Radboud University

AI and Side-channel analysis: Lessons learned so far

International Winter School on Microarchitectural Security 2022
December 7, 2022

Lejla Batina

Institute for Computing and Information Sciences
Radboud University
lejla@cs.ru.nl

Outline

Intro to side-channel analysis

Side-channel Analysis (SCA) Attacks

SCA Countermeasures

Leakage evaluation

SCA and AI

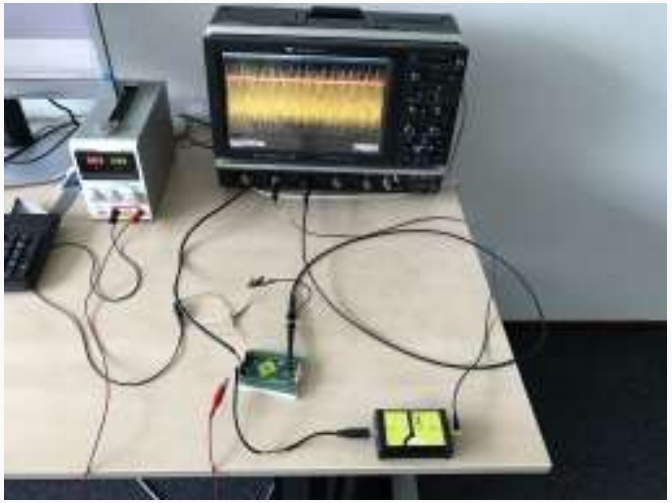
Leakage emulation and ROSITA

Screen Gleaning

Intro to side-channel analysis

Known challenge: embedded crypto devices

Implementation attacks



Relevance

November 13, 2019

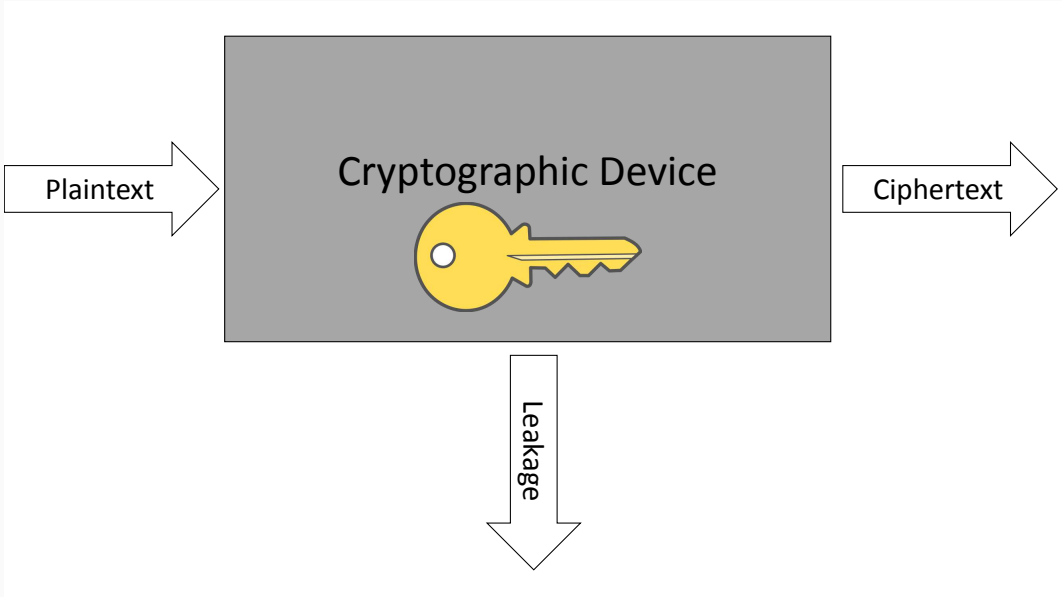
October 3, 2019

May 28, 2020

January 7, 2021

Side-channel Analysis (SCA) Attacks

Greybox/Whitebox scenario



Greybox = SCA adversary in the wild:

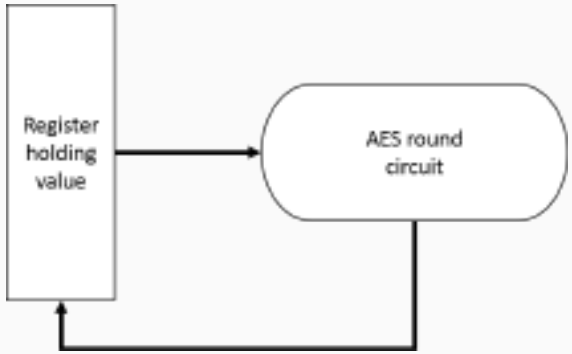
- Crypto is **implemented on a real device** such as a microcontroller, FPGA, ASIC
- Adversary can measure and process *physical quantities* in the device's vicinity
- Adversary's goal: secret key or plaintext recovery by observing plaintext/ciphertext pairs **and a side channel**

Whitebox = Security evaluator:

- Algorithms and implementation details are (partially) known
- Adversary's goal: secret key or plaintext recovery by observing plaintext/ciphertext pairs while trying all known attacks, including profiling

Power side-channel: Modeling the leakage

- The Hamming distance model counts the number of $0 \rightarrow 1$ and $1 \rightarrow 0$ transitions
- Example 1: Assume a hardware register R storing the result of an AES round. The register initially contains value v_0 and gets overwritten with value v_1

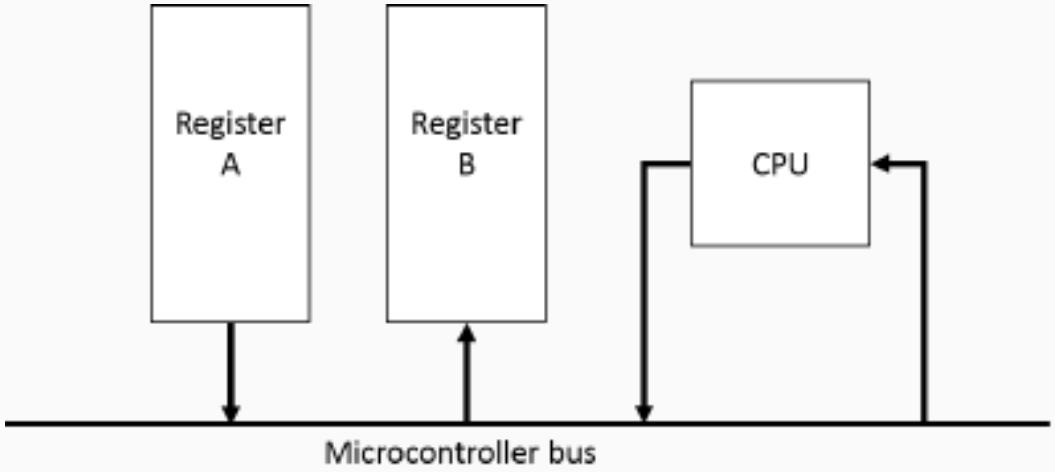


- The power consumption because of the register transition $v_0 \rightarrow v_1$ is related to the number of bit flips that occurred
- Thus it can be modeled as $\text{HammingDistance}(v_0, v_1) = \text{HammingWeight}(v_0 \oplus v_1)$

Power side-channel: Modeling the leakage

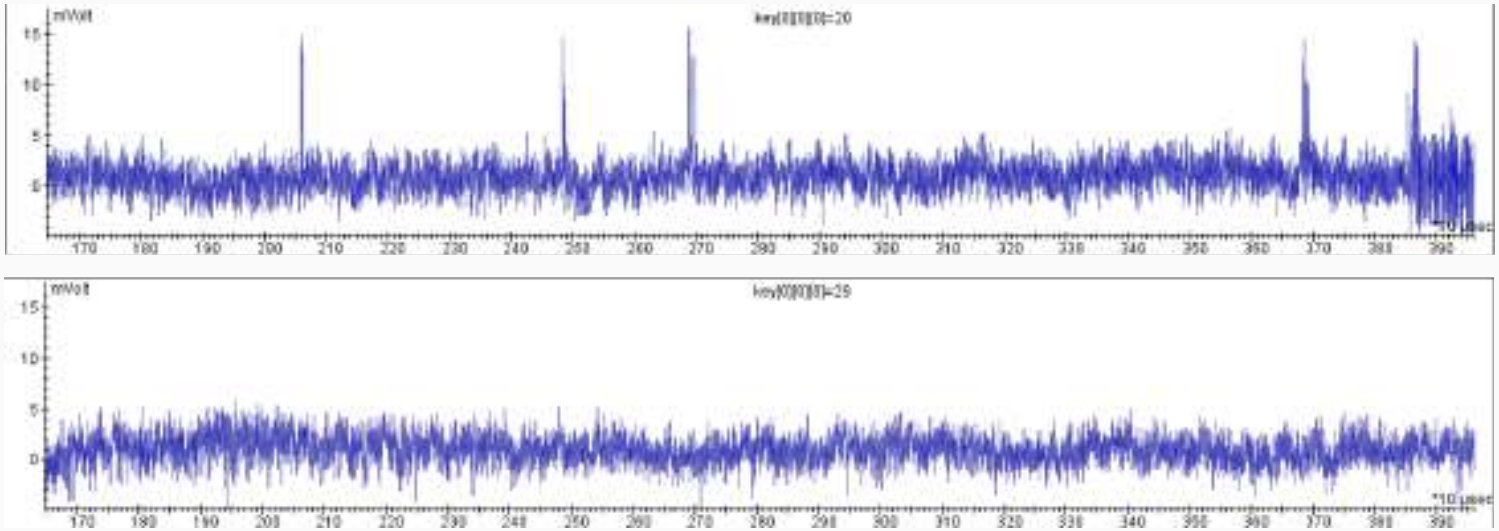
- Example 2: In a microcontroller, assume register A with value v_0 and an assembly instruction that moves the contents of register A to register B

```
mov rB, rA
```



- In general-purpose processors the instruction will transfer value v_0 from register A to B via the CPU, using the bus
- Often the bus is a very leaky component and also precharged to all bits to zeros (or all to 1) i.e. busInitialValue
- The power consumption of the assembly instruction can be modeled as $\text{HammingDistance}(\text{busInitialValue}, v_0) = \text{HammingWeight}(v_0 \oplus 0) = \text{HW}(v_0)$

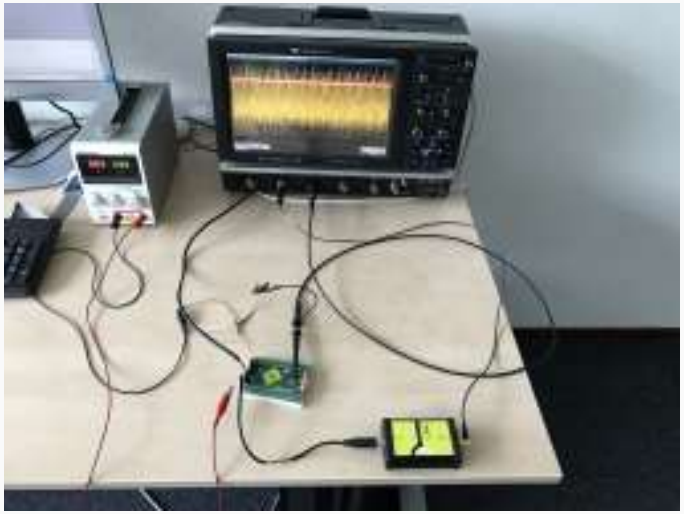
Differential Power Analysis (DPA)



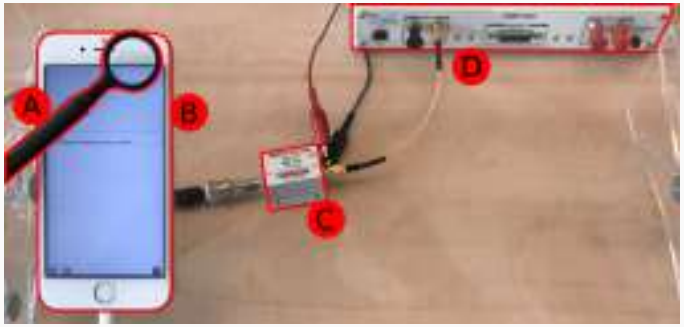
- The most popular side-channel attack
- Aims at recovering the secret key by using a large number of power measurements (traces)
- Nowadays often combined/replaced with a leakage evaluation methodology such as TVLA

Actual setups

DPA setup with ARM CortexM4

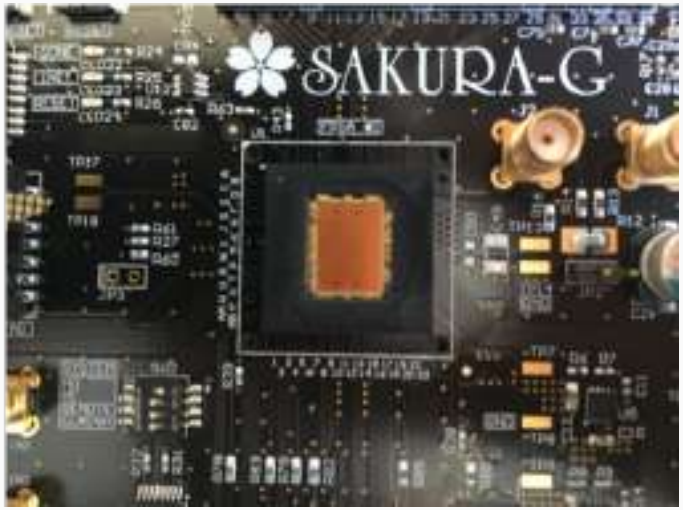


Tempest



FA setup

FPGA board for SCA



SCA Countermeasures

Main idea

Goal: break the link between the actual data and power consumption

- Masking: power consumption remains dependent on the data on which computation is performed but not the actual data
- Hiding: power consumption is independent of the intermediate values and of the operations

Masking

Boolean masking: a dth-order (Boolean) masking scheme splits an internal sensitive value v into $d+1$ shares $(v_0, v_1, ..., v_d)$, as follows:

$$v = v_0 \oplus v_1 \oplus \cdots \oplus v_d$$

Probing-secure scheme. We refer to a scheme that uses certain families of shares as $d-$probing-secure iff any set of at most d intermediate variables is independent from the sensitive values.

Consequently, the leakage of up to d values does not disclose any information to the attacker.

Masking with 2 shares

- $X = X_1 \oplus X_2$
- The leakage $L(X) = HW(X_1, X_2)$ depends on two variables.
- It does not reveal info on the value of X when a DPA is performed, in theory

Masking in practice: unintended interactions between values in the processor cause leakage in 1st order (caused often by transitional effects and glitches).

If a program that processes a secret value X contains two consecutive instructions (the first uses X_1 and the second uses X_2), then the transitional effect of changing the contents of the bus leaks the Hamming distance between X_1 and X_2.

Leakage evaluation

Independent Leakage Assumption (ILA) [RSVKF11]

Independent computations give rise to independent leakage.

Practically: The underlying assumption of this model is that the adversary can only observe a single intermediate value with every probe used.

As a consequence: All masks (shares) need to be processed independently

Physical side-effects when implementing masking, such as glitches and distance-based leakages, violate ILA in practice.

Test Vector Leakage Assessment (TVLA)

- Leakage assessment of a device is very important for the semiconductor and the security evaluation industries
- Number of attacks to check the device's resistance against keeps on growing
- Various attackers' models possible but security evaluation often goes for the strongest adversary
- It is using Welch's t-test to differentiate between two sets of measurements, one with fixed inputs and the other with random inputs
- Far from perfect, false positive and negatives are possible
- Leakage from combining multiple points is not detected

SCA and AI

AI for Side-channel Analysis (SCA): Why

- Machine learning for SCA was a natural direction:
  - PCA to assist profiling/template attacks (dimensionality reduction)
  - PCA for pre-processing measurement traces
  - Machine Learning (ML)-based SCA distingushers
- Deep learning in SCA:
  - neural nets for profiling attacks
  - defeating countermeasures (e.g. skip the alignment phase)
  - leakage assessment/simulators
  - TEMPEST-like techniques e.g. screen gleaning

AI for SCA: When

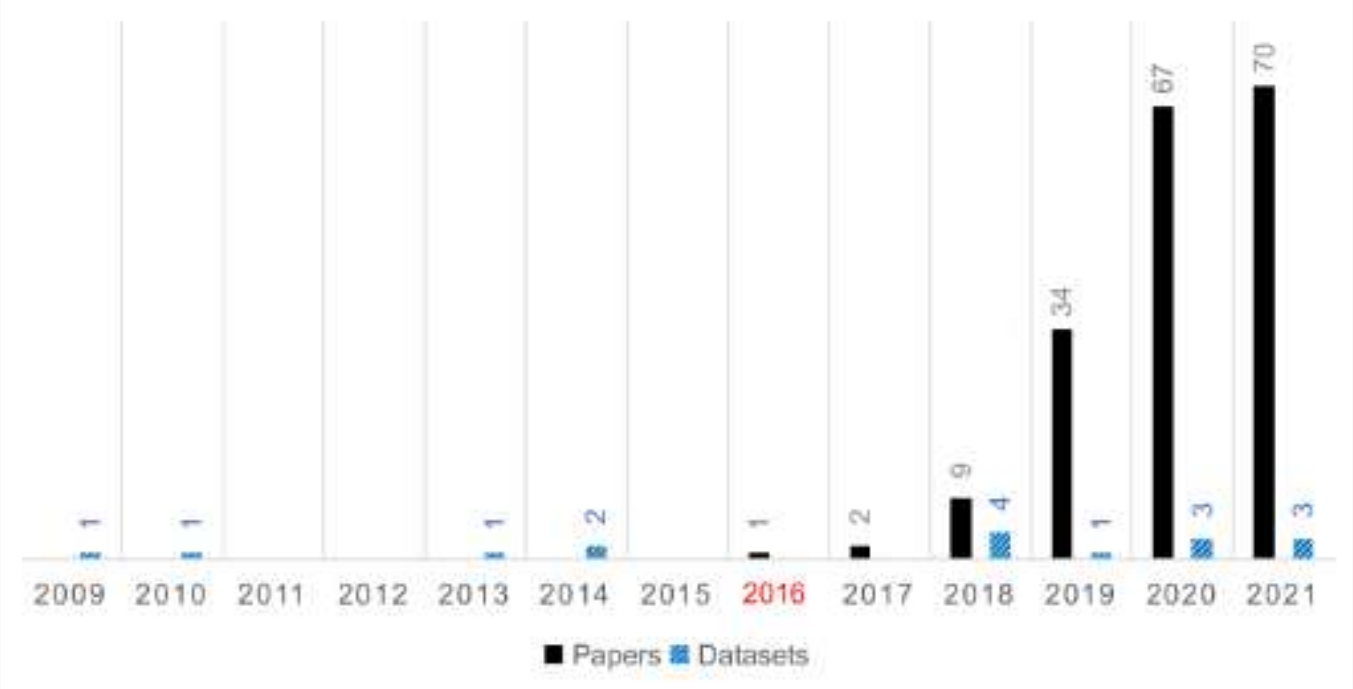


Figure: Deep learning papers and datasets.

S. Picek, G. Perin, L. Mariot, L. Wu and L. Batina, SoK: Deep Learning-based Physical Side-channel Analysis, `https://eprint.iacr.org/2021/1092`, accepted at ACM Computing Surveys, 2022.

Leakage emulation and Rosita

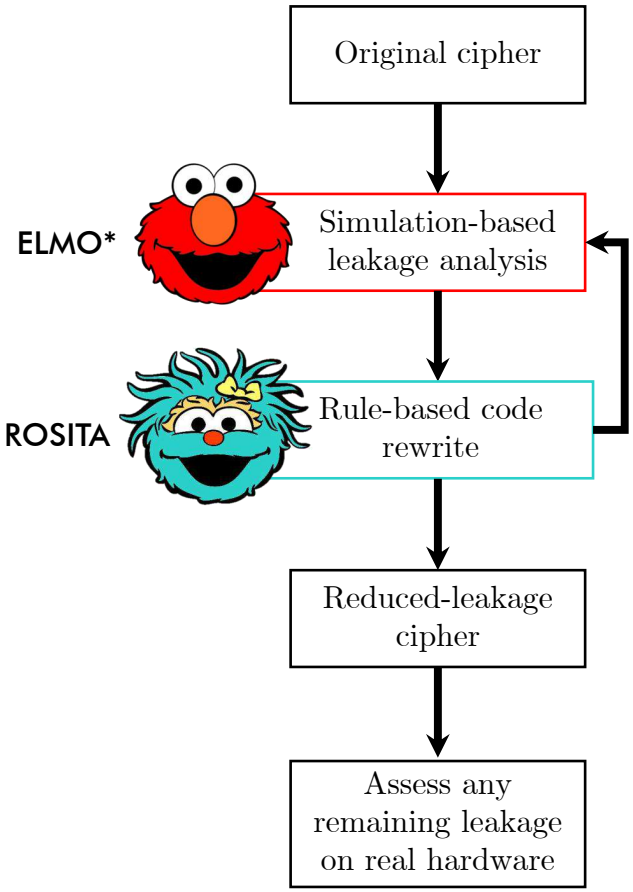
Motivation

- Crypto implementations go through multiple cycles of leakage evaluation
- This procedure requires a lot of expertise and it is time consuming
- It can be improved by using leakage emulators
- Recently several emulators of the power consumption (or EM leakage) such as ELMO were proposed
- The results of such emulations are combined with standard statistical tests such as t-test

Previous work on leakage simulators and automatized approaches to SCA

- The most accurate simulators use SPICE
- Main difference is in emulating at the source code level (platform-independent) and at machine instruction level (with a specific CPU in mind)
- ELMO [MOW17] is an instruction level emulator that uses power consumption traces from real experiments to make better estimates
- Previous approaches can be divided into:
  - simulation-based e.g. SILK (Veshchikov)
  - using code analysis (Barthe et al.)
  - hardware-assisted (implementing masking within a processor)

Rosita overview



M. A. Shelton, N. Samwel, L. Batina, F. Regazzoni, M. Wagner, Y. Yarom: Rosita: Towards Automatic Elimination of Power-Analysis Leakage in Ciphers. NDSS 2021.

Elmo

- ELMO models 21 instructions divided into five groups
- Power traces are collected while the processor executes instruction triplets
- Each operand is compared to the corresponding operand of the preceding instruction
- Each trace is processed to select a PoI as a representative of the trace
- ELMO then performs a linear regression on the data collected in the traces to find the coefficients for the model
- Power consumption is modeled as linear combinations of bit values or bit changes
- ELMO is emulating leakage for the ARM Cortex-M0

Elmo*

- ELMO* is using the same STM32F0302 evaluation board with an ARM Cortex-M0 and performs TVLA
- ELMO* also looks at combinations of bits across the two operands of an instruction
- Introduces the concept of dominating instructions i.e. instruction pairs that interact via hidden storage within the processor
- ELMO* finds which of the components of the model are causing the leak

Dominating instructions

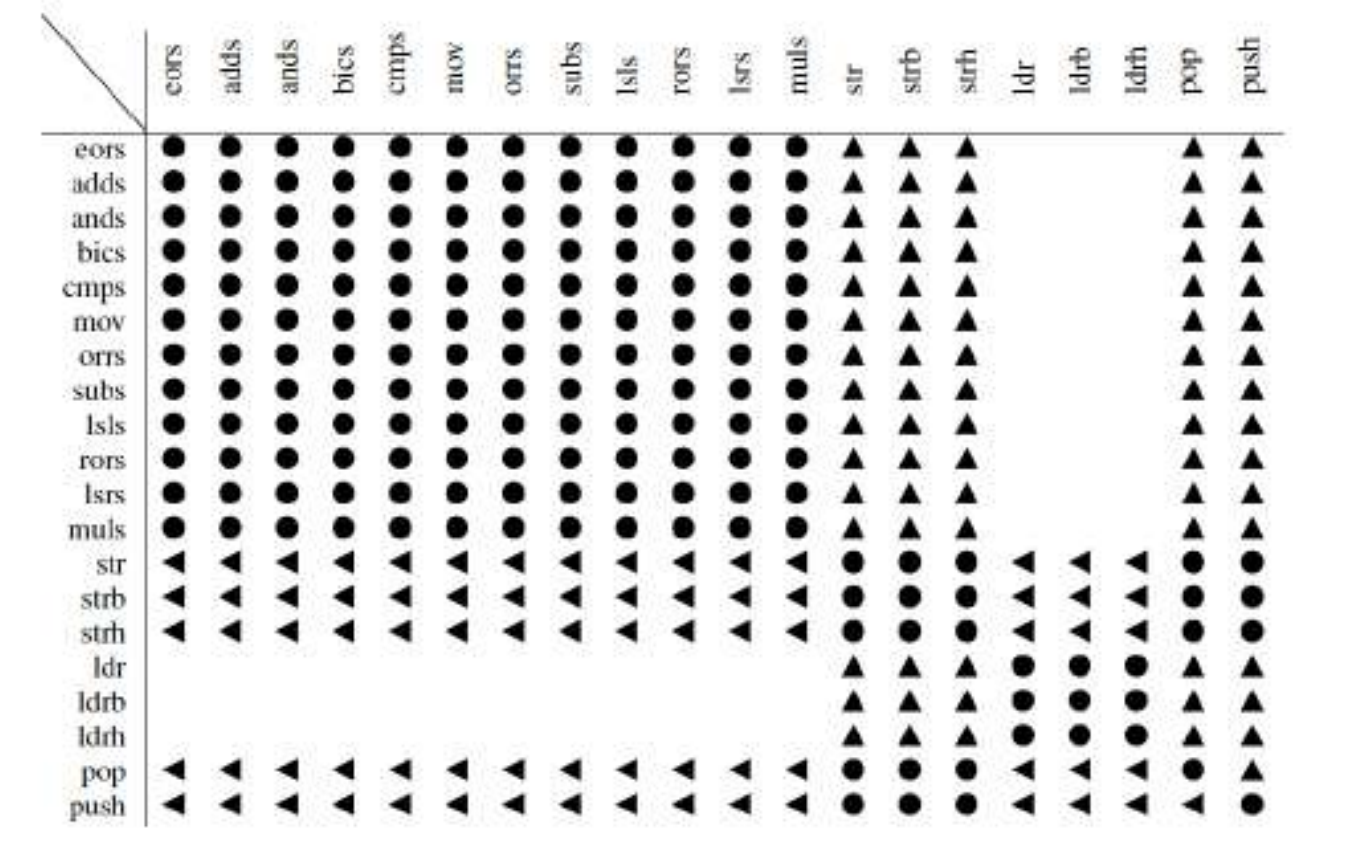
```
1  str   r1, [r2]
2  movs  r7, r7
3  movs  r7, r7
4  movs  r7, r7
5  eors  r3, r7
6  movs  r7, r7
7  movs  r7, r7
8  movs  r7, r7
9  str   r4, [r5]
```

This code sequence checks if str dominates eors.

If str dominates the eors, leakage will be visible at Line 9.

Basically, all instructions set some state, and most instruction pairs interact with this state.

Instructions interactions



Triangles point to the dominating instruction. Circles indicate interactions on the same storage.

Sources of leakage

- Registers: Overwriting a register leaks the (weighted) Hamming distance between the previous value and the new value
- Memory: Writing data to memory interacts with data already stored in the same location
- Instruction Pairs: All instructions set some state, and that most pairs do interact with this state
- Memory Bus: The memory bus seems to have a storage element that stores the most recent value stored to or loaded from the memory (leaking the Hamming distance between the previous and the new value)

Rosita design

- First, leaky components are identified using ELMO*
- Main strategy is to wipe stored state with a random mask using a dedicates mask register, which is initialized with a random 32-bit mask
- When *t*-test values suggests operand interaction, ROSITA inserts an access to the mask register

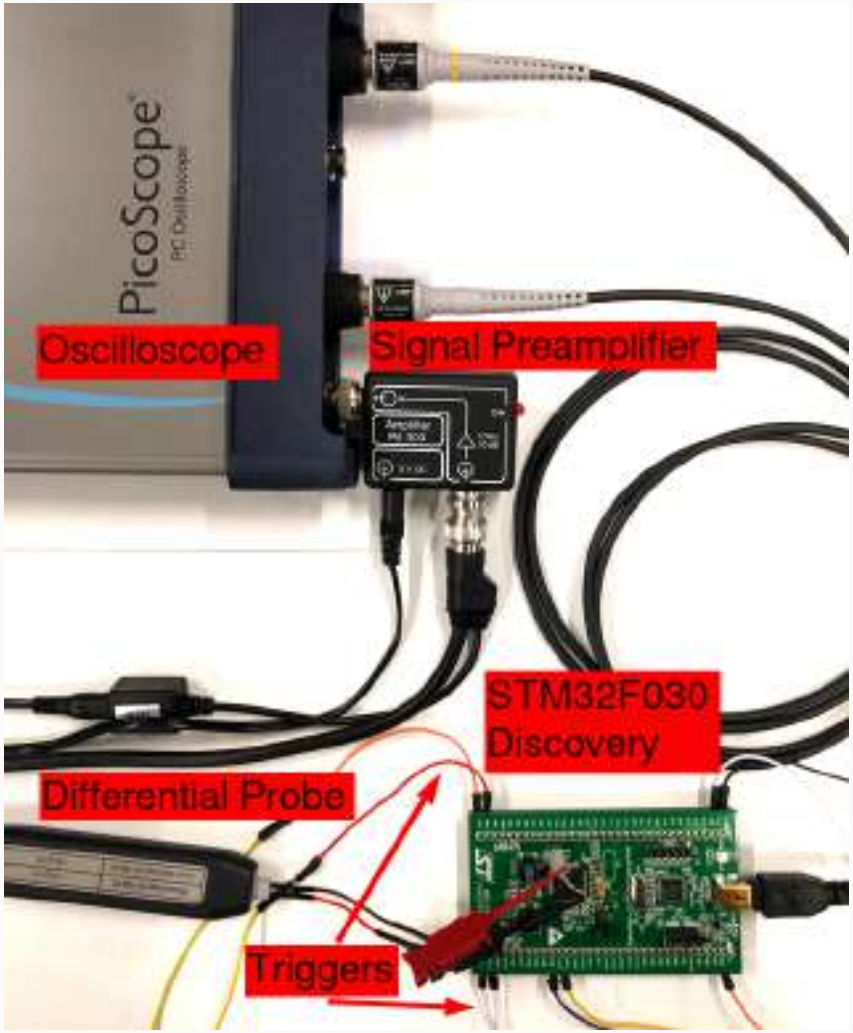
Fixing example

```
str r2, [r3]
```

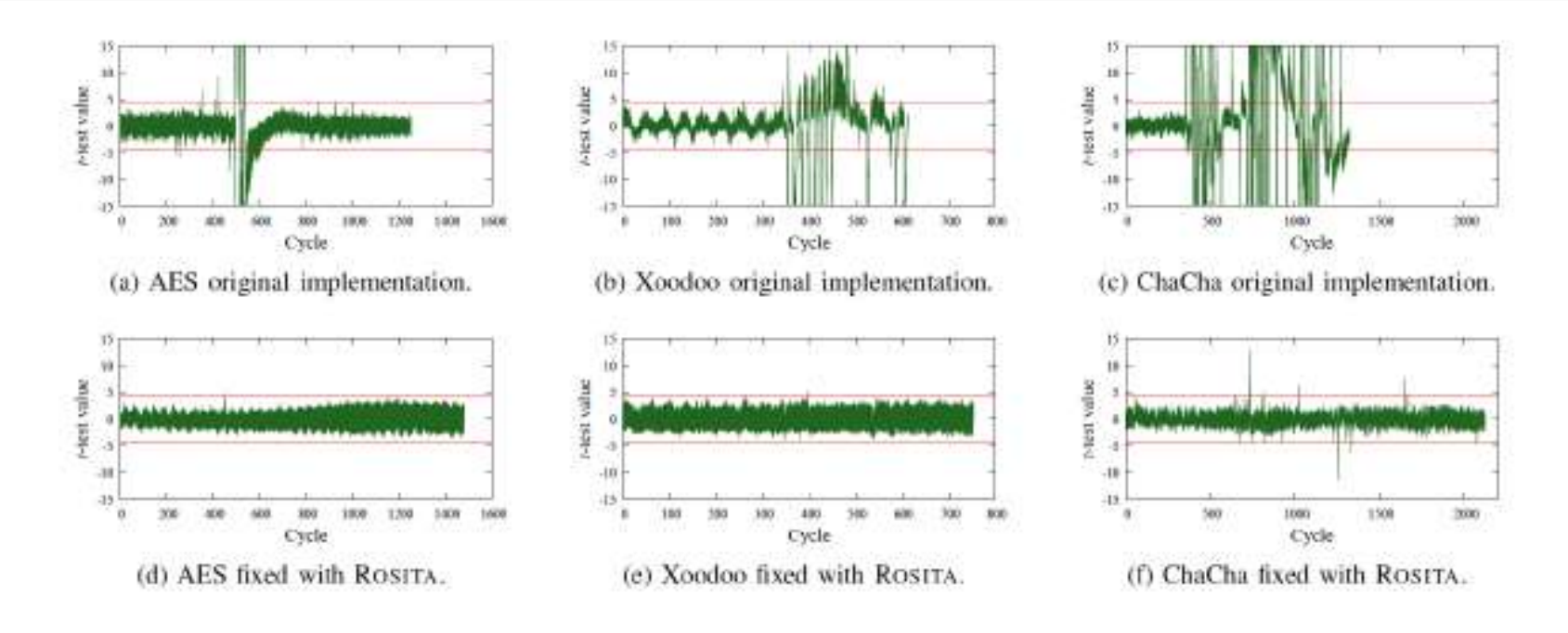
```
str r7, [r3]
str r2, [r3]
```

To fix interactions with the previous value used on the memory ROSITA first stores the mask register into the destination location and then performs the required store.

Evaluation setup



Results after fixing 3 different implementations

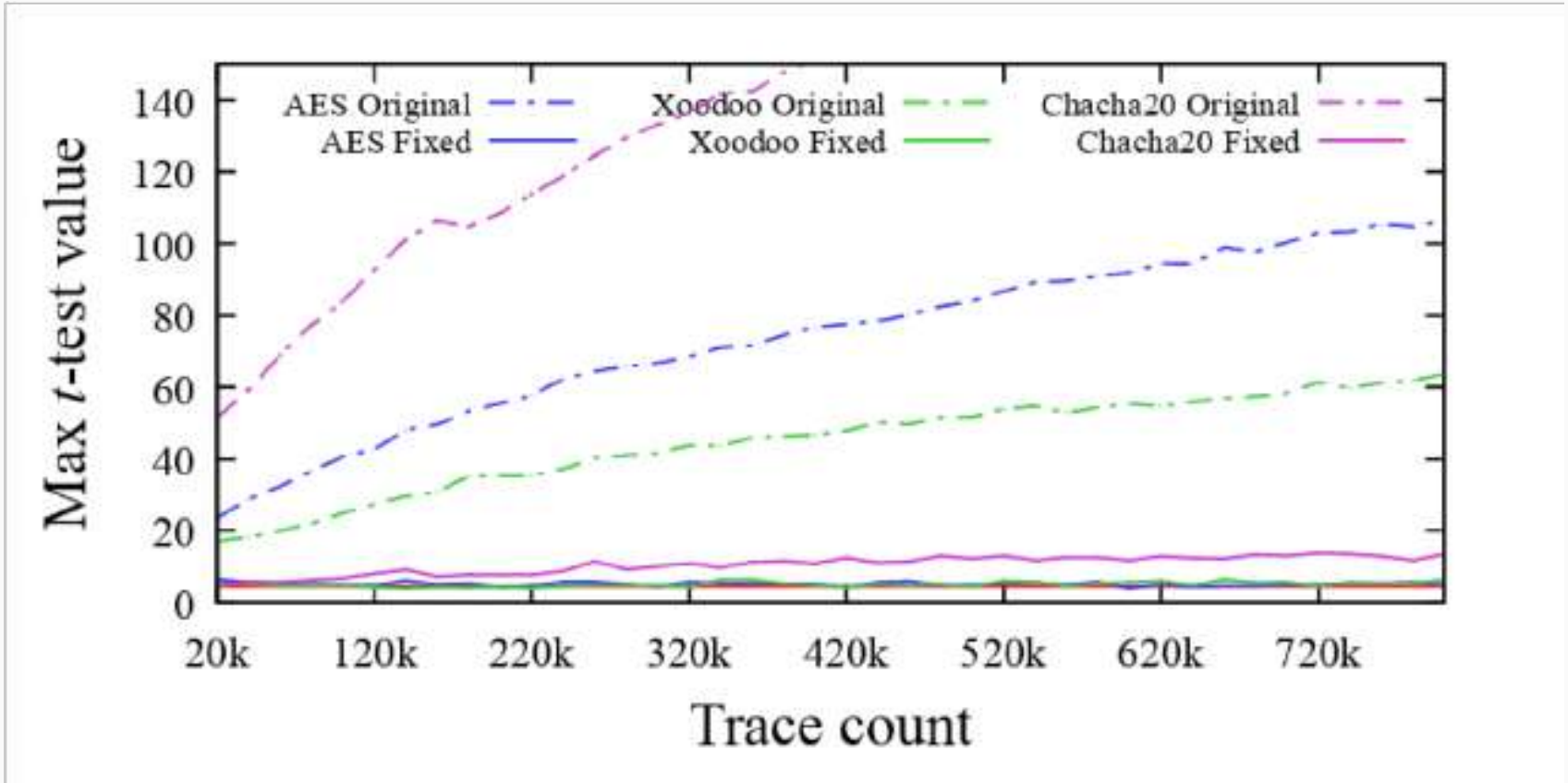


(a) AES original implementation. (b) Xoodoo original implementation. (c) ChaCha original implementation.

(d) AES fixed with ROSITA. (e) Xoodoo fixed with ROSITA. (f) ChaCha fixed with ROSITA.

The slowdowns of the "fixes" for ChaCha, Xoodoo and AES are 61% (1 322 vs. 2 122 cycles), 18% (637 vs. 753 cycles) and 15% (1 285 vs 1 479).

Leakage reduction



Results of running Rosita on masked AES, ChaCha, and Xoodo

	Cycles		Leakage Points	
Function	**Original**	**Fixed**	**Original**	**Remaining**
AES	1285	1479	31	0
ChaCha	1322	2162	238	1
Xoodoo	637	769	38	0

Contributions of Rosita

- A framework for generating 1st-order leakage-resistant implementations of masked ciphers by iteratively rewriting the code at leakage points
- We first developed Elmo*, a leakage emulator that improves on Elmo
- We created Rosita, a code rewrite engine that uses Elmo*
- After "fixing" the original code with Rosita we show the absence of observable leakage at 1 000 000 traces for AES and Xoodoo
- We show how Rosita can be used to automatically protect masked implementations of AES and Xoodoo with overheads of less than 21%

Follow-up work

- ROSITA extension on higher-order leakage → see ROSITA ++
- Extending the leakage evaluation part to some other statistical tests
- Can we come up with some more generic approach for leakage emulation?

Summary

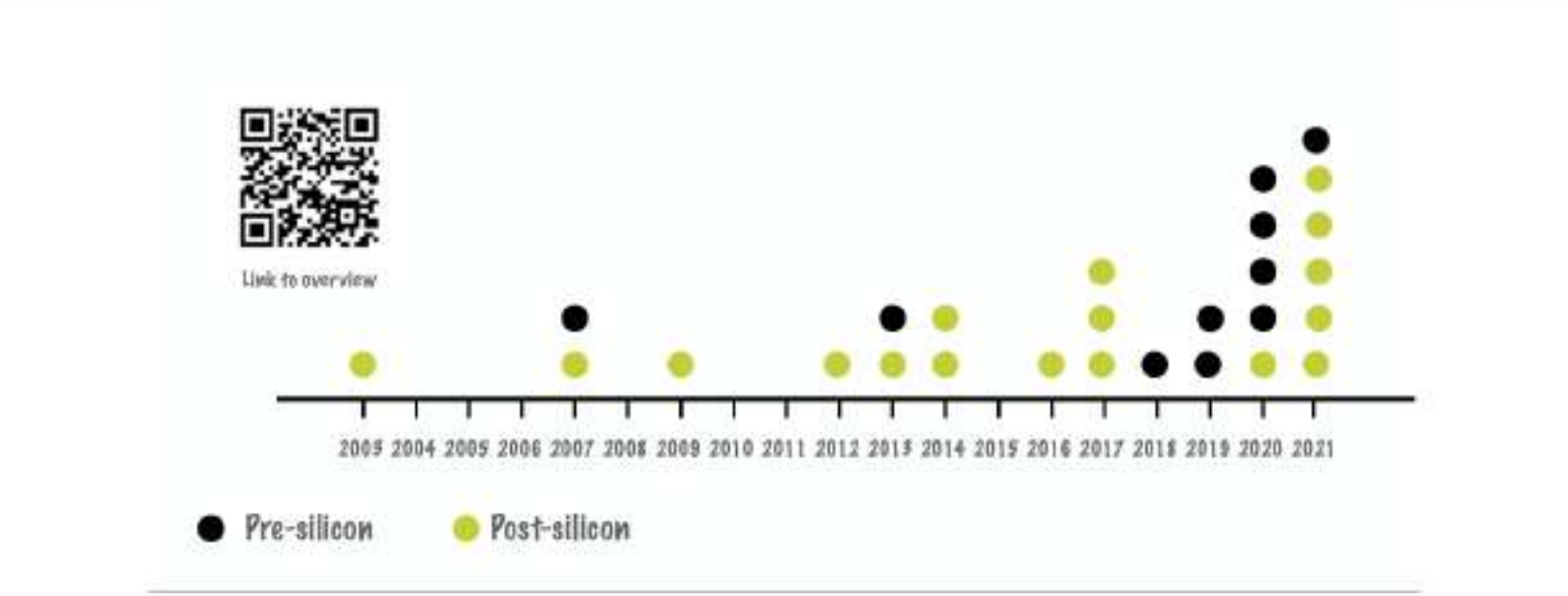
- Side-channel evaluation requires skilled evaluators performing complex and time consuming procedures
- Leakage emulation is a very active area of research
- We show how to remove leakage by using Rosita's code rewrites
- Results are evaluated using real hardware experiments
- Fixes are affordable wrt performance
- New directions in modeling the leakage: ABBY simulator

References

ROSITA: Towards Automatic Elimination of Power-Analysis Leakage in Ciphers
Madura A. Shelton (University of Adelaide), Niels Samwel and Lejla Batina (Radboud University), Francesco Regazzoni (University of Amsterdam and ALaRI – USI), Markus Wagner (University of Adelaide), Yuval Yarom (University of Adelaide and Data61). NDSS symposium, Feb. 21–25, 2021.
`https://github.com/0xADE1A1DE/Rosita`

ROSITA ++: Automatic Higher-Order Leakage Elimination from Cryptographic Code
Madura A. Shelton (University of Adelaide), Lukasz Chmielewski and Niels Samwel (Radboud University), Markus Wagner (University of Adelaide), Lejla Batina (Radboud University), Yuval Yarom (University of Adelaide and Data61). ACM CCS 2021, Nov. 15–19, 2021.

Leakage simulation is an active research area



SoK: Design Tools for Side-Channel-Aware Implementations: Ileana Buhan, Lejla Batina, Yuval Yarom and Patrick Schaumont, ASIACCS 2022.

Screen Gleaning

TEMPEST: Cause and History

Oscillating electric currents create EM radiation in the RF range and those signal drive the video display of various screens.

- Bell Labs noted this vulnerability for teleprinter communications during World War II producing 75% of the plaintext being processed from a distance of 24 m
- Van Eck phreaking: In 1985 published the first unclassified analysis of the security risks of emanations from computer monitors using just 15$ equipment+TV set
- Van Eck phreaking was used to successfully compromise ballot secrecy for electronic voting in Brazil
- NSA published TEMPEST Fundamentals in 1982 referring to spying on systems through leaking emanations, including radio or el. signals, sounds and vibrations
- TEMPEST covers both methods to spy and to shield equipment against such spying

Motivation and Outcomes

Motivation:

- TEMPEST attack is known for a long time but **no methodology** has been established to evaluate it **on mobile devices**
- Using TEMPEST the adversaries can reconstruct the images displayed through leaking emanations

In this work we:

- Introduce Screen Gleaning, a new electromagnetic TEMPEST attack targeting mobile phones
- Demonstrate the attack and its portability to different targets using machine learning
- Provide a testbed and parameterized attacker model for further research

Screen gleaning (Theory)

Screen gleaning (Practise)

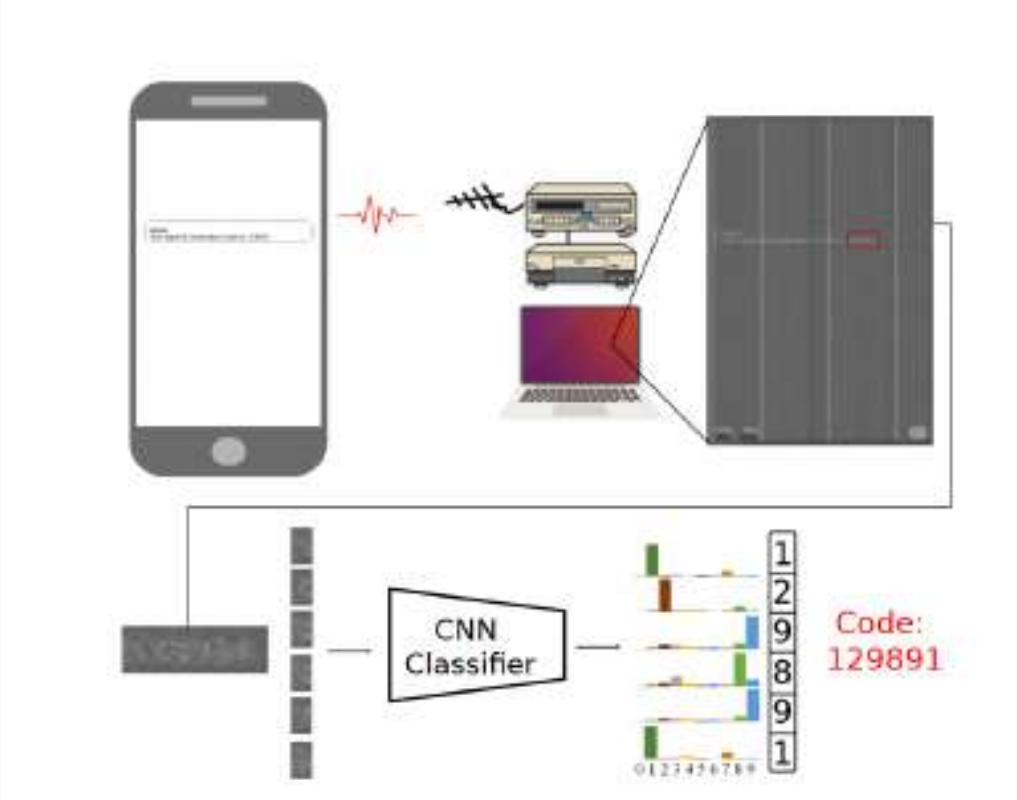
The signal we observe is, in most cases, not interpretable to the human eye.

Attacker model: Motivating story and assumptions

Alice keeps her phone on a stack of magazines on her desk (face down) to block the visual line of sight to the screen. Eve has hidden an antenna under the top magazine to read the security code via electromagnetic emanations of the phone.

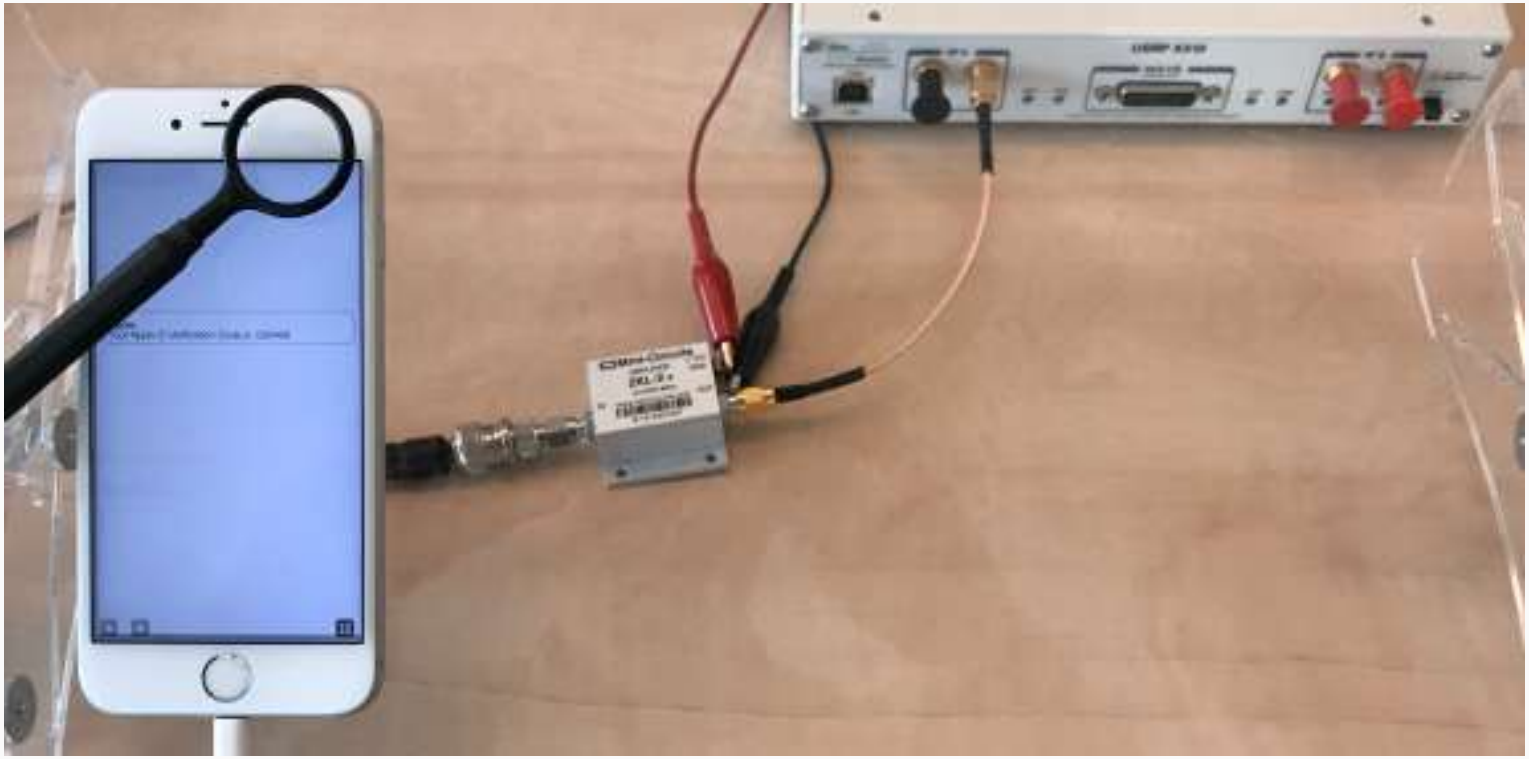
- The set of symbols displayed on the phone is finite and known (digits 0-9)
- The attacker has access to a profiling device that is "similar" to the target device
- The attacker can collect electromagnetic traces from the target device (representing the image displayed on the screen)

Attacker model: setup



- The target emits EM signal intercepted by an antenna connected to a software-defined radio (SDR)
- The leaked information is collected and reconstructed as a gray-scale image (emage)
- From emage, the 6-digit security code is cropped and fed into a CNN classifier for recognition

Screen gleaning setup



The security code

Use of authentication code like to extract all digits.

- display code
- sample leakage
- analyze the leakage
- interpret the results

Security code results

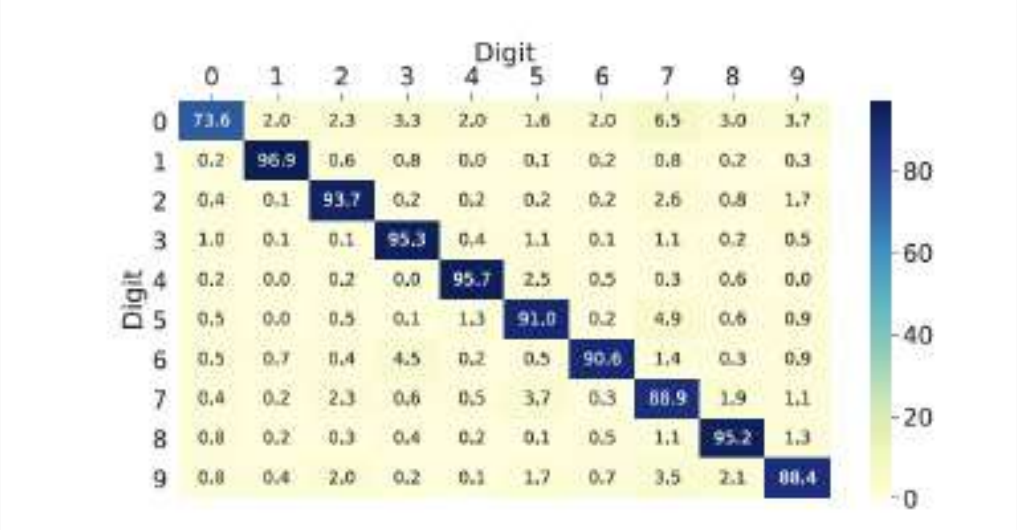


Figure: Confusion matrix of the inter-session accuracy of the security.

Digits	0	1	2	3	4	5	6	7	8	9	All
Acc. (%)	87.2	86.8	97.4	75.8	99.1	97.4	95.1	93.1	82.5	86.1	89.8

Table: Accuracy with respect to different digits (0-9) and overall accuracy in our security code attack.

	6 digits	≥ 5 **digits**	≥ 4 **digits**
Acc. (%)	50.5	89.5	99.0

Table: Accuracy of predicting partial security code correctly.

Security code results

- Attack on different phones of the same model
  E.g., cross-device accuracy of 61.5%, where the classifier is trained and tested on two distinct iPhone 6.
- Attack on different phone of different model
  E.g., accuracy of 74.0% on Huawei Honor 6X.
- Attack at a greater distance (through a magazine)
  E.g., accuracy of 65.8% on Huawei Honor 6X through 200 pages.

Z. Liu, Niels Samwel, L. Weissbart, Z. Zhao, D. Lauret, L. Batina, M. Larson, Screen Gleaning: A Screen Reading TEMPEST Attack on Mobile Devices Exploiting an Electromagnetic Side Channel, NDSS 2021.

Conclusions

- Screen gleaning is a new TEMPEST attack that uses an antenna and SDR to capture an electromagnetic side channel, i.e., emanations leaking from a mobile phone
- We demonstrated the effectiveness of it on three different phones with an example of the recovery of a security code
- We introduced 5-dimension attacker model that can be extended further
- We proposed a testbed providing a standard setup in which screen gleaning can be tested further with different attacker models

The end

Thank you for your attention!

`https://cescalab.cs.ru.nl/`