then $\Box(p, \sigma) = \Box(p, \sigma')$

$ISA_1$

$\approx ISA$

$HW_1$
Verification: 4-way product circuit + Induction

**Contract satisfaction**

Hardware \( \approx \) satisfies contract \( = \) if for all programs \( p \) and arch. states \( \sigma, \sigma' \): if \( (p, \sigma) = (p, \sigma') \) then \( (p, \sigma) = (p, \sigma') \)

\[
\begin{align*}
\text{ISA}_1 & \approx \text{ISA} \\
\text{HW}_1 & \approx \text{ISA} \\
\text{ISA}_2 & \\
\text{HW}_2 &
\end{align*}
\]
Verification: 4-way product circuit + Induction

Contract satisfaction
Hardware satisfies contract if for all programs $p$ and arch. states $\sigma, \sigma'$: if $(p, \sigma) = (p, \sigma')$ then $(p, \sigma) = (p, \sigma')$
Verification: 4-way product circuit + Induction

Contract satisfaction

Hardware $\approx$ satisfies contract $\approx$ if for all programs $p$ and arch. states $\sigma, \sigma'$: if $\approx(p, \sigma) = \approx(p, \sigma')$ then $\approx(p, \sigma) = \approx(p, \sigma')$
Verification: 4-way product circuit + Induction

ISA₁ $\approx$ ISA₁'

ISA₂ $\approx$ ISA₂'

HW₁ $\approx$ HW₁'

HW₂ $\approx$ HW₂'
Verification: 4-way product circuit + Induction

\[
\begin{align*}
ISA_1 & \approx ISA_1' \\
ISA_2 & \approx ISA_2' \\
HW_1 & \approx HW_1' \\
HW_2 & \approx HW_2'
\end{align*}
\]

Relational Inductive Invariant
Verification: 4-way product circuit + Induction

ISA$_1$ \[\approx\] ISA$_1'$

ISA$_2$ \[\approx\] ISA$_2'$

HW$_1$ \[\approx\] HW$_1'$

HW$_2$ \[\approx\] HW$_2'$

Relational Inductive Invariant
Verification: 4-way product circuit + Induction

\[ \text{ISA}_1 \approx \text{ISA}_1' \]
\[ \text{ISA}_2 \approx \text{ISA}_2' \]
\[ \text{HW}_1 \approx \text{Inv} \approx \text{HW}_1' \]
\[ \text{HW}_2 \approx \text{Inv} \approx \text{HW}_2' \]

Relational
Inductive
Invariant

... synthesized using Houdini algorithm
(Flanagan+Leino: Houdini, an Annotation Assistant for ESC/Java, Formal Methods Europe)
Verification: 4-way product circuit + Induction

Works on small examples. RISC-V Sodor Work-in-progress

... synthesized using Houdini algorithm
(Flanagan+Leino: Houdini, an Annotation Assistant for ESC/Java, Formal Methods Europe)
Verification: 4-way product circuit + Induction

\[ \text{ISA}_1 \approx \text{ISA}_1' \]
\[ \text{ISA}_2 \approx \text{ISA}_2' \]
\[ \text{HW}_1 \approx \text{HW}_1' \]
\[ \text{HW}_2 \approx \text{HW}_2' \]

Relational Inductive Invariant

Works on small examples. RISC-V Sodor Work-in-progress

Scalability limited due to 4-way product circuit

... synthesized using Houdini algorithm
(Flanagan+Leino: Houdini, an Annotation Assistant for ESC/Java, Formal Methods Europe)
Disentangling Leakage and ISA satisfaction

Contract satisfaction
Hardware satisfies contract if for all programs $p$ and states $\sigma, \sigma'$:

if $\text{Obs}(\text{ISA})(p, \sigma) = \text{Obs}(\text{ISA})(p, \sigma')$

then $\text{Atk}(\mu\text{Arch})(p, \sigma) = \text{Atk}(\mu\text{Arch})(p, \sigma')$
Disentangling Leakage and ISA satisfaction

**Contract satisfaction**

Hardware satisfies contract if for all programs \( p \) and states \( \sigma, \sigma' \):

\[
\text{if } \text{Obs(ISA)}(p, \sigma) = \text{Obs(ISA)}(p, \sigma') \text{ then } \text{Atk}(\mu\text{Arch})(p, \sigma) = \text{Atk}(\mu\text{Arch})(p, \sigma')
\]

**ISA satisfaction** + **Leakage satisfaction**
ISA satisfaction

\[ \text{ISA}_0 \xRightarrow{\approx} \text{ISA} \xRightarrow{\approx} \text{HW}_0 \]
ISA satisfaction

ISA₀ → ISA₁

ISA

HW₀
ISA satisfaction

\[ \text{ISA}_0 \rightarrow \text{ISA}_1 \]

\[ \text{HW}_0 \rightarrow \rightarrow \rightarrow \text{HW}_{i_1} \]
ISA satisfaction

ISA_0 \rightarrow ISA_1 \rightarrow ISA_2

\sim ISA

HW_0 \rightarrow \rightarrow \rightarrow HW_{i_1}
ISA satisfaction

ISA₀ → ISA₁ → ISA₂

HW₀ → HWᵢ₁ → HWᵢ₂
ISA satisfaction

\[ ISA_0 \rightarrow ISA_1 \rightarrow ISA_2 \rightarrow ISA_3 \]

\[ HW_0 \rightarrow HW_{i1} \rightarrow HW_{i2} \]
ISA satisfaction

ISA₀ → ISA₁ → ISA₂ → ISA₃

HW₀ → HWᵢ₁ → HWᵢ₂ → HWᵢ₃
ISA satisfaction

$ISA_0 \rightarrow ISA_1 \rightarrow ISA_2 \rightarrow ISA_3 \rightarrow \ldots$

$HW_0 \rightarrow HW_{i_1} \rightarrow HW_{i_2} \rightarrow HW_{i_3} \rightarrow \ldots$
ISA satisfaction

The ISA trace is embedded in the HW trace.
Leakage satisfaction

For all programs \( p \) and states \( \sigma, \sigma' \):

\[
\text{if } \text{Obs}(\mu\text{Arch})(p, \sigma) = \text{Obs}(\mu\text{Arch})(p, \sigma')
\text{ then } \text{Atk}(\mu\text{Arch})(p, \sigma) = \text{Atk}(\mu\text{Arch})(p, \sigma')
\]

+ Observations determine the timing of instruction retirement.
Leakage satisfaction

For all programs $p$ and states $\sigma, \sigma'$:

If $\text{Obs}(\mu\text{Arch})(p, \sigma) = \text{Obs}(\mu\text{Arch})(p, \sigma')$

then $\text{Atk}(\mu\text{Arch})(p, \sigma) = \text{Atk}(\mu\text{Arch})(p, \sigma')$

+ Observations determine the timing of instruction retirement.

Can be checked with 2-way product circuit.
Leakage satisfaction

For all programs $p$ and states $\sigma, \sigma'$:

if $\text{Obs}(\mu\text{Arch})(p, \sigma) = \text{Obs}(\mu\text{Arch})(p, \sigma')$
then $\text{Atk}(\mu\text{Arch})(p, \sigma) = \text{Atk}(\mu\text{Arch})(p, \sigma')$

+ Observations determine the timing of instruction retirement.

Can be checked with 2-way product circuit.

Verified DarkRISCV leakage
Leakage satisfaction

For all programs $p$ and states $\sigma, \sigma'$:

\[
\text{if } \text{Obs}(\mu\text{Arch})(p, \sigma) = \text{Obs}(\mu\text{Arch})(p, \sigma') \cr \text{then } \text{Atk}(\mu\text{Arch})(p, \sigma) = \text{Atk}(\mu\text{Arch})(p, \sigma')
\]

+ Observations determine the timing of instruction retirement.

Can be checked with 2-way product circuit.

Verified DarkRISCV leakage

Proof relies on "pipeline invariants"
Pipeline invariants: Example
Consider
R0 ← DIV R1, R2
Consider

\( R0 \leftarrow \text{DIV} \ R1, \ R2 \)

Contract:

“Reg[R2] leaks”
Consider
R0 ← DIV R1, R2

Contract: “Reg[R2] leaks”
Pipeline invariants: Example

Consider

\[ R_0 \leftarrow \text{DIV} \ R_1, \ R_2 \]

Contract: "Reg[R2] leaks"
Consider: \( R0 \leftarrow \text{DIV} \ R1, \ R2 \)

**Contract:**
“Reg\( [R2] \) leaks”

**Pipeline invariant allows to evaluate contract when leakage occurs**
Conclusions
Conclusions

HW/SW contracts capture leakage at ISA level
... thus enable secure programming
Conclusions

HW/SW contracts capture leakage at ISA level … thus enable secure programming

Verifying microarchitectures is challenging … leakage verification and functional verification can be disentangled
Conclusions

HW/SW contracts capture leakage at ISA level
... thus enable secure programming

Verifying microarchitectures is challenging
... leakage verification and functional verification can be disentangled

Come to our poster to discuss!