Explainable Leakage Assessments

Elisabeth Oswald

D!ARC, University of Klagenfurt,

School of Computer Science, University of Birmingham

Roadmap

Context: Hardware enables "better" security

"Secure" integrated circuits are used as banking cards, are also hidden as SIMs in mobile phones, embedded in laptops, USB sticks, and all sorts of IoT devices.

Software runs on this hardware, implementing all sorts of cryptographic functionality.

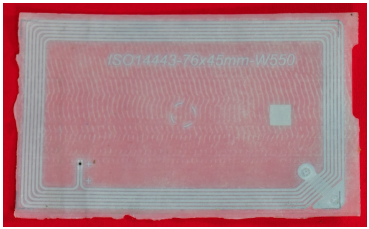


Figure: Contact-less smart card, Z22, CC BY-SA 4.0 `https://creativecommons.org/licenses/by-sa/4.0`, via Wikimedia Commons

Hardware modules are a "root of trust" in many modern systems.
They store **cryptographic keys** that **must remain secret even during computations**.

Context: Side Channels

There are many side channels out there:

- side channels directly linked to physics: power, EM, timing
- side channels linked to micro-architectural properties: cache, timing
- side channels linked to high level implementation choices: error messages

> Side channel attacks can reveal key and message information efficiently: they typically lead to an exponential loss in security. Assessments mandatory for some product groups before entering the market.

Roadmap

Context: Evaluating Crypto Implementations
- Implementations as a fundamental security anchor
- Formal Security Evaluations

Next Generation Leakage Assessment: Assurance and Explainability
- A novel framework for explainable assessments
- Micro-architectural Simulators

Context: Leakage Assessment

In-house testing: whatever provably secure strategy is used, “in-house” assessments must be carried out.

High stakes crypto devices undergo an evaluation process to achieve some certificate before they are released to the market: the most “popular” evaluation schemes are

- NIST FIPS 140-3

and

- Common Criteria

(both schemes have corresponding ISO/IEC standards).

Conformance testing: “Does the device leak information via side channels?”
Penetration testing: “Can a (powerful) side channel adversary succeed?”

Desirable Properties of Evaluation Methods

Low effort: trace and computational complexity matter.

Reasoning about “classes of attacks’: e.g. differential input attacks, vs single trace attacks.

Confidence about “trace complexity”: could an attack that fails with n observations, perhaps succeed with $2 \cdot n$ observations?

Explaining what “causes the leakage”: micro-architectural effects are a common cause of complicated leakage characteristics.

WHAT IS LEAKAGE?

Informally leakage is understood as unintended, additional information $\mathbf{L}$ about a secret value (message or key). I focus on "key recovery attacks".

Observed leakage usually is of the form $L = f(X, K) + R$
($X \ldots$ message space, $K \ldots$ key space, R independent term).

"Leakage" implies:

Change in key distribution – attack based assessment: $\exists k, l$ for which

$$\Pr[K = k] \neq \Pr[K = k | \mathbf{L} = l]$$

Non-zero Dependence – quantification based assessment: $MI(\mathbf{K}, \mathbf{L}) > 0$

Change in leakage distribution – detection test $\exists k_i, k_j, l$ such that

$$\Pr[\mathbf{L} = l | \mathbf{K} = k_i] \neq \Pr[\mathbf{L} = l | \mathbf{K} = k_j]$$

Assessment via attack: $\Pr[K = k] \neq \Pr[K = k|\mathbf{L}]$

- Must perform a full attack, then compute scores/rank.
- Attack requires to be (more or less) specific about approximation of f (device leakage function, device architecture, target intermediate variable).

We only learn something regarding the specific attack configuration.

Often not enough to “explain why”. Limited coverage of entire attack surface.

ASSESSMENT VIA QUANTIFICATION: $MI(\mathbf{K}, \mathbf{L}) > 0$

Quantify the dependency between the key or a known (key dependent) value, and the observed side channel data, e.g. by computing the MI, SNR, NICV:

- Under certain assumptions we can reduce the effort by computing quantity just for $K = sk$
- Can be akin to "known key attack", e.g. $\mathrm{Corr}(HW(S(x + sk)), L)$, but this implies more specificity
- Some quantities do not require a device leakage model, or an intermediate value, we can compute for e.g. tuple (x, k)

Can be less specific than an attack. Still limited coverage of attack surface. Unclear if exploitable.

Specificity prevents the efficient detection of arbitrary leaks

Specific attacks or quantities for intermediate values are always limited to this single attack vector or intermediate value or adversarial leakage model.

But does that matter? If an evaluation aims to "quickly reject very vulnerable" devices, then finding a single leak efficiently is enough.

- Finding and patching "a" leak then enables passing an evaluation.
- Finding all leaks clearly better for long term security, aka high assurance in evaluation outcome.

Thus specific approaches are ill suited to find all leaks.

If an evaluation aims to reveal many (in the ideal case all) leaks, then we need an approach to **efficiently identify** many (all) leaks.

MOVING TOWARDS A MORE GENERAL APPROACH

We can avoid specificity by checking properties of the observed leakage distribution rather than the key distribution:

$$\Pr[\mathbf{L}|K = k_i] \neq \Pr[\mathbf{L}|K = k_j] \text{ or } \Pr[\mathbf{L}|X = x_i] \neq \Pr[\mathbf{L}|X = x_j].$$

Practice:

- We sample leakage sets for specific choices of key/input.
- And check if the sets can be distinguished statistically: if so, they must depend on key/input.
- If not distinguishable we conclude that we do not have enough evidence to distinguish them.
- Testing many pairs is inefficient: fix one value, and vary the other value, is more efficient.

Non-specific Assessment

If the distribution of $\mathbf{L}$ is known then we can use tailored statistics: this what **TVLA:** (Test Vector Leakage Assessment) suggests.

- Classical plaintext-based TVLA: assumption that L_i is Gaussian; thus we test if $E[\mathbf{L}|X = x_i] \neq E[\mathbf{L}|X]$ (**fixed-vs-random test)**

TVLA reveals data dependent points in side channel traces for a fixed key: not pure key dependency.

With trace processing, one can "shift" differences in higher central moments to a difference in the mean.

Can not determine all leaks: only what shows in central moments, interactions between bits not necessarily exposed.

Exploitability unclear, explainability poor.

Roadmap

Context: Evaluating Crypto Implementations
- Implementations as a fundamental security anchor
- Formal Security Evaluations

Next Generation Leakage Assessment: Assurance and Explainability
- A novel framework for explainable assessments
- Micro-architectural Simulators

A Novel Framework for Explainable Leakage Assessment

We propose a new approach to achieve explainability alongside cost efficiency and long-term security.

The approach is based on three key observations:

- We suggest checking properties of the leakage distribution as a function of a set varying key chunks $K = \{K_i\}$.
- We suggest to utilise **a statistical model building approach** instead of a moment-based statistic to test for evidence of leakage
- We make suggestions for the notions of "exploitable leakage" via and "explainable leakage" (via size of key guess and a link to certificational attacks)

These ideas have been developed in [4], [3], [1], and lead to [2].

EXPLOITABLE/EXPLAINABLE LEAKAGE

“Exploitable leakage”: leakage that leads to a differential attack vector that requires guessing of less than B key chunks. (B needs to be set by the evaluation scheme).

“Explainable Leakage Detection”: a method that enables to directly derive a “certificational” attack vector. This is an attack vector that succeeds, but it is not necessarily the most trace efficient attack vector.

Statistical Model Building as Detection

We use nested regression models (L_f, L_r, notice I am overloading variables) as a way to express key dependency:

$$L_f(K) = \sum_j \beta_j u_j(K),\ j \in J$$

$$L_r(K) = \sum_j \beta_j u_j(K),\ j \in J', \text{with}\, J' \subset J$$

The full model includes all combinations of terms. The reduced model includes only a subset of terms.

The coefficients β_j for the two models are estimated from the available side channel observations. A statistical test decides if the two models differ significantly

If there is enough evidence to distinguish the models, then the "extra" coefficients (=key chunks) matter.

Statistical Model Building as Detection - Toy Example

Suppose that we have only two key chunks K_1 and K_2.
Then the full model is given as:

$$L_f(K) = \beta_0 + \beta_1 K_1 + \beta_2 K_2 + \beta_3 K_1 \cdot K_2.$$

The naive model would be

$$L_0 = \beta_0.$$

We say that there is key leakage on some observed data, if L_0 does not explain the observed data as well as L_f.

- Analysis will reveal trace points are that key dependent

A Detection "Framework" for Leakage

Building more fine grained restricted models leads to a framework as follows:

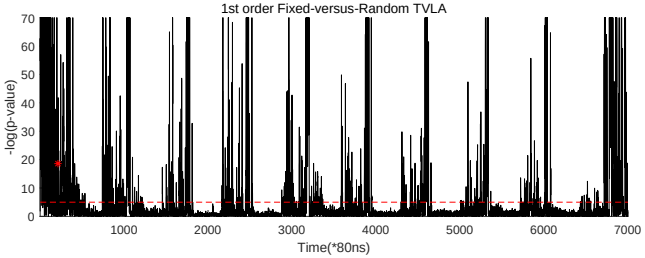
1. Leakage detection based on testing the full vs. the naive model ("all or nothing test"). Same interface as TVLA.
2. Identifying exploitable leakage by building/testing restricted models with degree lower than a given bound ("degree analysis)".
3. Explainable assessment: for exploitable leakage, build the smallest reduced model that is statistically equivalent to the full model, and then turn this into a "certificational" template attack.

Example: Boolean Masking on a 32-bit Processor

TVLA (top right).

We configure the full model to contain all 16 bytes of an AES key and run our non-specific detection (bottom right).

Model based detection discovers more key dependent points in first round than TVLA (as expected).

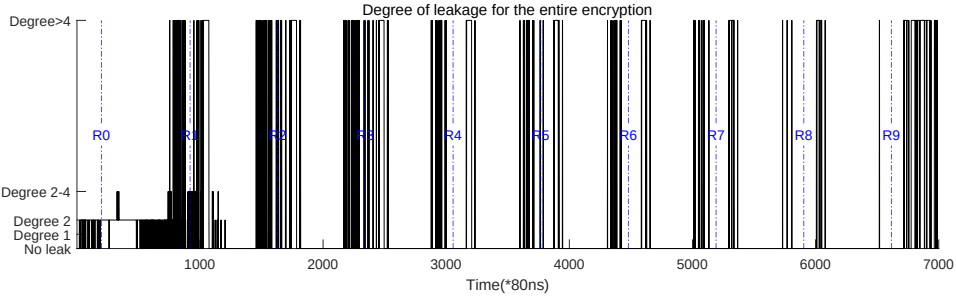


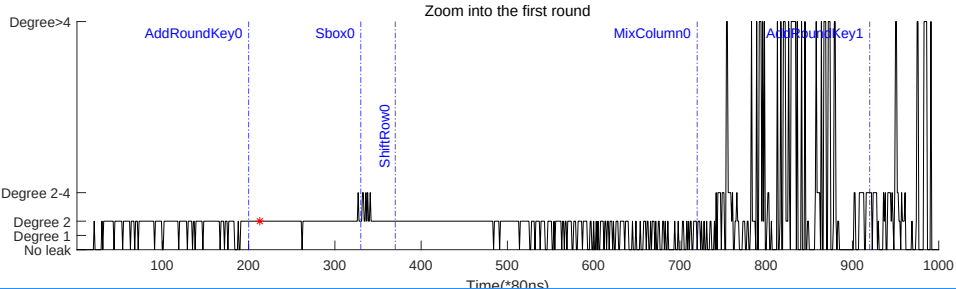
Example: Boolean Masking on 32-Bit Processor

But which of the identified points offer **exploitable key leakage**?

We next configure our reduced model to contain 1, 2, 3, or 4 key bytes and compare with the full model which is based on 16 key bytes.

EXAMPLE: BOOLEAN MASKING ON 32-BIT PROCESSOR





Example: Boolean Masking on 32-Bit Processor

We pick one of the low degree points (marked with a red asterisk in the previous figure).

It only depends on at most 2 key bytes: but which ones?

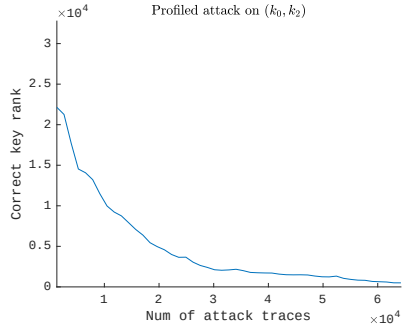
We find out by configuring the respective models, and check their test results (see Table on the right).

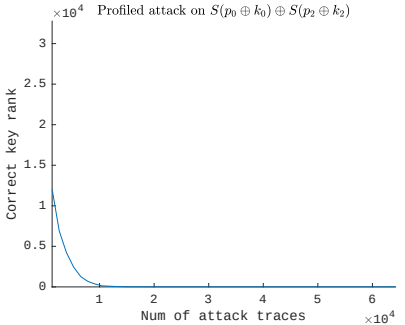
Key bytes	$-log_{10}(p - value)$
(k_0, k_1)	14.93
(k_0, k_2)	max
(k_1, k_2)	max
(k_0, k_3)	140.26
(k_1, k_3)	93.12
(k_2, k_3)	max

Example: Boolean Masking on 32-Bit Processor

A certificational attack vector can be established easily: build templates for the identified key pairs, and use them in a profiled attack.

The left hand side shows a certificational attack for the key pair (k_0, k_2). The right hand side shows an attack vector for the same two key bytes but with knowledge of the implementation.





Roadmap

Context: Evaluating Crypto Implementations
- Implementations as a fundamental security anchor
- Formal Security Evaluations

Next Generation Leakage Assessment: Assurance and Explainability
- A novel framework for explainable assessments
- Micro-architectural Simulators

Leakage due to Micro-architectural choices

Processor architecture: high level interface; aligned with a programming language (aka Assembly).

E.g. ARM Cortex M family, has a number of architecture generations, v8 is most current.

Standardised for a processor family: comes with guarantees about behaviour, e.g. execution cycles and dependencies.

But **how** this behaviour is exactly realised in hardware is left up to any specific IP licence holder (e.g. STM, NXP, etc.)

Effect on security of (provably) secure masking scheme is devastating.

LEAKAGE DUE TO MICRO-ARCHITECTURAL CHOICES

“If it’s provably secure then it’s probably not.” (Lars Knudsen)

Proofs are based on abstractions of reality, e.g. “only computation leaks information” (but what if we don’t “know” all the computations that are taking place).

Proofs are often for high level descriptions of an algorithm: even Assembly language is a high-level description.

Practice: Typically we must hand tune implementations of provably secure schemes to make them secure in practice.

HELPING HAND FOR DEVELOPERS

Make tools that take off the burden of developers by either automating tasks or supporting difficult tasks.

Leakage simulators: combination of instruction set emulator, processor model, and device leakage model with the aim of rapid testing/problem finding and fixing.

Long history: modern ones are MAPS, ELMO and derivatives (aka ELMO*, ROSITA), μELMO.

Shortcomings: either built on "abstract" processor description (MAPS, derived from public ARM IP), or, built with implicit micro-architectural modelling (ELMO and derivatives).

μELMO is the only simulator that has been built on reverse engineering of a modern M3 core.

Leakage Simulators to Aid Developers

μElmo: Simulator produces execution traces, which can either be mapped to leakage traces, or be used to directly interface with proving tools.

Reverse engineering based on **nested model** building: i.e. the technique we also use for explainable assessments:

- We check existence of interactions between components/variables using this technique.
- Simulation is hence based on an instruction-set emulator that we enhanced with components that are "leakage equivalent" to physical device.

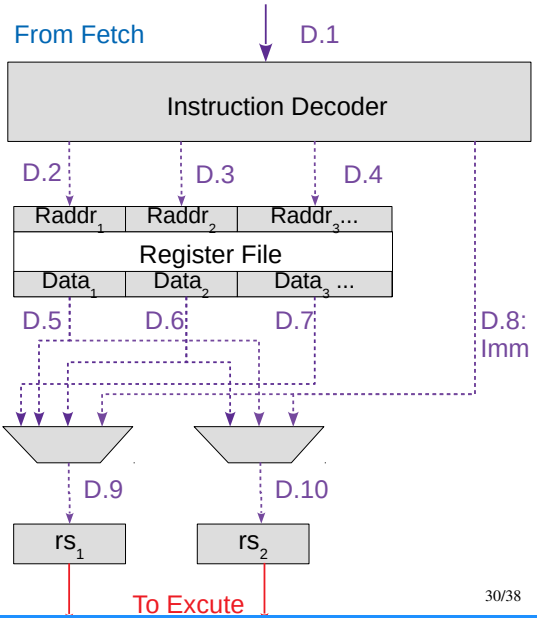
Example of Leakage Equivalent Component

Assume read/write ports in “typical decoder architecture”.

Unclear how many there are, and how operands are assigned to them.

Formulate hypotheses as nested models, and check which ones “fit best”.

We did this for a concrete M3 (v8 architecture): we model fetch, decode, execute, and the AMBA bus transfer (to some extent).

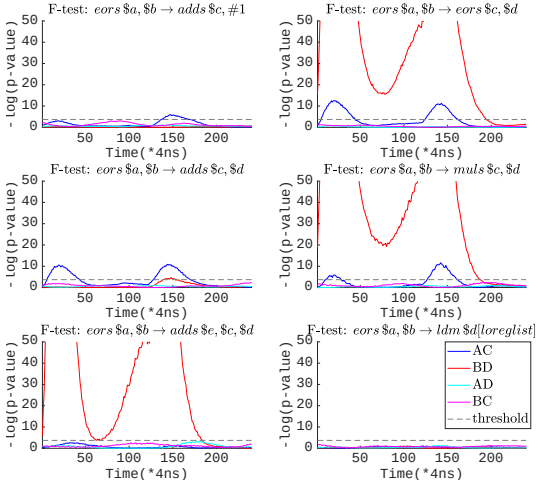


Example of Port Analysis

Which variables are loaded on which ports, and do they produce joint leakage?

Aka: do variables interact?

Configure sequences of instructions (different types), and test using nested models.



Example of Port Analysis

Instruction types are grouped.

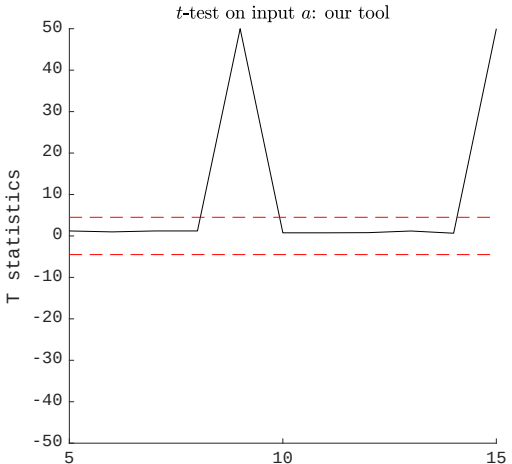
Can be associated uniquely with micro-architectural ports/registers.

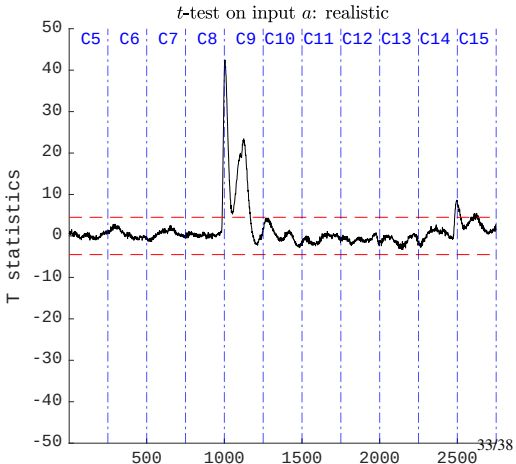
Enables to include transition leakage effects in simulation.

Group		Assembler	Encoding				Decoding			Executing	
	Operand		Type	Rd	Rm	Rn	Port 1	Port 2	Port 3	RS_1	RS_2
ALU	0	MOVS Rd, #<imm8>	I	10-8	-	-	Rd	-	-	-	-
	1	INSTR-a Rd, Rm{, #<imm3/5>}	II	2-0	5-3	-	Rd	Rm	-	Rm	-
		INSTR-b Rd, Rm{, #<imm3/5>}	II	2-0	5-3	-	Rd	Rm	-	-	Rm
		INSTR Rd, Rd, #<imm8>	I	10-8	-	-	Rd	-	-	Rd	-
		INSTR Rd, Rm	III	7,2-0	6-3	-	Rd	Rm	-	-	Rm
	2	INSTR Rd, Rm	II	2-0	5-3	-	Rd	Rm	-	Rd	Rm
		INSTR Rd, Rn, Rm	IV	2-0	8-6	5-3	Rd	Rm	Rn	Rn	Rm
		ADD Rdn, Rm	III	7,2-0	6-3	-	Rdn	Rm	-	Rdn	Rm
		MUL Rdm, Rn	IV	2-0	-	5-3	Rdm	Rn	-	Rdm	Rn
LOAD	Imm	LDR(H/B) Rd, [Rn, #<imm>]	IV	2-0	-	5-3	Rd	Rn	-	Rn	-
	Reg	LDR(H/B) Rd, [Rn, Rm]	IV	2-0	8-6	5-3	Rd	Rn	Rm	Rn	Rm
	Multiple	LDM Rn!, <loreglist>	V	-	-	10-8	-	-	Rn	Rn	-
	Pop	POP <loreglist>		-	-	-	-	-	-	C	-
STORE	Imm	STR(H/B) Rd, [Rn, #<imm>]	IV	2-0	-	5-3	Rd	Rn	-	Rn	Rd
	Reg	STR Rd, [Rn, Rm]	IV	2-0	8-6	5-3	Rd	Rn	Rm	Rn->Rd	Rm
	Multiple	STM Rn!, <loreglist>	V	-	-	10-8	-	-	Rn	Rn	-
	Push	PUSH <loreglist>		-	-	-	-	-	-	C	-

Example of Leakage Analysis

μElmo leakage traces exhibit the same behaviour (micro-architectural leakage) than real traces sampled from that specific M3 core. Picture below: sequence related to multiplication with 2 shares.





SEAL API

Integrates leakage simulation functionality into almost any emulator:

Enables extraction of execution traces, mapping to leakage traces, or to traces compatible with proving tools.

Enables integration with pre-silicon descriptions.

`https://github.com/sca-research/uELMO/` (SEAL enabled version of μElmo).
`https://github.com/sca-research/SealIbexEmulator` (IBEX leakage simulator enabled by SEAL API)

Much of what we developed is open source. But badly documented.

Ongoing Work - PRESEC

UKRI funded project on **pre-silicon leakage analysis** across University of Birmingham and Bristol. Considerable industrial focus: integration of leakage simulation into pre-silicon design/configuration flow; aim to automatically generate accurate leakage simulator for arbitrary processor cores.

Collaborative work with Codasip, supported also by NCSC, aligned with the UK RISE initiative.

We are recruiting for: researchers (also potentially before completion of a PhD) with background in computer architecture, hardware design, low-level software development, etc.

Thank You
For Your Attention

This research in this talk was supported in part by the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation program (grant agreement no. 725042).

If you are interested in PRESEC then please contact me via email. If you are interested in joining the editorial board of the Journal of Cryptographic Engineering then please contact me via email.

REFERENCES I

[1] S. Gao and E. Oswald.
A novel completeness test for leakage models and its application to side channel attacks and responsibly engineered simulators.
In EUROCRYPT, 2022.

[2] S. Gao and E. Oswald.
A Novel Framework for Explainable Leakage Assessment.
In EUROCRYPT 2024, 2024.

[3] S. Gao, E. Oswald, and D. Page.
Towards micro-architectural leakage simulators: Reverse engineering micro-architectural leakage features is practical.
In EUROCRYPT, 2022.

REFERENCES II

[4] D. McCann, E. Oswald, and C. Whitnall.
Towards practical tools for side channel aware software engineering: 'grey box' modelling for instruction leakages.
In USENIX Security 2017.