MIRACLE: MIcRo-ArChitectural Leakage Evaluation

A study of micro-architectural power leakage across many devices

Ben Marshall[1,2], Dan Page[1] and James Webb[1]

[1] Department of Computer Science, University of Bristol,
Merchant Venturers Building, Woodland Road,
Bristol, BS8 1UB, UK.
{ben.marshall,daniel.page,james.webb}@bristol.ac.uk
[2] PQShield Ltd, Oxford, UK.
ben.marshall@pqshield.com

Outline

Context - Some Background

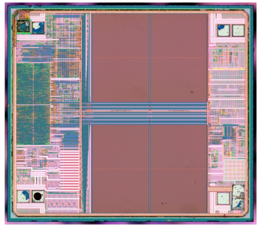
- We all want secure computing & communication, cheaply & quickly.
- It’s a multi-dimensional problem.
- Security models rely on abstractions like HW / ISA / SW.
- Good abstractions enable formal reasoning & proofs of security *over that abstraction*.
- Unless the abstraction captures everything necessary to model threats, you can’t prove security.
- Does the abstraction actually hold for the implementation?

```
int main (int argc, char ** argv)
{
        printf("Hello, CHES!\n");
        return rand() & 0x1;
}
```

Software

RISC-V intel arm

ISAs



Digital (micro-arch)

Analog

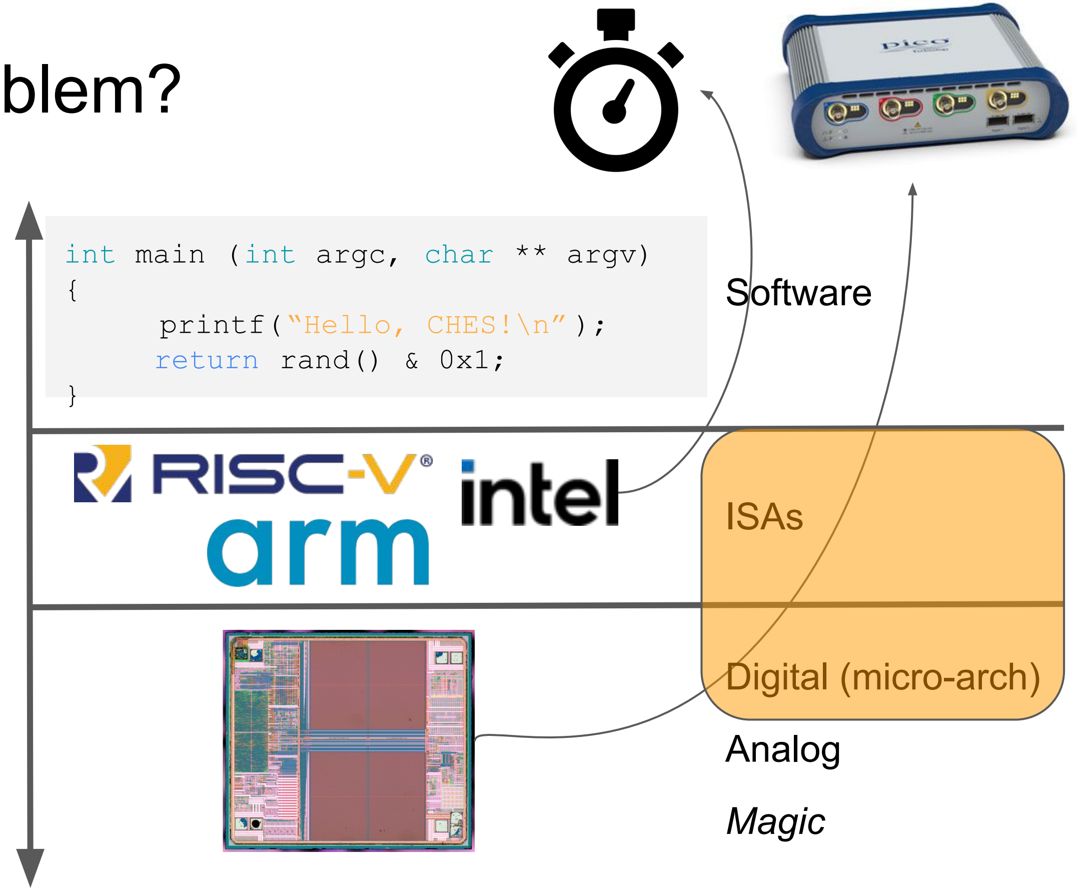
Magic

University of BRISTOL PQSHIELD

* Die shot from Zeptobars.com

Context - What's the problem?

- The ISA is an excellent abstraction for functionality, but leaks implementation details.
- Without being able to build good abstractions, it's harder to reason about security / correctness.
- Start by understanding what the abstraction does (not) capture.
- In the past, we've looked at power side-channels mostly in the context of just one device…
- We can't model security without being able to understand the abstractions we rely on.

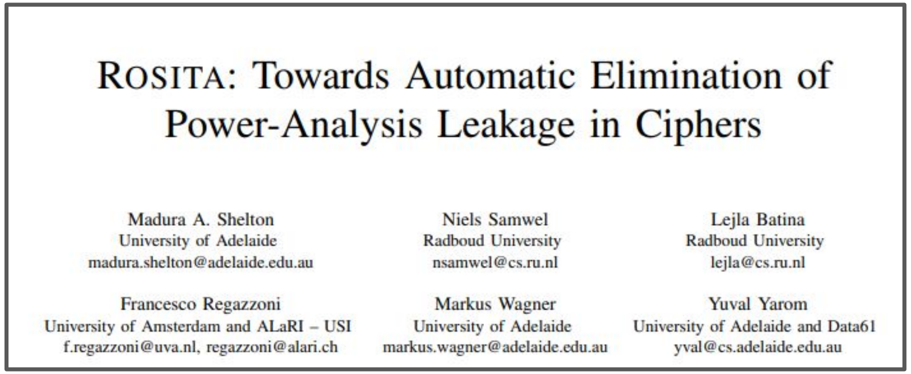


```
int main (int argc, char ** argv)
{
        printf("Hello, CHES!\n");
        return rand() & 0x1;
}
```

Goals: What did we set out to achieve?

- Understand the ISA abstraction from the perspective of micro-architectural power leakage.
- Try to replicate “folklore” and past results from previous works.
- See if we can use our knowledge of CPU design to find new effects.
- See if these effects are present across a range of devices.
- Create a suite of “micro-benchmarks” which can be reused across new devices to find effects and qualify models.
- Qualitatively assess how different devices will leak differently.
- Create a framework for understanding and presenting all of this.
    - (A paper was *not* a good way of doing it…)

Context: (Some) Related Work



ROSITA: Towards Automatic Elimination of Power-Analysis Leakage in Ciphers

Madura A. Shelton
University of Adelaide
madura.shelton@adelaide.edu.au

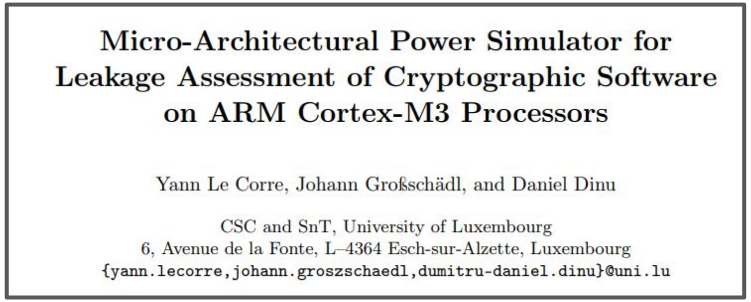
Niels Samwel
Radboud University
nsamwel@cs.ru.nl

Lejla Batina
Radboud University
lejla@cs.ru.nl

Francesco Regazzoni
University of Amsterdam and ALaRI – USI
f.regazzoni@uva.nl, regazzoni@alari.ch

Markus Wagner
University of Adelaide
markus.wagner@adelaide.edu.au

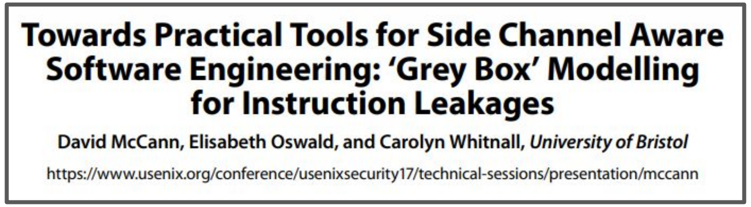
Yuval Yarom
University of Adelaide and Data61
yval@cs.adelaide.edu.au



Micro-Architectural Power Simulator for Leakage Assessment of Cryptographic Software on ARM Cortex-M3 Processors

Yann Le Corre, Johann Großschädl, and Daniel Dinu

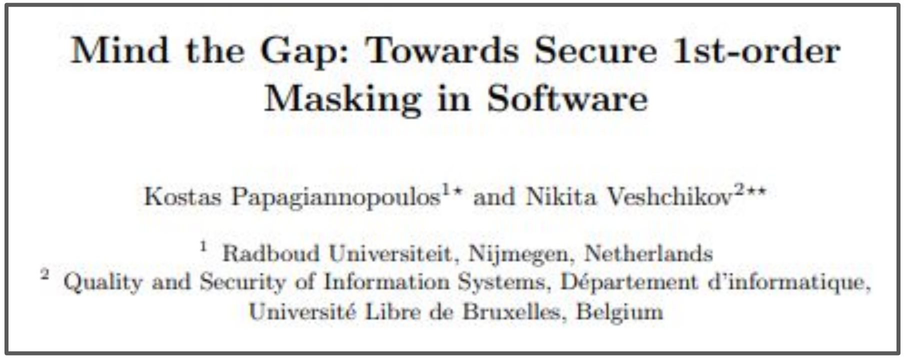
CSC and SnT, University of Luxembourg
6, Avenue de la Fonte, L–4364 Esch-sur-Alzette, Luxembourg
{yann.lecorre,johann.groszschaedl,dumitru-daniel.dinu}@uni.lu



Towards Practical Tools for Side Channel Aware Software Engineering: 'Grey Box' Modelling for Instruction Leakages

David McCann, Elisabeth Oswald, and Carolyn Whitnall, *University of Bristol*

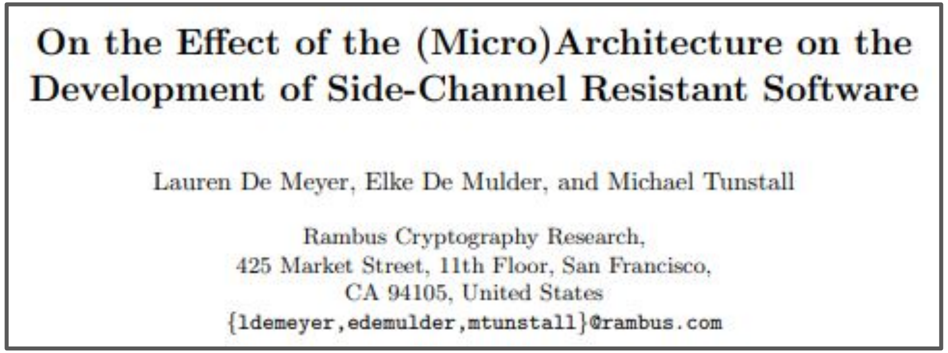
https://www.usenix.org/conference/usenixsecurity17/technical-sessions/presentation/mccann



Mind the Gap: Towards Secure 1st-order Masking in Software

Kostas Papagiannopoulos[1]* and Nikita Veshchikov[2]**

[1] Radboud Universiteit, Nijmegen, Netherlands
[2] Quality and Security of Information Systems, Département d'informatique, Université Libre de Bruxelles, Belgium



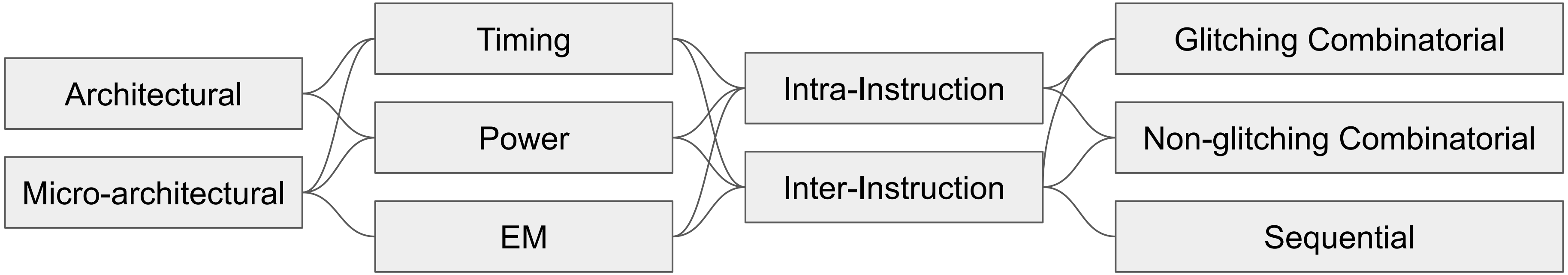
On the Effect of the (Micro)Architecture on the Development of Side-Channel Resistant Software

Lauren De Meyer, Elke De Mulder, and Michael Tunstall

Rambus Cryptography Research,
425 Market Street, 11th Floor, San Francisco,
CA 94105, United States
{ldemeyer,edemulder,mtunstall}@rambus.com

Terminology: My implementation is broken because…

- {**Architectural**, ***Micro-architectural***} leakage from a {**Timing**, ***Power***, ***EM***} Channel.
- The Leakage is {**Inter**, ***Intra***}-instruction in nature.
- As Power/EM leakage, it stems from {**Sequential**, ***Combinatorial***} circuit behaviour.
- Combinatorial leakage is always intra-cycle, and is {**Glitching**, ***Non-glitching***} in nature.
- Sequential leakage may also be {**Inter**, ***Intra***}-instruction.



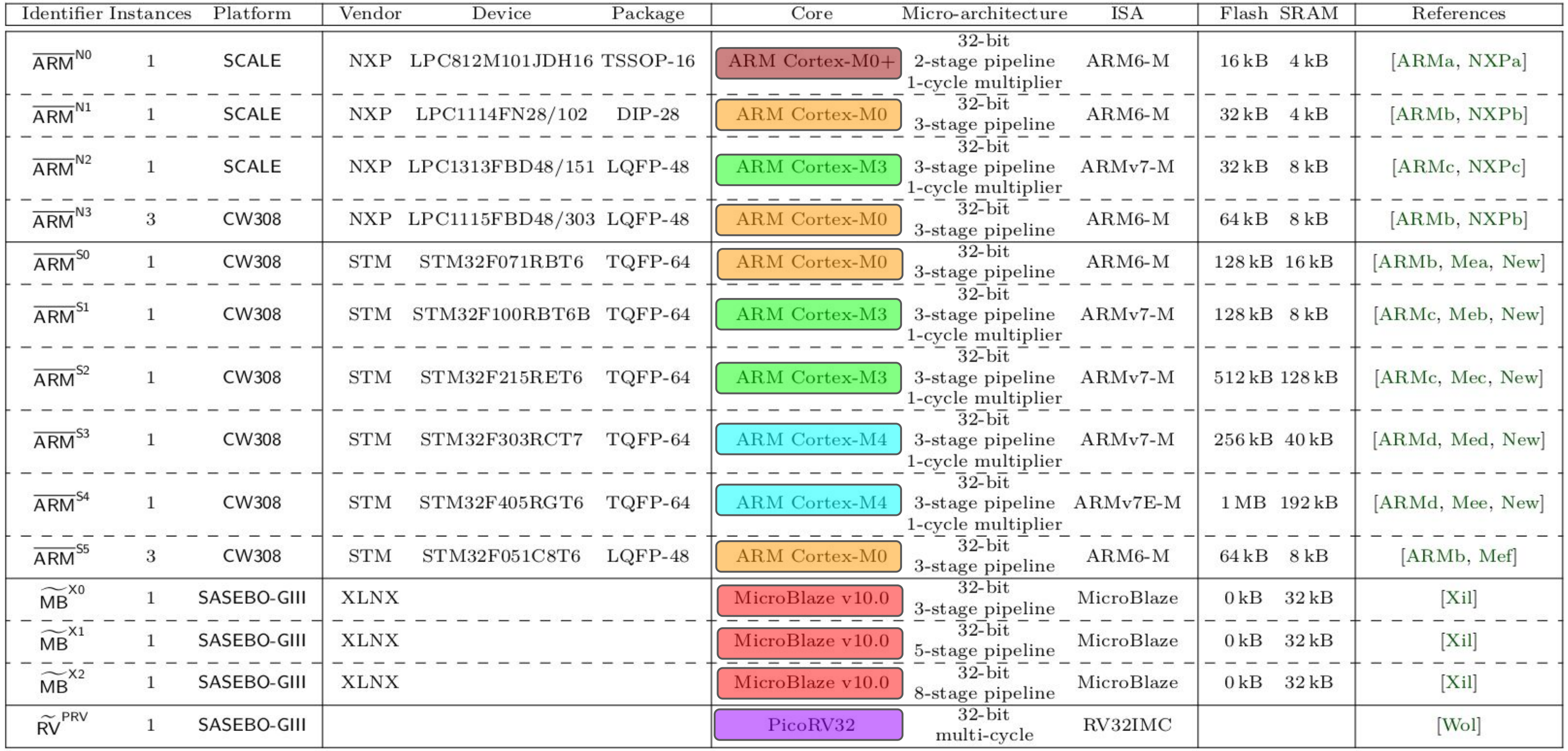
Identifier	Instances	Platform	Vendor	Device	Package	Core	Micro-architecture	ISA	Flash	SRAM	References
$\overline{\mathsf{ARM}}^{N0}$	1	SCALE	NXP	LPC812M101JDH16	TSSOP-16	ARM Cortex-M0+	32-bit 2-stage pipeline 1-cycle multiplier	ARM6-M	16 kB	4 kB	[ARMa, NXPa]
$\overline{\mathsf{ARM}}^{N1}$	1	SCALE	NXP	LPC1114FN28/102	DIP-28	ARM Cortex-M0	32-bit 3-stage pipeline	ARM6-M	32 kB	4 kB	[ARMb, NXPb]
$\overline{\mathsf{ARM}}^{N2}$	1	SCALE	NXP	LPC1313FBD48/151	LQFP-48	ARM Cortex-M3	32-bit 3-stage pipeline 1-cycle multiplier	ARMv7-M	32 kB	8 kB	[ARMc, NXPc]
$\overline{\mathsf{ARM}}^{N3}$	3	CW308	NXP	LPC1115FBD48/303	LQFP-48	ARM Cortex-M0	32-bit 3-stage pipeline	ARM6-M	64 kB	8 kB	[ARMb, NXPb]
$\overline{\mathsf{ARM}}^{S0}$	1	CW308	STM	STM32F071RBT6	TQFP-64	ARM Cortex-M0	32-bit 3-stage pipeline	ARM6-M	128 kB	16 kB	[ARMb, Mea, New]
$\overline{\mathsf{ARM}}^{S1}$	1	CW308	STM	STM32F100RBT6B	TQFP-64	ARM Cortex-M3	32-bit 3-stage pipeline 1-cycle multiplier	ARMv7-M	128 kB	8 kB	[ARMc, Meb, New]
$\overline{\mathsf{ARM}}^{S2}$	1	CW308	STM	STM32F215RET6	TQFP-64	ARM Cortex-M3	32-bit 3-stage pipeline 1-cycle multiplier	ARMv7-M	512 kB	128 kB	[ARMc, Mec, New]
$\overline{\mathsf{ARM}}^{S3}$	1	CW308	STM	STM32F303RCT7	TQFP-64	ARM Cortex-M4	32-bit 3-stage pipeline 1-cycle multiplier	ARMv7-M	256 kB	40 kB	[ARMd, Med, New]
$\overline{\mathsf{ARM}}^{S4}$	1	CW308	STM	STM32F405RGT6	TQFP-64	ARM Cortex-M4	32-bit 3-stage pipeline 1-cycle multiplier	ARMv7E-M	1 MB	192 kB	[ARMd, Mee, New]
$\overline{\mathsf{ARM}}^{S5}$	3	CW308	STM	STM32F051C8T6	LQFP-48	ARM Cortex-M0	32-bit 3-stage pipeline	ARM6-M	64 kB	8 kB	[ARMb, Mef]
$\widetilde{\mathsf{MB}}^{X0}$	1	SASEBO-GIII	XLNX			MicroBlaze v10.0	32-bit 3-stage pipeline	MicroBlaze	0 kB	32 kB	[Xil]
$\widetilde{\mathsf{MB}}^{X1}$	1	SASEBO-GIII	XLNX			MicroBlaze v10.0	32-bit 5-stage pipeline	MicroBlaze	0 kB	32 kB	[Xil]
$\widetilde{\mathsf{MB}}^{X2}$	1	SASEBO-GIII	XLNX			MicroBlaze v10.0	32-bit 8-stage pipeline	MicroBlaze	0 kB	32 kB	[Xil]
$\widetilde{\mathsf{RV}}^{PRV}$	1	SASEBO-GIII				PicoRV32	32-bit multi-cycle	RV32IMC			[Wol]

3 Different Evaluation Platforms

Identifier	Instances	Platform	Vendor	Device	Package	Core	Micro-architecture	ISA	Flash	SRAM	References
$\overline{\text{ARM}}^{N0}$	1	SCALE	NXP	LPC812M101JDH16	TSSOP-16	ARM Cortex-M0+	32-bit 2-stage pipeline 1-cycle multiplier	ARM6-M	16 kB	4 kB	[ARMa, NXPa]
$\overline{\text{ARM}}^{N1}$	1	SCALE	NXP	LPC1114FN28/102	DIP-28	ARM Cortex-M0	32-bit 3-stage pipeline	ARM6-M	32 kB	4 kB	[ARMb, NXPb]
$\overline{\text{ARM}}^{N2}$	1	SCALE	NXP	LPC1313FBD48/151	LQFP-48	ARM Cortex-M3	32-bit 3-stage pipeline 1-cycle multiplier	ARMv7-M	32 kB	8 kB	[ARMc, NXPc]
$\overline{\text{ARM}}^{N3}$	3	CW308	NXP	LPC1115FBD48/303	LQFP-48	ARM Cortex-M0	32-bit 3-stage pipeline	ARM6-M	64 kB	8 kB	[ARMb, NXPb]
$\overline{\text{ARM}}^{S0}$	1	CW308	STM	STM32F071RBT6	TQFP-64	ARM Cortex-M0	32-bit 3-stage pipeline	ARM6-M	128 kB	16 kB	[ARMb, Mea, New]
$\overline{\text{ARM}}^{S1}$	1	CW308	STM	STM32F100RBT6B	TQFP-64	ARM Cortex-M3	32-bit 3-stage pipeline 1-cycle multiplier	ARMv7-M	128 kB	8 kB	[ARMc, Meb, New]
$\overline{\text{ARM}}^{S2}$	1	CW308	STM	STM32F215RET6	TQFP-64	ARM Cortex-M3	32-bit 3-stage pipeline 1-cycle multiplier	ARMv7-M	512 kB	128 kB	[ARMc, Mec, New]
$\overline{\text{ARM}}^{S3}$	1	CW308	STM	STM32F303RCT7	TQFP-64	ARM Cortex-M4	32-bit 3-stage pipeline 1-cycle multiplier	ARMv7-M	256 kB	40 kB	[ARMd, Med, New]
$\overline{\text{ARM}}^{S4}$	1	CW308	STM	STM32F405RGT6	TQFP-64	ARM Cortex-M4	32-bit 3-stage pipeline 1-cycle multiplier	ARMv7E-M	1 MB	192 kB	[ARMd, Mee, New]
$\overline{\text{ARM}}^{S5}$	3	CW308	STM	STM32F051C8T6	LQFP-48	ARM Cortex-M0	32-bit 3-stage pipeline	ARM6-M	64 kB	8 kB	[ARMb, Mef]
$\widetilde{\text{MB}}^{X0}$	1	SASEBO-GIII	XLNX			MicroBlaze v10.0	32-bit 3-stage pipeline	MicroBlaze	0 kB	32 kB	[Xil]
$\widetilde{\text{MB}}^{X1}$	1	SASEBO-GIII	XLNX			MicroBlaze v10.0	32-bit 5-stage pipeline	MicroBlaze	0 kB	32 kB	[Xil]
$\widetilde{\text{MB}}^{X2}$	1	SASEBO-GIII	XLNX			MicroBlaze v10.0	32-bit 8-stage pipeline	MicroBlaze	0 kB	32 kB	[Xil]
$\widetilde{\text{RV}}^{PRV}$	1	SASEBO-GIII				PicoRV32	32-bit multi-cycle	RV32IMC			[Wol]

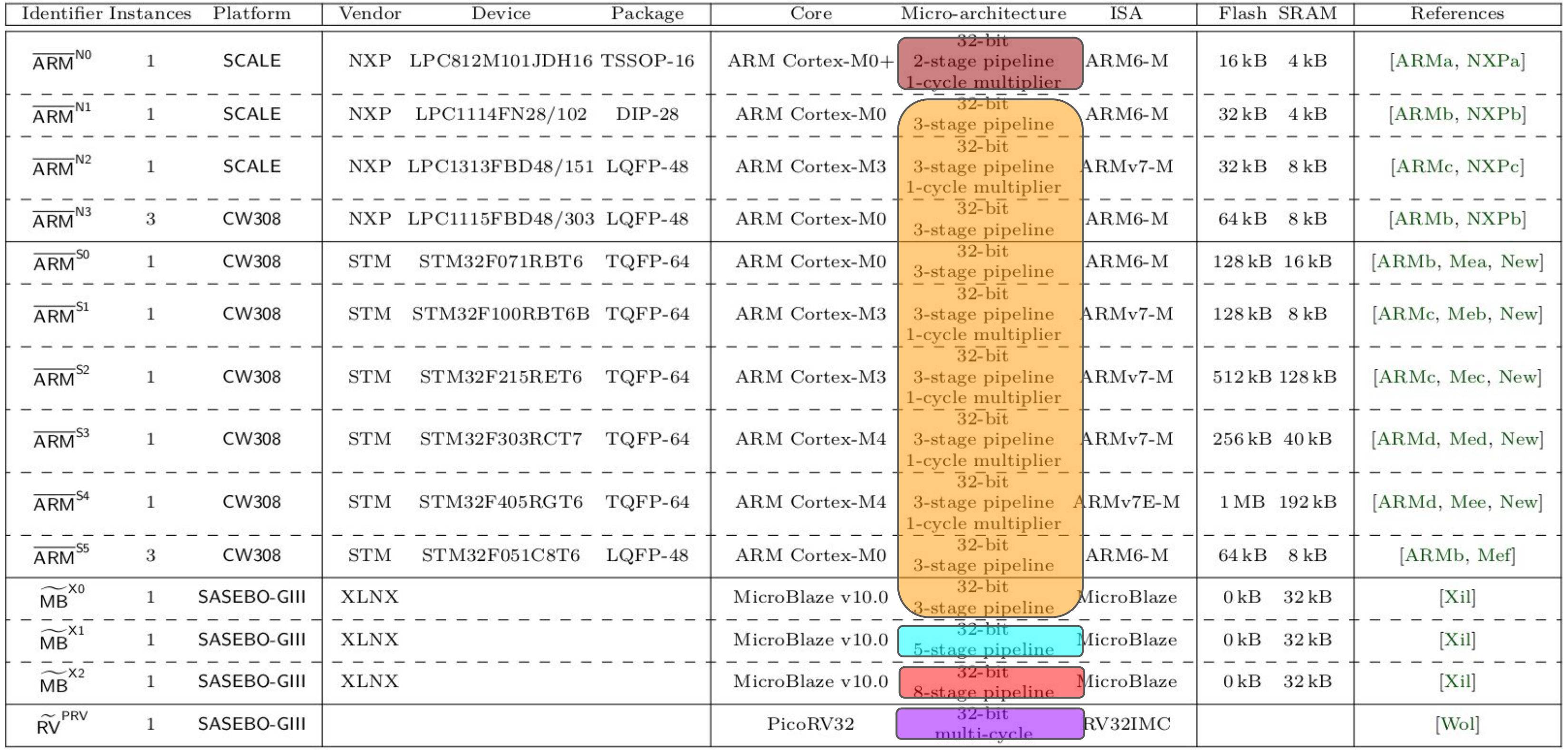
4 Different *device* vendors.

University of BRISTOL PQSHIELD



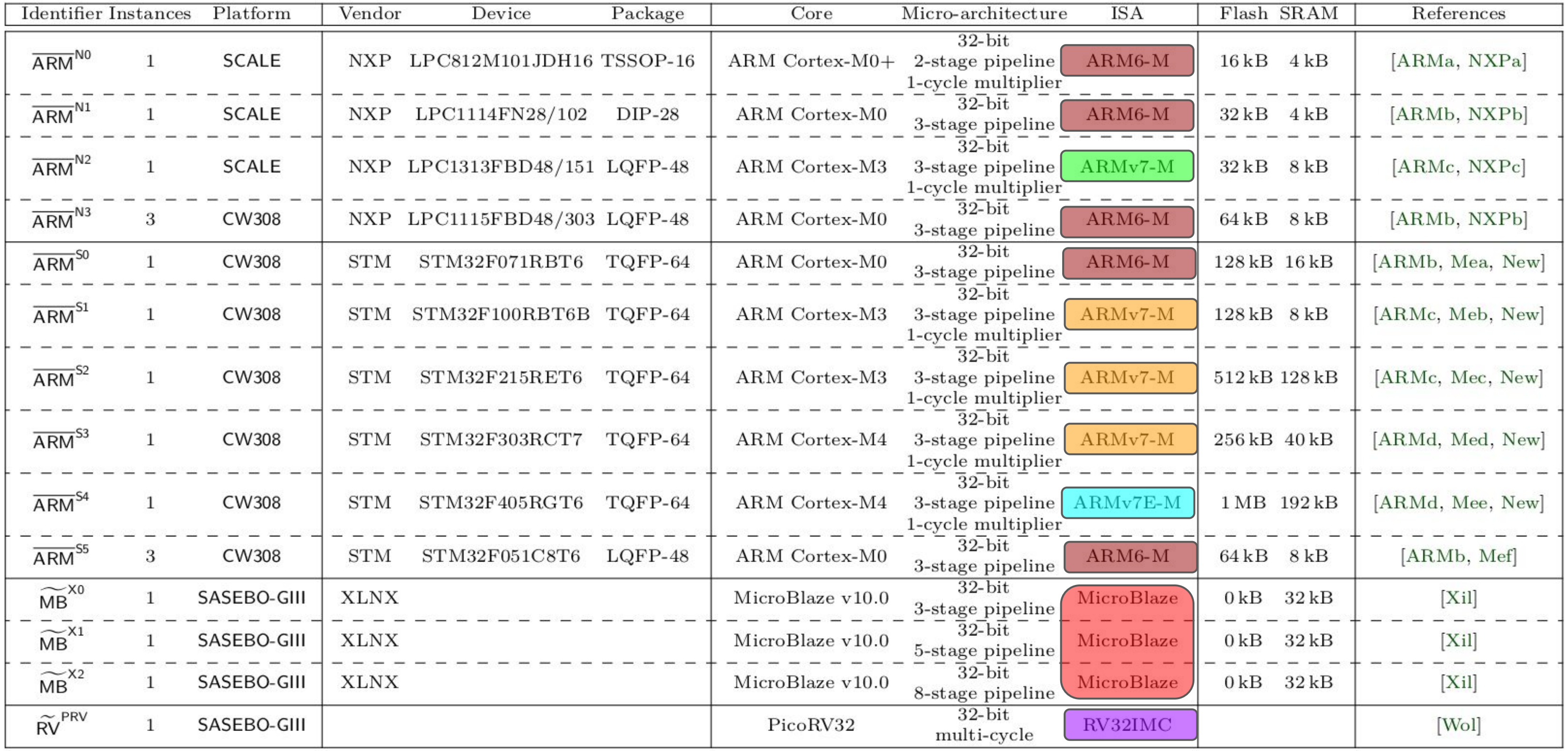
Identifier	Instances	Platform	Vendor	Device	Package	Core	Micro-architecture	ISA	Flash	SRAM	References
$\overline{\text{ARM}}^{N0}$	1	SCALE	NXP	LPC812M101JDH16	TSSOP-16	ARM Cortex-M0+	32-bit 2-stage pipeline 1-cycle multiplier	ARM6-M	16 kB	4 kB	[ARMa, NXPa]
$\overline{\text{ARM}}^{N1}$	1	SCALE	NXP	LPC1114FN28/102	DIP-28	ARM Cortex-M0	32-bit 3-stage pipeline	ARM6-M	32 kB	4 kB	[ARMb, NXPb]
$\overline{\text{ARM}}^{N2}$	1	SCALE	NXP	LPC1313FBD48/151	LQFP-48	ARM Cortex-M3	32-bit 3-stage pipeline 1-cycle multiplier	ARMv7-M	32 kB	8 kB	[ARMc, NXPc]
$\overline{\text{ARM}}^{N3}$	3	CW308	NXP	LPC1115FBD48/303	LQFP-48	ARM Cortex-M0	32-bit 3-stage pipeline	ARM6-M	64 kB	8 kB	[ARMb, NXPb]
$\overline{\text{ARM}}^{S0}$	1	CW308	STM	STM32F071RBT6	TQFP-64	ARM Cortex-M0	32-bit 3-stage pipeline	ARM6-M	128 kB	16 kB	[ARMb, Mea, New]
$\overline{\text{ARM}}^{S1}$	1	CW308	STM	STM32F100RBT6B	TQFP-64	ARM Cortex-M3	32-bit 3-stage pipeline 1-cycle multiplier	ARMv7-M	128 kB	8 kB	[ARMc, Meb, New]
$\overline{\text{ARM}}^{S2}$	1	CW308	STM	STM32F215RET6	TQFP-64	ARM Cortex-M3	32-bit 3-stage pipeline 1-cycle multiplier	ARMv7-M	512 kB	128 kB	[ARMc, Mec, New]
$\overline{\text{ARM}}^{S3}$	1	CW308	STM	STM32F303RCT7	TQFP-64	ARM Cortex-M4	32-bit 3-stage pipeline 1-cycle multiplier	ARMv7-M	256 kB	40 kB	[ARMd, Med, New]
$\overline{\text{ARM}}^{S4}$	1	CW308	STM	STM32F405RGT6	TQFP-64	ARM Cortex-M4	32-bit 3-stage pipeline 1-cycle multiplier	ARMv7E-M	1 MB	192 kB	[ARMd, Mee, New]
$\overline{\text{ARM}}^{S5}$	3	CW308	STM	STM32F051C8T6	LQFP-48	ARM Cortex-M0	32-bit 3-stage pipeline	ARM6-M	64 kB	8 kB	[ARMb, Mef]
$\widetilde{\text{MB}}^{X0}$	1	SASEBO-GIII	XLNX			MicroBlaze v10.0	32-bit 3-stage pipeline	MicroBlaze	0 kB	32 kB	[Xil]
$\widetilde{\text{MB}}^{X1}$	1	SASEBO-GIII	XLNX			MicroBlaze v10.0	32-bit 5-stage pipeline	MicroBlaze	0 kB	32 kB	[Xil]
$\widetilde{\text{MB}}^{X2}$	1	SASEBO-GIII	XLNX			MicroBlaze v10.0	32-bit 8-stage pipeline	MicroBlaze	0 kB	32 kB	[Xil]
$\widetilde{\text{RV}}^{PRV}$	1	SASEBO-GIII				PicoRV32	32-bit multi-cycle	RV32IMC			[Wol]

6 Different CPU models



Identifier	Instances	Platform	Vendor	Device	Package	Core	Micro-architecture	ISA	Flash	SRAM	References
$\overline{\mathsf{ARM}}^{N0}$	1	SCALE	NXP	LPC812M101JDH16	TSSOP-16	ARM Cortex-M0+	32-bit 2-stage pipeline 1-cycle multiplier	ARM6-M	16 kB	4 kB	[ARMa, NXPa]
$\overline{\mathsf{ARM}}^{N1}$	1	SCALE	NXP	LPC1114FN28/102	DIP-28	ARM Cortex-M0	32-bit 3-stage pipeline	ARM6-M	32 kB	4 kB	[ARMb, NXPb]
$\overline{\mathsf{ARM}}^{N2}$	1	SCALE	NXP	LPC1313FBD48/151	LQFP-48	ARM Cortex-M3	32-bit 3-stage pipeline 1-cycle multiplier	ARMv7-M	32 kB	8 kB	[ARMc, NXPc]
$\overline{\mathsf{ARM}}^{N3}$	3	CW308	NXP	LPC1115FBD48/303	LQFP-48	ARM Cortex-M0	32-bit 3-stage pipeline	ARM6-M	64 kB	8 kB	[ARMb, NXPb]
$\overline{\mathsf{ARM}}^{S0}$	1	CW308	STM	STM32F071RBT6	TQFP-64	ARM Cortex-M0	32-bit 3-stage pipeline	ARM6-M	128 kB	16 kB	[ARMb, Mea, New]
$\overline{\mathsf{ARM}}^{S1}$	1	CW308	STM	STM32F100RBT6B	TQFP-64	ARM Cortex-M3	32-bit 3-stage pipeline 1-cycle multiplier	ARMv7-M	128 kB	8 kB	[ARMc, Meb, New]
$\overline{\mathsf{ARM}}^{S2}$	1	CW308	STM	STM32F215RET6	TQFP-64	ARM Cortex-M3	32-bit 3-stage pipeline 1-cycle multiplier	ARMv7-M	512 kB	128 kB	[ARMc, Mec, New]
$\overline{\mathsf{ARM}}^{S3}$	1	CW308	STM	STM32F303RCT7	TQFP-64	ARM Cortex-M4	32-bit 3-stage pipeline 1-cycle multiplier	ARMv7-M	256 kB	40 kB	[ARMd, Med, New]
$\overline{\mathsf{ARM}}^{S4}$	1	CW308	STM	STM32F405RGT6	TQFP-64	ARM Cortex-M4	32-bit 3-stage pipeline 1-cycle multiplier	ARMv7E-M	1 MB	192 kB	[ARMd, Mee, New]
$\overline{\mathsf{ARM}}^{S5}$	3	CW308	STM	STM32F051C8T6	LQFP-48	ARM Cortex-M0	32-bit 3-stage pipeline	ARM6-M	64 kB	8 kB	[ARMb, Mef]
$\widetilde{\mathsf{MB}}^{X0}$	1	SASEBO-GIII	XLNX			MicroBlaze v10.0	32-bit 3-stage pipeline	MicroBlaze	0 kB	32 kB	[Xil]
$\widetilde{\mathsf{MB}}^{X1}$	1	SASEBO-GIII	XLNX			MicroBlaze v10.0	32-bit 5-stage pipeline	MicroBlaze	0 kB	32 kB	[Xil]
$\widetilde{\mathsf{MB}}^{X2}$	1	SASEBO-GIII	XLNX			MicroBlaze v10.0	32-bit 8-stage pipeline	MicroBlaze	0 kB	32 kB	[Xil]
$\widetilde{\mathsf{RV}}^{PRV}$	1	SASEBO-GIII				PicoRV32	32-bit multi-cycle	RV32IMC			[Wol]

5 Different Pipeline Architectures

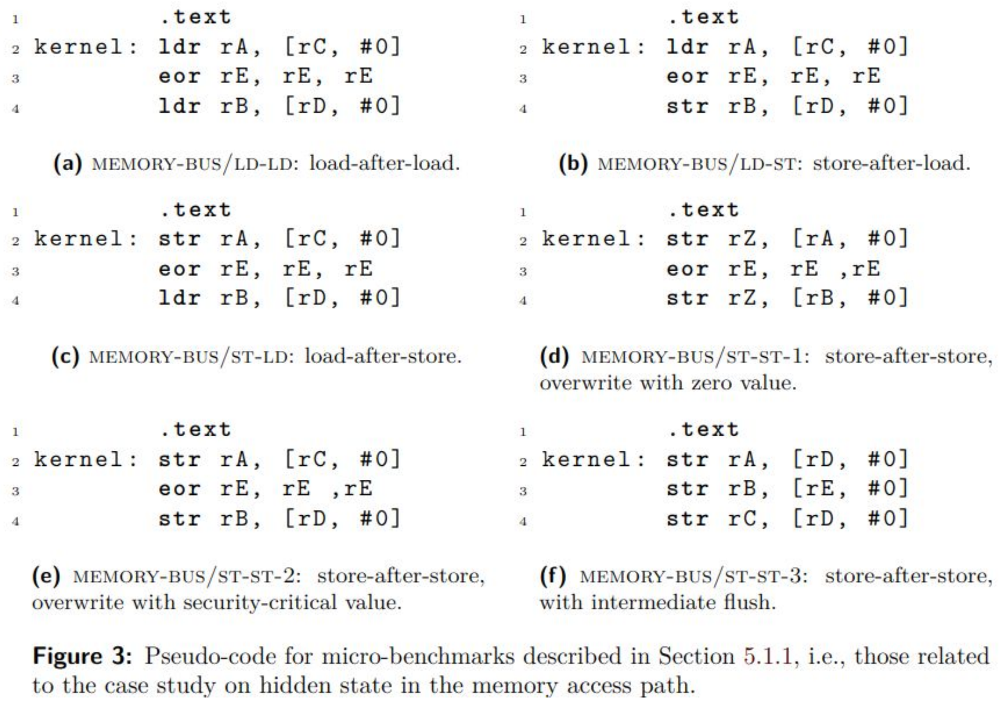


Identifier	Instances	Platform	Vendor	Device	Package	Core	Micro-architecture	ISA	Flash	SRAM	References
$\overline{\mathsf{ARM}}^{N0}$	1	SCALE	NXP	LPC812M101JDH16	TSSOP-16	ARM Cortex-M0+	32-bit 2-stage pipeline 1-cycle multiplier	ARM6-M	16 kB	4 kB	[ARMa, NXPa]
$\overline{\mathsf{ARM}}^{N1}$	1	SCALE	NXP	LPC1114FN28/102	DIP-28	ARM Cortex-M0	32-bit 3-stage pipeline	ARM6-M	32 kB	4 kB	[ARMb, NXPb]
$\overline{\mathsf{ARM}}^{N2}$	1	SCALE	NXP	LPC1313FBD48/151	LQFP-48	ARM Cortex-M3	32-bit 3-stage pipeline 1-cycle multiplier	ARMv7-M	32 kB	8 kB	[ARMc, NXPc]
$\overline{\mathsf{ARM}}^{N3}$	3	CW308	NXP	LPC1115FBD48/303	LQFP-48	ARM Cortex-M0	32-bit 3-stage pipeline	ARM6-M	64 kB	8 kB	[ARMb, NXPb]
$\overline{\mathsf{ARM}}^{S0}$	1	CW308	STM	STM32F071RBT6	TQFP-64	ARM Cortex-M0	32-bit 3-stage pipeline	ARM6-M	128 kB	16 kB	[ARMb, Mea, New]
$\overline{\mathsf{ARM}}^{S1}$	1	CW308	STM	STM32F100RBT6B	TQFP-64	ARM Cortex-M3	32-bit 3-stage pipeline 1-cycle multiplier	ARMv7-M	128 kB	8 kB	[ARMc, Meb, New]
$\overline{\mathsf{ARM}}^{S2}$	1	CW308	STM	STM32F215RET6	TQFP-64	ARM Cortex-M3	32-bit 3-stage pipeline 1-cycle multiplier	ARMv7-M	512 kB	128 kB	[ARMc, Mec, New]
$\overline{\mathsf{ARM}}^{S3}$	1	CW308	STM	STM32F303RCT7	TQFP-64	ARM Cortex-M4	32-bit 3-stage pipeline 1-cycle multiplier	ARMv7-M	256 kB	40 kB	[ARMd, Med, New]
$\overline{\mathsf{ARM}}^{S4}$	1	CW308	STM	STM32F405RGT6	TQFP-64	ARM Cortex-M4	32-bit 3-stage pipeline 1-cycle multiplier	ARMv7E-M	1 MB	192 kB	[ARMd, Mee, New]
$\overline{\mathsf{ARM}}^{S5}$	3	CW308	STM	STM32F051C8T6	LQFP-48	ARM Cortex-M0	32-bit 3-stage pipeline	ARM6-M	64 kB	8 kB	[ARMb, Mef]
$\widetilde{\mathsf{MB}}^{X0}$	1	SASEBO-GIII	XLNX			MicroBlaze v10.0	32-bit 3-stage pipeline	MicroBlaze	0 kB	32 kB	[Xil]
$\widetilde{\mathsf{MB}}^{X1}$	1	SASEBO-GIII	XLNX			MicroBlaze v10.0	32-bit 5-stage pipeline	MicroBlaze	0 kB	32 kB	[Xil]
$\widetilde{\mathsf{MB}}^{X2}$	1	SASEBO-GIII	XLNX			MicroBlaze v10.0	32-bit 8-stage pipeline	MicroBlaze	0 kB	32 kB	[Xil]
$\widetilde{\mathsf{RV}}^{PRV}$	1	SASEBO-GIII				PicoRV32	32-bit multi-cycle	RV32IMC			[Wol]

6 Different Instruction Set Architectures

Selected Content: Micro-benchmarks

- We built 32 different micro-benchmarks.
- Each designed to test some hypothesis about the micro-architecture.
- Using common instructions present in all CPU architectures.
- Split into a high level goal and a low level implementation.



```
         .text
kernel:  ldr rA, [rC, #0]
         eor rE, rE, rE
         ldr rB, [rD, #0]
```

(a) MEMORY-BUS/LD-LD: load-after-load.

```
         .text
kernel:  ldr rA, [rC, #0]
         eor rE, rE, rE
         str rB, [rD, #0]
```

(b) MEMORY-BUS/LD-ST: store-after-load.

```
         .text
kernel:  str rA, [rC, #0]
         eor rE, rE, rE
         ldr rB, [rD, #0]
```

(c) MEMORY-BUS/ST-LD: load-after-store.

```
         .text
kernel:  str rZ, [rA, #0]
         eor rE, rE ,rE
         str rZ, [rB, #0]
```

(d) MEMORY-BUS/ST-ST-1: store-after-store, overwrite with zero value.

```
         .text
kernel:  str rA, [rC, #0]
         eor rE, rE ,rE
         str rB, [rD, #0]
```

(e) MEMORY-BUS/ST-ST-2: store-after-store, overwrite with security-critical value.

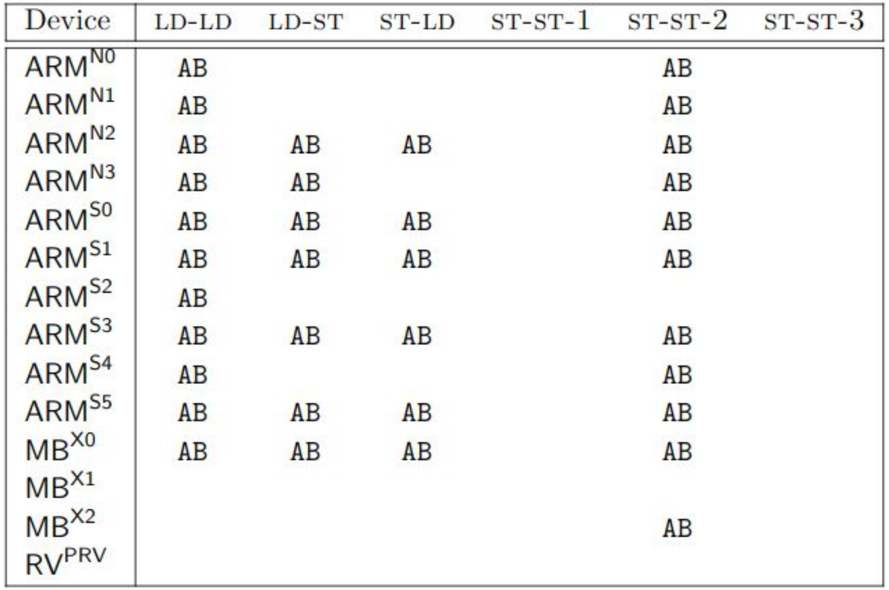
```
         .text
kernel:  str rA, [rD, #0]
         str rB, [rE, #0]
         str rC, [rD, #0]
```

(f) MEMORY-BUS/ST-ST-3: store-after-store, with intermediate flush.

Figure 3: Pseudo-code for micro-benchmarks described in Section 5.1.1, i.e., those related to the case study on hidden state in the memory access path.

Selected Case Study 1: Load/Store & Hidden State

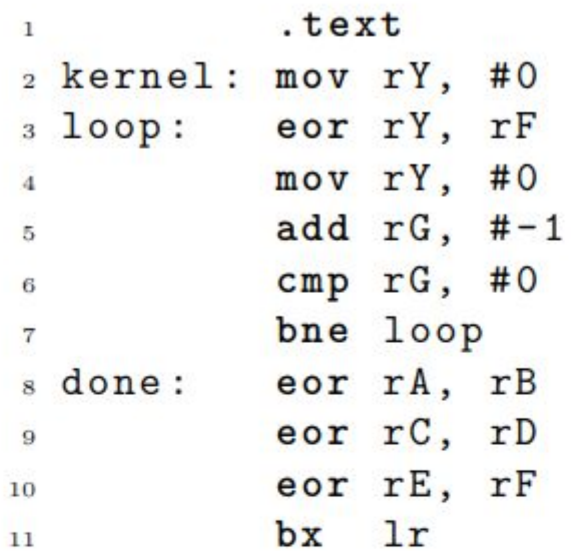
Table 2: A summary of results stemming from the micro-benchmarks in Figure 3, i.e., cases which explore Hamming distance leakage from combinations of `ldr` and `str` instructions. Note that `AC`, for example, indicates that the Hamming distance between `A` and `C` was leaked.



Device	LD-LD	LD-ST	ST-LD	ST-ST-1	ST-ST-2	ST-ST-3
ARM^{N0}	AB				AB	
ARM^{N1}	AB				AB	
ARM^{N2}	AB	AB	AB		AB	
ARM^{N3}	AB	AB			AB	
ARM^{S0}	AB	AB	AB		AB	
ARM^{S1}	AB	AB	AB		AB	
ARM^{S2}	AB					
ARM^{S3}	AB	AB	AB		AB	
ARM^{S4}	AB				AB	
ARM^{S5}	AB	AB	AB		AB	
MB^{X0}	AB	AB	AB		AB	
MB^{X1}						
MB^{X2}					AB	
RV^{PRV}						

Selected Case Study 2: “Speculative” leakage.

- Can instructions which aren’t executed from an architectural perspective still cause leakage?
- Yes. Yes they can.
- Very important in looping constructs, e.g. the instruction “after” the branch back.
- Implies leakage models need to be execution pipeline aware.



```
               .text
kernel:   mov  rY, #0
loop:     eor  rY, rF
          mov  rY, #0
          add  rG, #-1
          cmp  rG, #0
          bne  loop
done:     eor  rA, rB
          eor  rC, rD
          eor  rE, rF
          bx   lr
```

(e) SPECULATION/LOOP-0:

Selected Case Study 2: “Speculative” leakage.

- Can instructions which aren’t executed from an architectural perspective still cause leakage?
- Yes. Yes they can.
- Very important in looping constructs, e.g. the instruction “after” the branch back.
- Implies leakage models need to be execution pipeline aware.

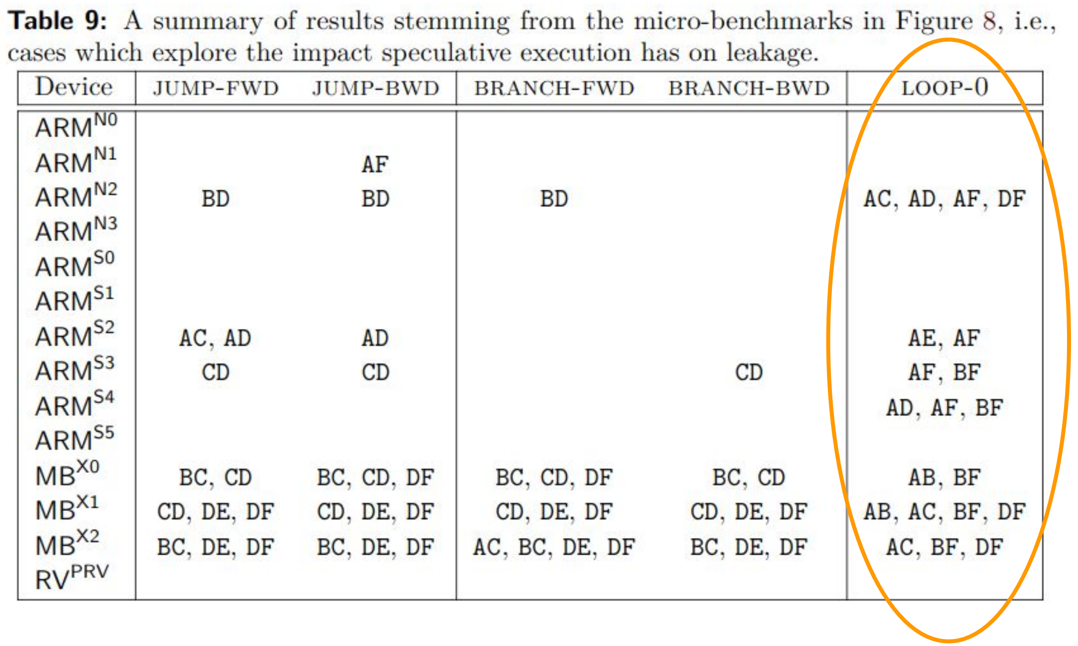
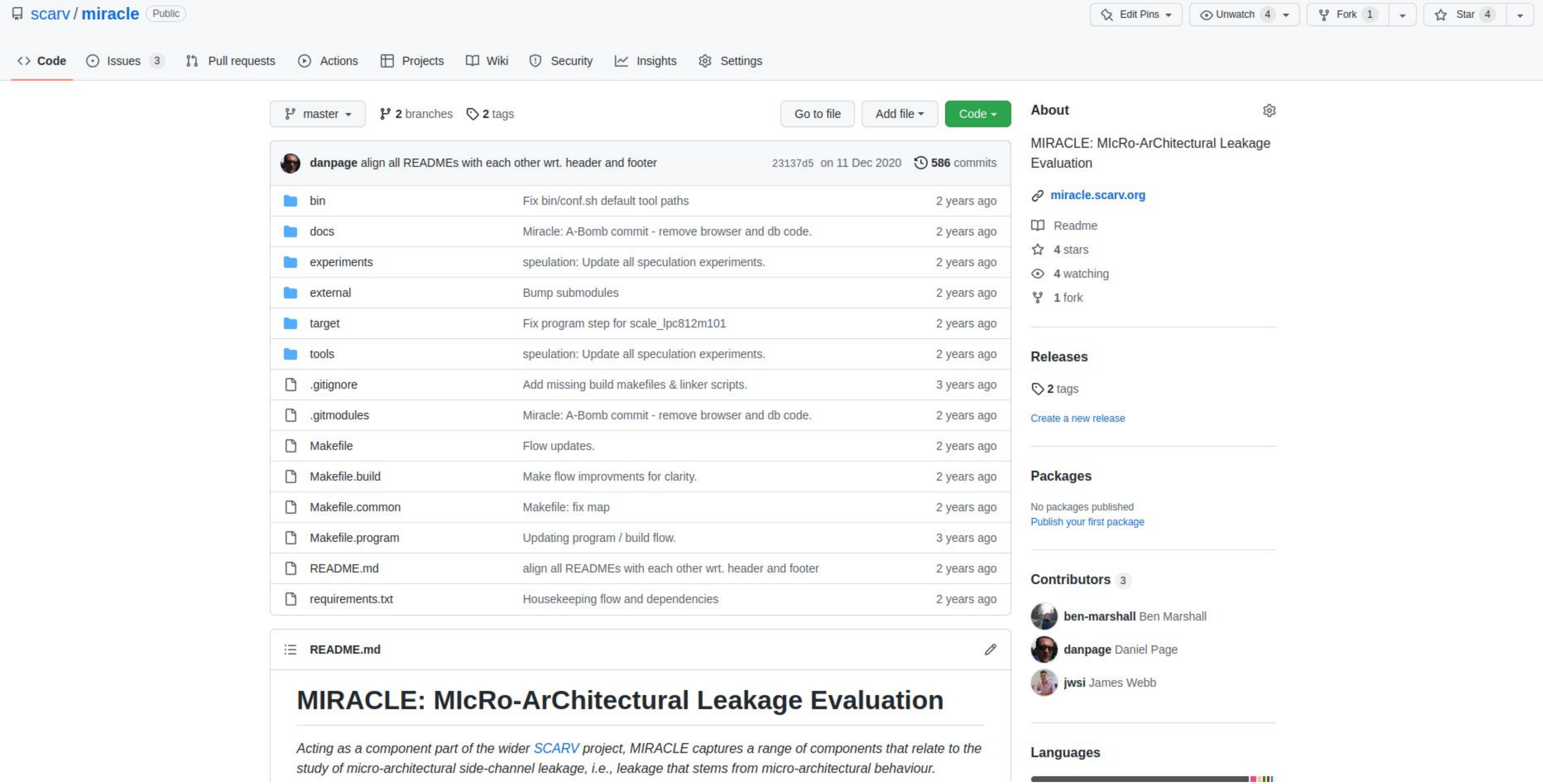


Table 9: A summary of results stemming from the micro-benchmarks in Figure 8, i.e., cases which explore the impact speculative execution has on leakage.

Device	JUMP-FWD	JUMP-BWD	BRANCH-FWD	BRANCH-BWD	LOOP-0
ARM^{N0}					
ARM^{N1}		AF			
ARM^{N2}	BD	BD	BD		AC, AD, AF, DF
ARM^{N3}					
ARM^{S0}					
ARM^{S1}					
ARM^{S2}	AC, AD	AD			AE, AF
ARM^{S3}	CD	CD		CD	AF, BF
ARM^{S4}					AD, AF, BF
ARM^{S5}					
MB^{X0}	BC, CD	BC, CD, DF	BC, CD, DF	BC, CD	AB, BF
MB^{X1}	CD, DE, DF	CD, DE, DF	CD, DE, DF	CD, DE, DF	AB, AC, BF, DF
MB^{X2}	BC, DE, DF	BC, DE, DF	AC, BC, DE, DF	BC, DE, DF	AC, BF, DF
RV^{PRV}					



scarv / miracle Public

Edit Pins · Unwatch 4 · Fork 1 · Star 4

Code · Issues 3 · Pull requests · Actions · Projects · Wiki · Security · Insights · Settings

master · 2 branches · 2 tags · Go to file · Add file · Code

danpage align all READMEs with each other wrt. header and footer · 23137d5 on 11 Dec 2020 · 586 commits

bin	Fix bin/conf.sh default tool paths	2 years ago
docs	Miracle: A-Bomb commit - remove browser and db code.	2 years ago
experiments	speulation: Update all speculation experiments.	2 years ago
external	Bump submodules	2 years ago
target	Fix program step for scale_lpc812m101	2 years ago
tools	speulation: Update all speculation experiments.	2 years ago
.gitignore	Add missing build makefiles & linker scripts.	3 years ago
.gitmodules	Miracle: A-Bomb commit - remove browser and db code.	2 years ago
Makefile	Flow updates.	2 years ago
Makefile.build	Make flow improvments for clarity.	2 years ago
Makefile.common	Makefile: fix map	2 years ago
Makefile.program	Updating program / build flow.	3 years ago
README.md	align all READMEs with each other wrt. header and footer	2 years ago
requirements.txt	Housekeeping flow and dependencies	2 years ago

README.md

MIRACLE: MIcRo-ArChitectural Leakage Evaluation

Acting as a component part of the wider SCARV project, MIRACLE captures a range of components that relate to the study of micro-architectural side-channel leakage, i.e., leakage that stems from micro-architectural behaviour.

About

MIRACLE: MIcRo-ArChitectural Leakage Evaluation

miracle.scarv.org

Readme

4 stars

4 watching

1 fork

Releases

2 tags

Create a new release

Packages

No packages published

Publish your first package

Contributors 3

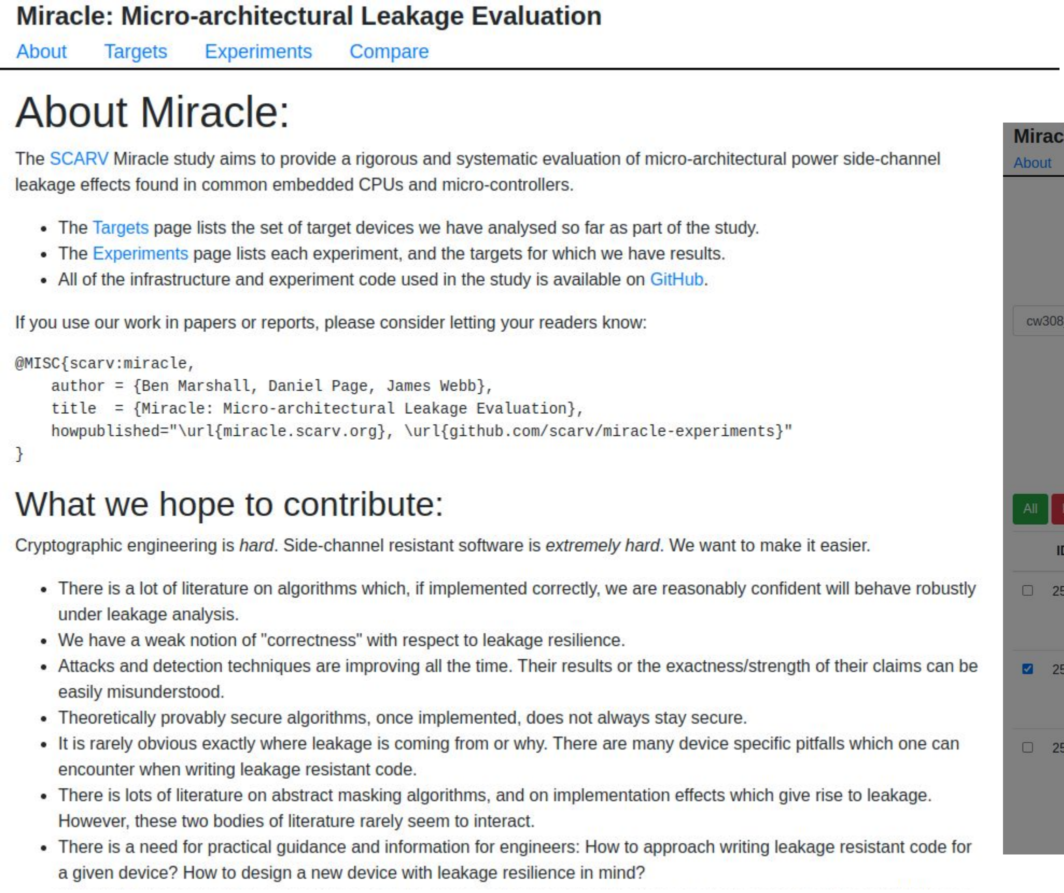
ben-marshall Ben Marshall

danpage Daniel Page

jwsi James Webb

Languages

University of BRISTOL · PQSHIELD



Miracle: Micro-architectural Leakage Evaluation

About Miracle:

The SCARV Miracle study aims to provide a rigorous and systematic evaluation of micro-architectural power side-channel leakage effects found in common embedded CPUs and micro-controllers.

- The Targets page lists the set of target devices we have analysed so far as part of the study.
- The Experiments page lists each experiment, and the targets for which we have results.
- All of the infrastructure and experiment code used in the study is available on GitHub.

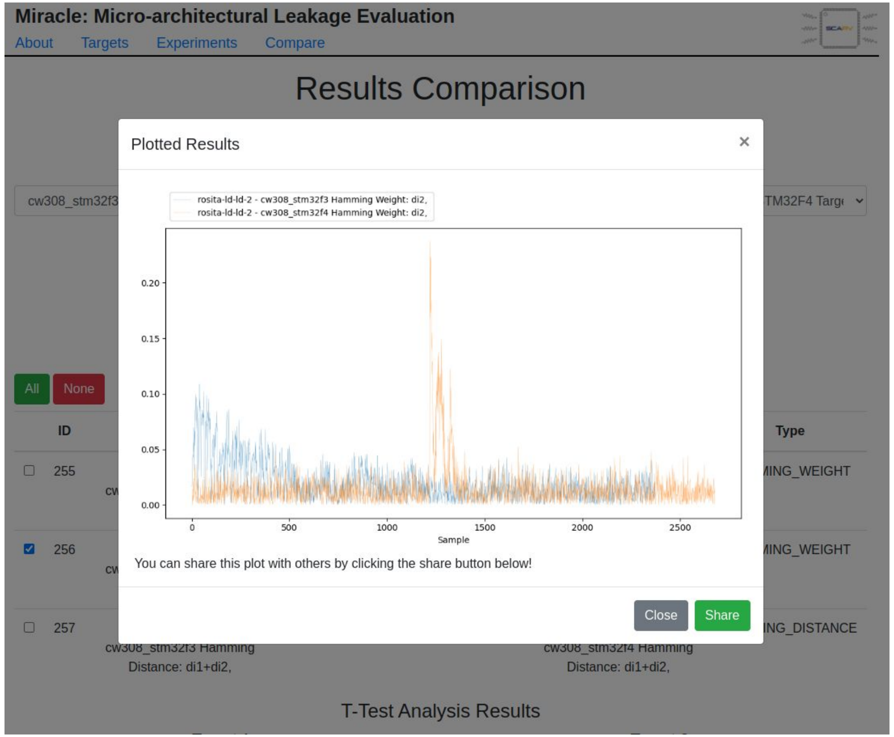
If you use our work in papers or reports, please consider letting your readers know:

```
@MISC{scarv:miracle,
    author = {Ben Marshall, Daniel Page, James Webb},
    title  = {Miracle: Micro-architectural Leakage Evaluation},
    howpublished="\url{miracle.scarv.org}, \url{github.com/scarv/miracle-experiments}"
}
```

What we hope to contribute:

Cryptographic engineering is *hard*. Side-channel resistant software is *extremely hard*. We want to make it easier.

- There is a lot of literature on algorithms which, if implemented correctly, we are reasonably confident will behave robustly under leakage analysis.
- We have a weak notion of "correctness" with respect to leakage resilience.
- Attacks and detection techniques are improving all the time. Their results or the exactness/strength of their claims can be easily misunderstood.
- Theoretically provably secure algorithms, once implemented, does not always stay secure.
- It is rarely obvious exactly where leakage is coming from or why. There are many device specific pitfalls which one can encounter when writing leakage resistant code.
- There is lots of literature on abstract masking algorithms, and on implementation effects which give rise to leakage. However, these two bodies of literature rarely seem to interact.
- There is a need for practical guidance and information for engineers: How to approach writing leakage resistant code for a given device? How to design a new device with leakage resilience in mind?



Miracle: Micro-architectural Leakage Evaluation

Results Comparison

Plotted Results

You can share this plot with others by clicking the share button below!

cw308_stm32f3 Hamming Distance: di1+di2,

cw308_stm32f4 Hamming Distance: di1+di2,

T-Test Analysis Results

An Aside: What We Could Have Done Better

- In the paper: We wanted to do more justice to past work but ran out of space. There is room for a wonderful SoK paper on this.
- A more quantitative assessment method: Just seeing if there is a leakage peak manually does not scale to many experiments!
- We went for breadth over depth. There are still many more sub-experiments one could perform to analyse certain effects further.
- We *really* wanted to include an x86 Micro-controller, but getting the thing to run code was a truly cursed process.

University of BRISTOL PQSHIELD

Conclusions: Device Naming

Not exactly news, but our results really emphasise this:

“*We evaluated our implementation on an:*”

- ARM Cortex-M3 🚫
- ARM Cortex-M3 with the ARM-v7m Architecture 🚫
- ARM Cortex-M3 with the ARM-v7m from ST-Micro 🚫
- ARM Cortex-M3 with the ARM-v7m from ST-Micro in an STM32F100RBT6B device 🚫
- **ARM Cortex-M3 *with X options*, in an STM32F100RBT6B device from ST-Micro, *hosted on a CW308 UFO board*.** 🙂

Conclusions: Evaluating Masked Implementations

Given that we found:

- The **“same” CPU core**, in two **different devices**, implemented by the **same manufacturer**…
- ***Had completely different leakage characteristics.***

What does this mean for evaluation of masked implementations?

Our recommendation:

- Try to evaluate on >1 “different” devices (guidance in the paper).
- You can claim a much more robust implementation if it works in both places.
- **Standards bodies must not pick only one standard evaluation platform.**

Conclusions: Building Quality Leakage Models

How can we better validate leakage models or tools?

- **Validate across multiple devices!**
- We hope our set of micro-benchmarks can serve as “unit tests” for per-device leakage models.
- Testing on one cipher or implementation might not be enough.
- Build machine-readable descriptions of micro-architecture for devices.
- These descriptions are inputs to more generic device leakage modelling engines.
- E.g. “Coco-alma”, or “Masking in fine-grained leakage models” are great examples of this direction.

What Next?

- There's always more devices to test…
- There's always more micro-architectural effects to characterise…
- This was not an attack paper, or a countermeasure paper.
  - How can these effects be exploited?
- Which types of leakage are "more" dangerous?
- How to characterise leakage in the UN-Core part of the system?

Bigger Picture:

- Community wide DSL for describing micro-architectures & their leakage characteristics as a common input to everyone's leakage tooling. Like SMT2 for the sat-solving community.

University of BRISTOL PQSHIELD

Thank You For Listening

I Hope You Have Questions?