RUHR-UNIVERSITÄT BOCHUM

Statistical Independence and Leakage Verification (SILVER)

Formal Verification of Masking Schemes in Hardware

Pascal Sasdrich

ISC Winter School on Information Security and Cryptology (ISCwsISC 2021)

Chair for Security Engineering
Electrical Engineering and Information Technology
Ruhr University Bochum

INTRODUCTION

Side-Channel Analysis and Hardware Masking

Side-Channel Analysis

Status Quo

ASSUMPTION:

Cryptography is secure in the black-box model.

THE PROBLEM: IMPLEMENTATION ATTACKS

- 20+ years of research on Side-Channel Analysis (SCA)
- secret-dependent side-channel leakage of hardware implementations:
  - *timing variations*
  - *power consumption*
  - *electromagnetic emanations*
  - …

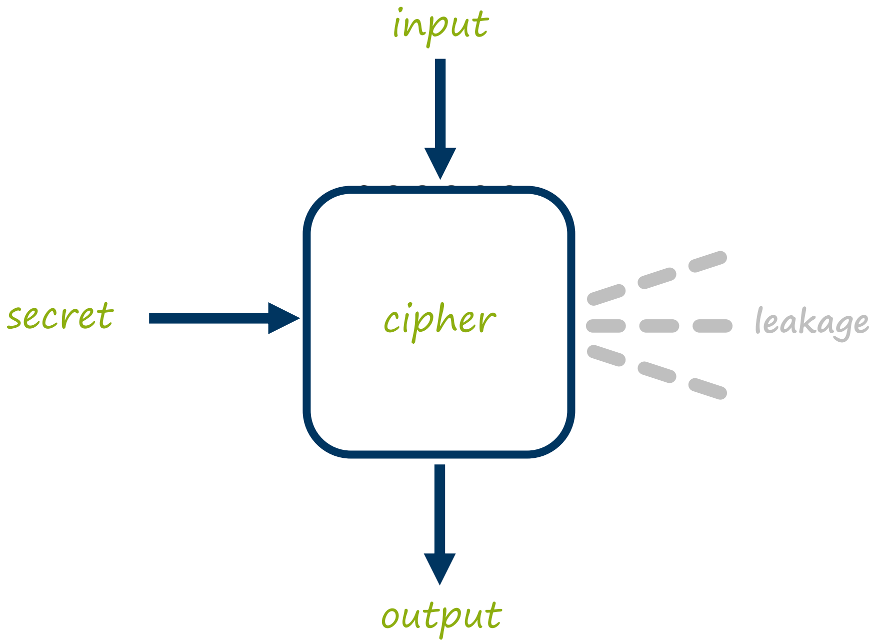


image sources: https://www.onlinewebfonts.com, https://www.flaticon.com

Side-Channel Analysis

Countermeasures

SOLUTION:

Randomizing execution in hardware.

THE APPROACH: MASKING SCHEMES

- *random sharing* of intermediate values
- algorithmic-level countermeasure
  - *independent of device characteristics*
  - *applied during implementation phase*
- different schemes used in practice:
  - *Threshold Implementations (TI)*
  - *Domain-Oriented Masking (DOM)*
  - …

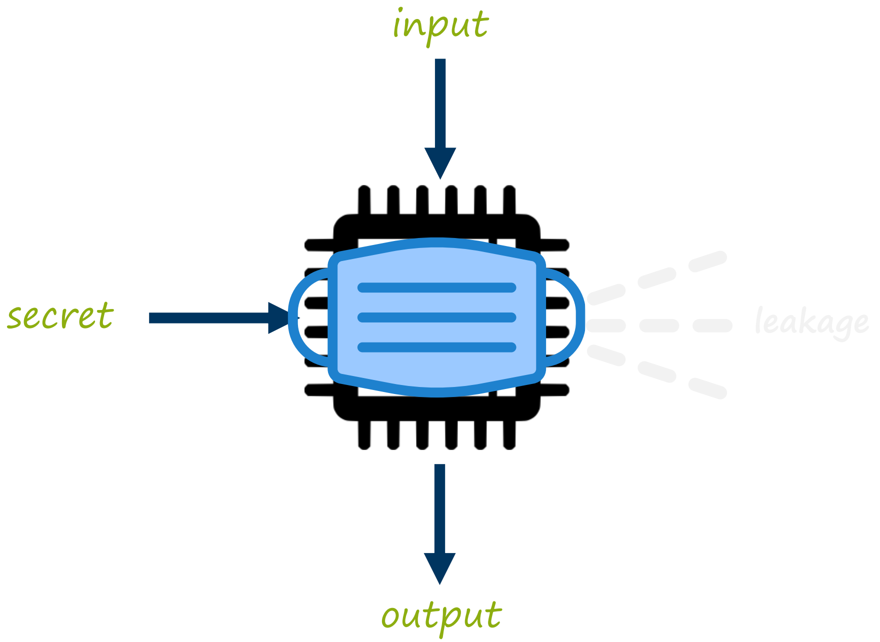


image sources: https://www.onlinewebfonts.com, https://www.flaticon.com

MOTIVATION

Why do we need formal verification of masking schemes in hardware?

Hardware Masking

The Hardware Design Process

OBSERVATION:

Secure design (hardware masking) spans across multiple phases of the hardware design process.

1. **concept:**
   → *selection of masking scheme*
2. **implementation:**
   → *protected hardware architecture*
3. **prototype:**
   → *practical evaluation & testing*

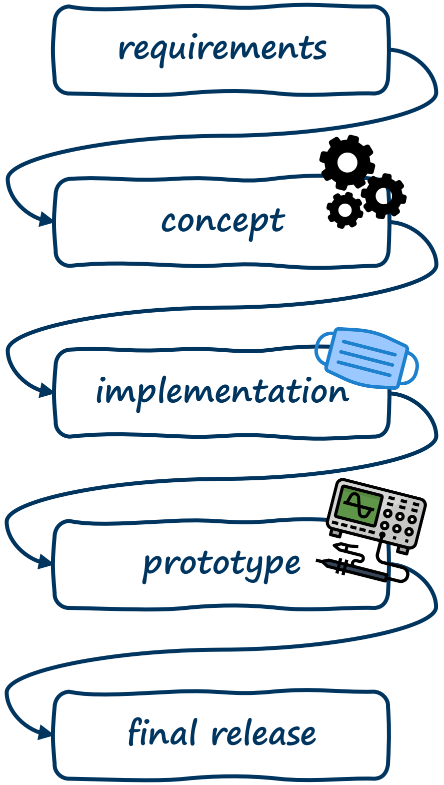


image sources: https://www.onlinewebfonts.com, https://www.flaticon.com

Hardware Masking

Empirical Testing & Evaluation

PROBLEM:

Design, implementation, and testing of masked hardware is a complex, manual, and error-prone task.

TESTING AND EVALUATION:

- long-standing security expertise required
- re-iteration for every detected flaw & issue
- downstream and time-consuming process

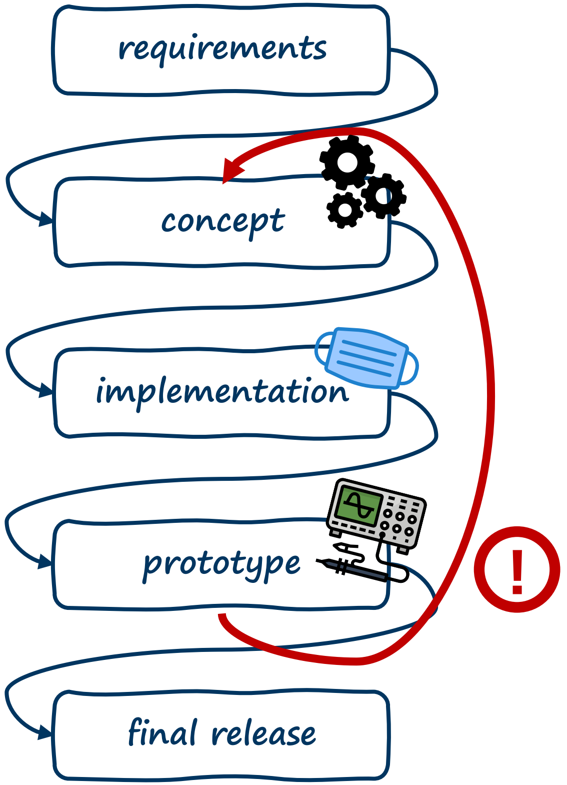


image sources: https://www.onlinewebfonts.com, https://www.flaticon.com

Hardware Masking

Formal Verification

IDEA:

Shorten the feedback path in the design cycle.

FORMAL VERIFICATION:

- early during the design cycle
- security proofs vs. expensive empirical testing

CHALLENGES (ADDRESSED IN THIS TALK):

→ *formal description of adversary model(s)*

→ *formal definition of security properties*

→ *automated verification of properties against model(s)*

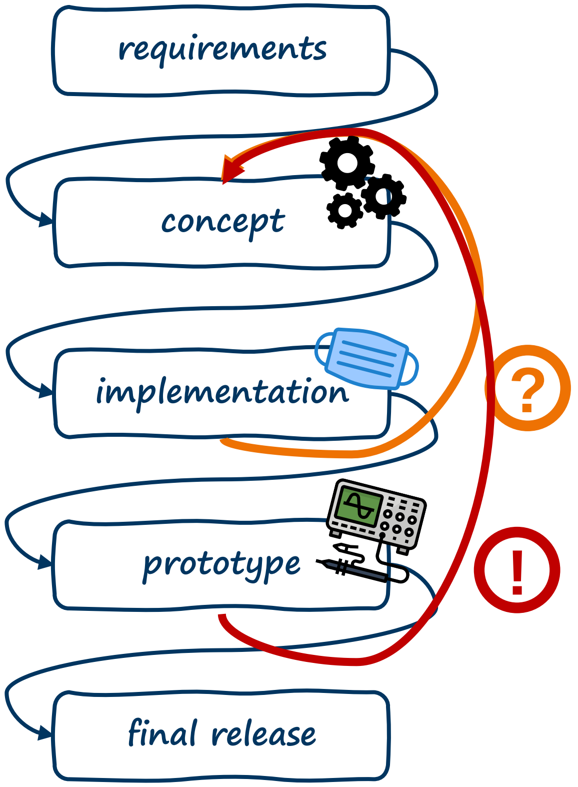


image sources: https://www.onlinewebfonts.com, https://www.flaticon.com

ADVERSARY MODEL

How to formally model adversaries and leakages?

Adversary Model

The Standard Probing Model [ISW03]

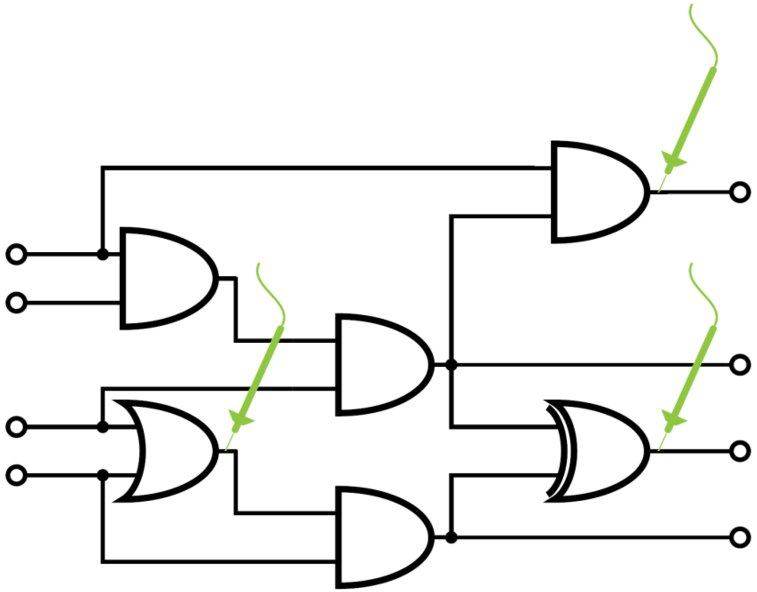
RUB

THE CORE IDEA:

Adversaries can probe up to d intermediate values of an ideal digital circuit.

THE PROBES:

- number of probes determines order of attack
- each adversarial probe is:
  - → exact (noise-free)
  - → instantaneous & stable
  - → independent of all other probes



Security in this model also implies security in other (more realistic) models [BDF+17].

Adversary Model

The Glitch-Extended (Robust) Probing Model [FGP+18]

THE PROBLEM:

Hardware circuits do not behave as ideal circuits.

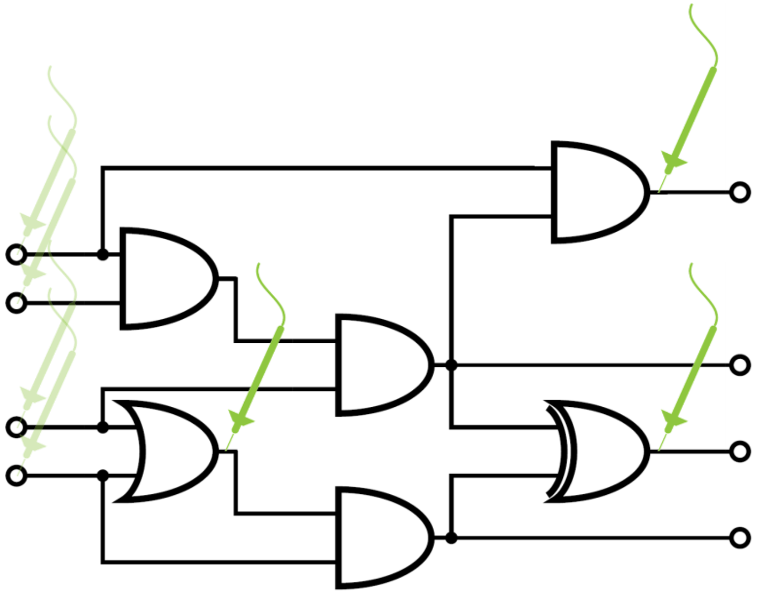
GLITCHES IN HARDWARE CIRCUITS:

- transient and unintentional computations
- can provide secret-dependent information
- sequential gates (e.g., flip-flops) stop glitches

HOW TO CONSIDER GLITCHES IN PROBING MODEL?

→ *glitch can reveal any combination of stable inputs*

→ *glitch-extended probes (worst-case scenario)*



PROBING SECURITY

How to formally define side-channel security?

Probing Security

Formal Definition

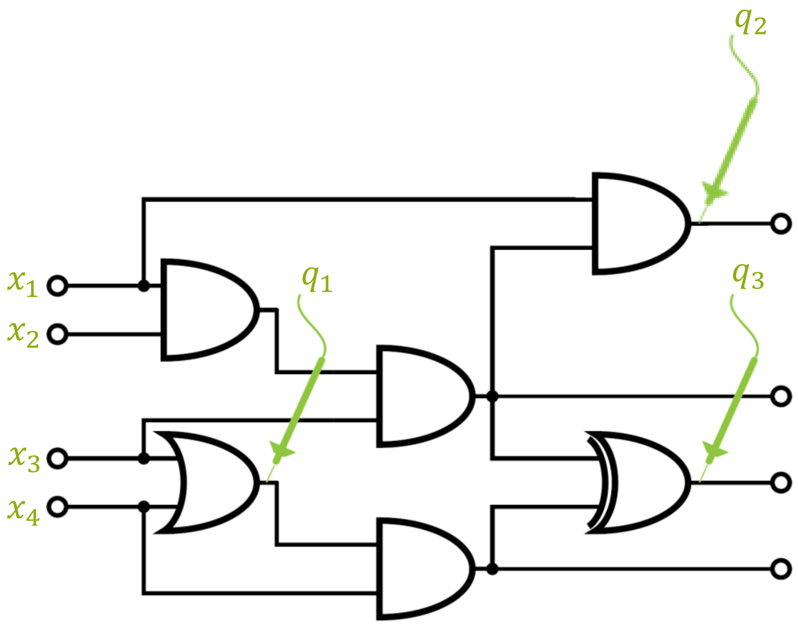
RUB

THE DEFINITION:

*A circuit C with secret input set **X** is d-probing secure, if and only if for any observation set **Q** containing d (probed) wires, **X** is statistically independent of the observation set:*

$$Pr[\mathbf{Q}|\mathbf{X}] = Pr[\mathbf{Q}]$$

- → ***Q***: joint distribution of probes
- → ***X***: joint distribution of secret values (unshared)
- → modeling wires as binary random variables
- → all primary inputs are assumed to be *independent and identically distributed* (i.i.d.)



Probing Security

Gadgets and Composability

THE CHALLENGE:

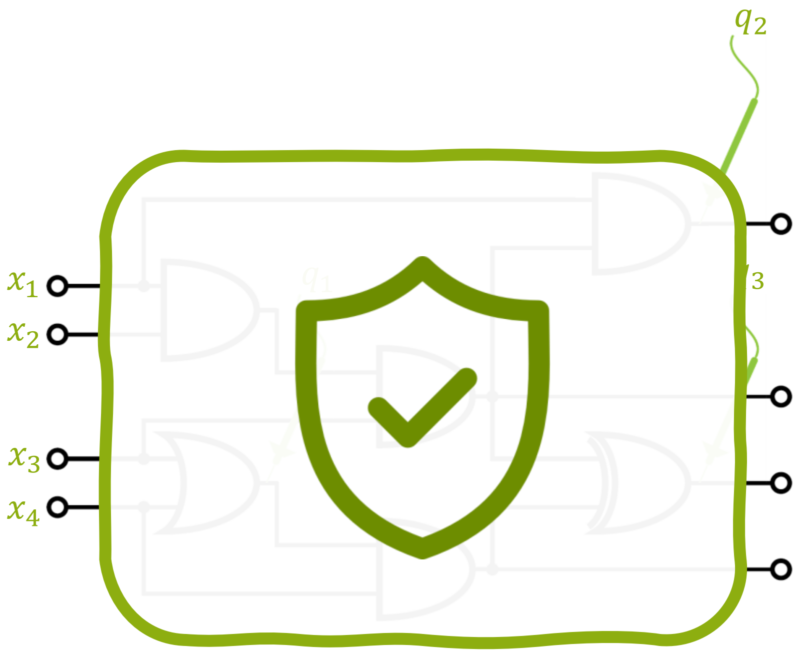
Design complexity increases with circuit size (number of gates) and d (number of probes).

THE APPROACH:

- masking of atomic functions (gadgets)
- compose full circuit of secure gadgets.

THE PROBLEM:

Composition of secure gadgets is not trivial and often results in insecure designs.



https://www.freepik.com

COMPOSABILITY

How to securely compose masked gadgets?

Composability

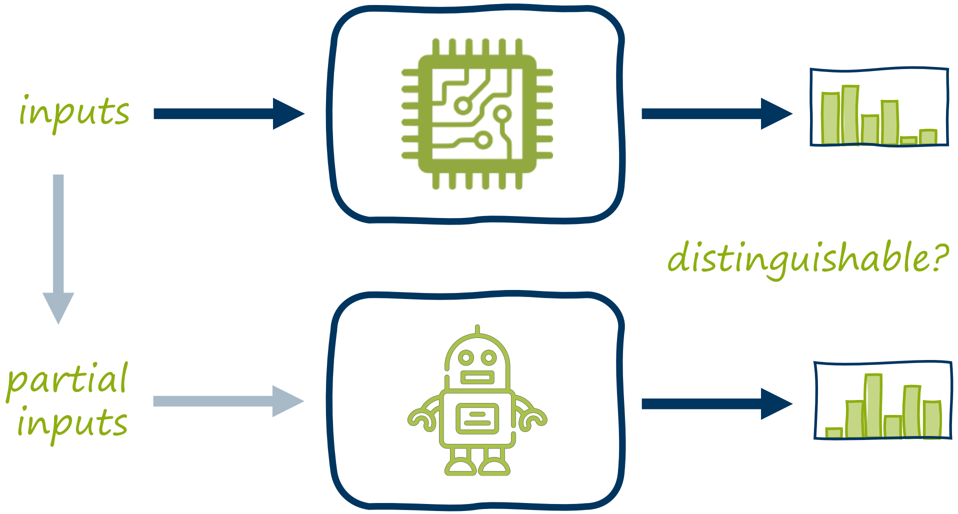
Non-Interference (NI) [BBD+15]

NON-INTERFERENCE & SIMULATABILITY:

- construction of a simulator for each probe
- simulator operates on partial information:
  - *uses ≤ d shares of secret variable*
  - *simulated distribution vs. observed distribution*

→ secure if we can find a simulator such that the distributions are indistinguishable

→ ensures composability (spoiler: no!)



OUR CONTRIBUTION:

→ non-interference based on statistical independence

→ allows arbitrary sharings and number of secrets

Image sources: https://www.flaticon.com

Composability

d-Non-Interference (NI)

THE DEFINITION:

*A circuit C with secret input set **X** is d-non-interfering, if and only if for any observation set **Q** containing $t \leq d$ (probed) wires, there exists a set **S** of input shares with $|\boldsymbol{S}_{\forall i}| \leq t$ such that*

$$Pr[\boldsymbol{Q}|\boldsymbol{S}] = Pr[\boldsymbol{Q}|Sh(\boldsymbol{X})]$$

- → ***Q*:** joint distribution of probes
- → ***S***: joint distribution of subset of secret shares
- → *Sh(**X**)*: joint distribution all secret shares

Adversary may learn partial information without interference on processed secrets.

https://www.freepik.com

Composability

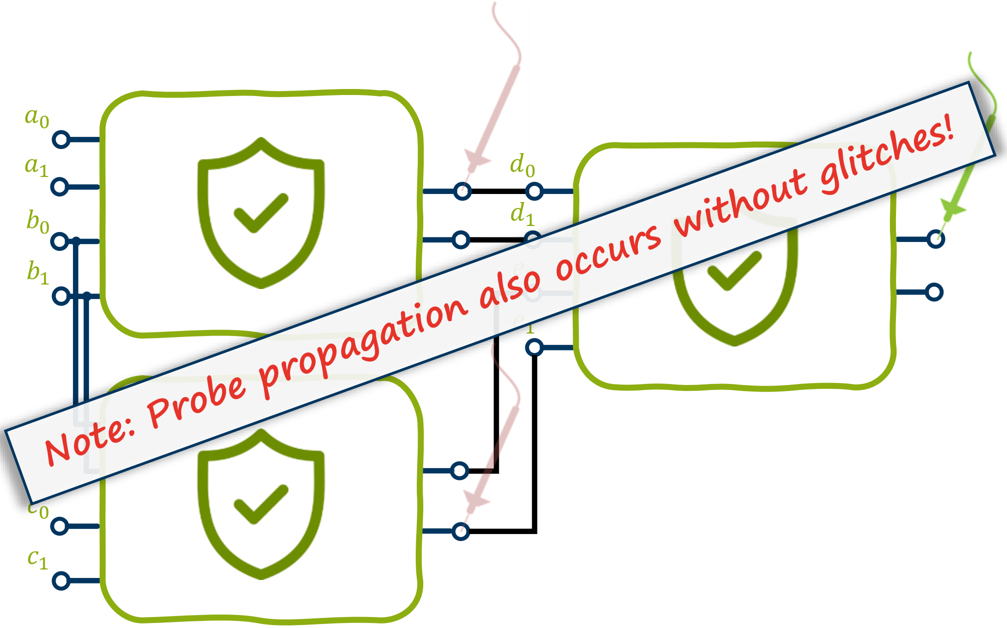
Non-Interference (NI) [BBD+15]

THE PROBLEM:

Probe propagation limits composability.

AN EXAMPLE:

- composition of three gadgets ($d \leq 1$)
- single probe on output of last gadget
- adversary learns d_0 and e_1 (distributions) ☑
  - (virtual) probes on outputs of gadgets
  - adversary learns b_0, b_1 (distributions) ☒



https://www.freepik.com

Composability

Strong Non-Interference (SNI) [BBD+16]

THE SOLUTION:

Stop propagation of output probes.

THE DEFINITION:

*A circuit C with secret input set **X** is d-strong-non-interfering, if and only if for any observation set **Q** containing $t = t_1 + t_2 \leq d$ (probed) wires, there exists a set **S** of input shares with $|\boldsymbol{S}_{\forall i}| \leq t_1$such that*

$$Pr[\boldsymbol{Q}|\boldsymbol{S}] = Pr[\boldsymbol{Q}|Sh(\boldsymbol{X})]$$

→ $\boldsymbol{t_1}$: number of probed internal wires

→ $\boldsymbol{t_2}$: number of probed output wires

https://www.freepik.com

Adversary may only learn partial information when probing internal wires.

Composability

Strong Non-Interference (SNI) [BBD+16]

A PROBLEM:

*Trivial gadgets of linear operations are **not** strong non-interfering.*

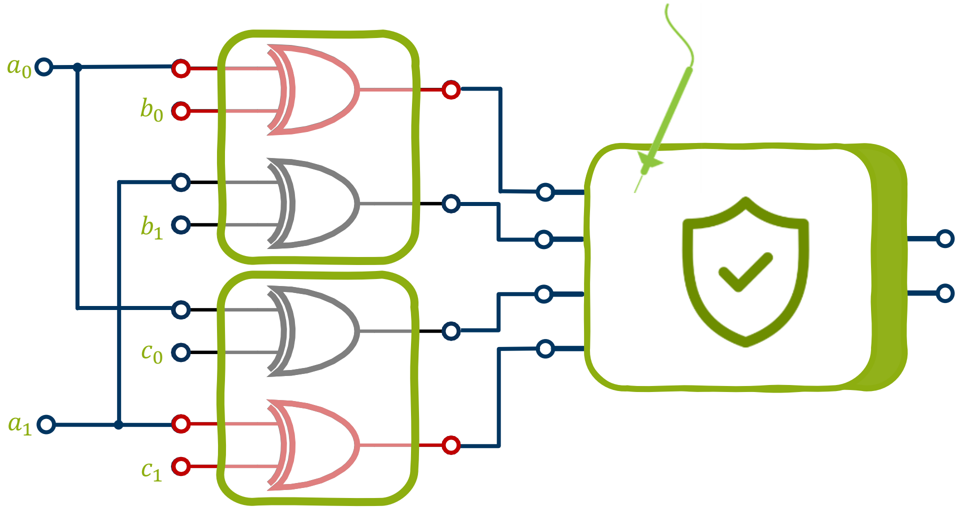
AN EXAMPLE:

- composition of a secure gadget ($d \leq 1$)
  - two trivial linear gadgets on the inputs
  - linear gadgets share one input
- adversary learns a_0, a_1 with one probe

THE SOLUTION:

Refreshing linear operations (at least one) to provide strong non-interference.

Expensive and inefficient!



$$f(a, b, c) = (a + b)(a + c)$$

https://www.freepik.com

Composability

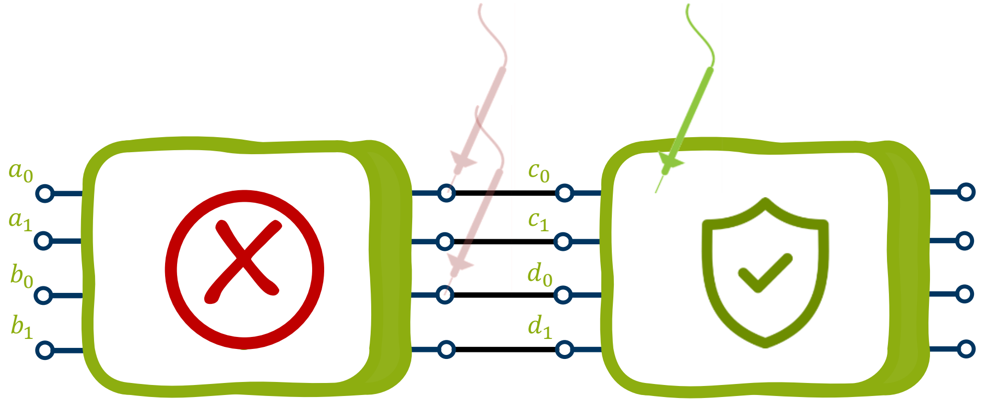
Strong Non-Interference (SNI) [BBD+16]

ANOTHER PROBLEM:

Strong non-interference *is limited to single-output gadgets.*

AN EXAMPLE:

- composition of two secure gadgets ($d \leq 1$)
- single internal probe on 2nd gadget
- adversary learns c_0 and d_0 (distributions)
  - same as having two probes on 1st gadget
  - violates security assumptions for gadget
  - worst-case: adversary learns all inputs



https://www.freepik.com

Composability

Probe-Isolating Non-Interference [CS20]

RUB

THE SOLUTION:

Ensure domain isolation for probe propagation.

THE DEFINITION:

A circuit C with secret input set $\boldsymbol{X}$ is d-probe-isolating-non-interfering, if and only if for any observation set $\boldsymbol{Q}$ containing $t = t_1 + t_2 \leq d$ (probed) wires, there exist sets $\boldsymbol{I}_{PI}$ and $\boldsymbol{I}_{PO}$ such that $\boldsymbol{S} = Sh(\boldsymbol{X})^{\boldsymbol{I}_{PI} \cup \boldsymbol{I}_{PO}}$ and

$$Pr[\boldsymbol{Q}|\boldsymbol{S}] = Pr[\boldsymbol{Q}|Sh(\boldsymbol{X})]$$

→ $\boldsymbol{I}_{PI}$: primary input indices, $|\boldsymbol{I}_{PI}| \leq t_1$

→ $\boldsymbol{I}_{PO}$: (probed) primary output indices, $|\boldsymbol{I}_{PO}| \leq t_2$

***Output probes* propagate to <u>same</u> input index,**
***internal probes* propagate to <u>at most one</u> (additional) input index.**

https://www.freepik.com

VERIFICATION

How to formally verify security notions for adversary models?

State-of-the-art in formal verification

Language-Based Formal Verification (maskVerif) [BBC+19]

THE STATE-OF-THE-ART IN FORMAL VERIFICATION OF MASKING SCHEMES IN HARDWARE:

→ maskVerif: *language-based tool with wide range of features*

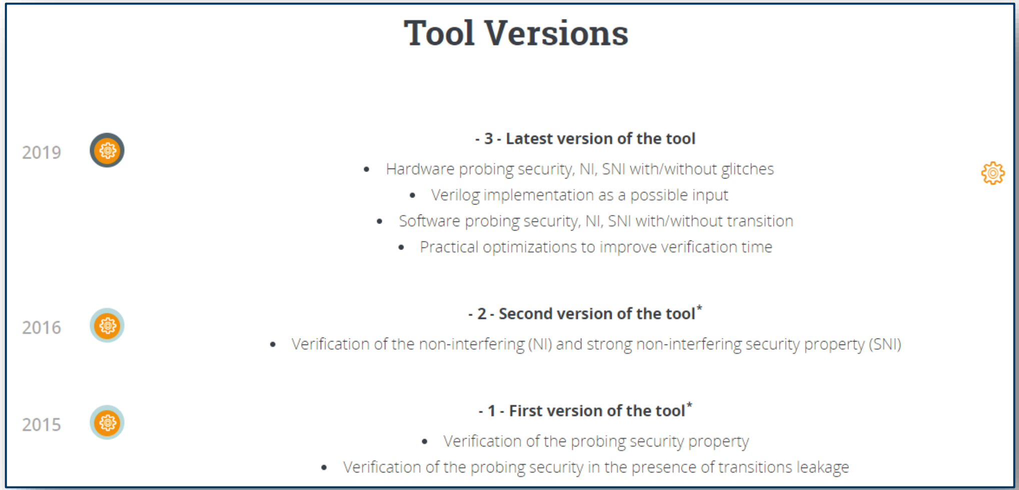
THE PROBLEM:

- *language-based verification not in direct conformity with security notions*
- *worst-case: may report false negatives*

OUR MISSION:

→ sound and complete verification in direct conformity with security notions

source: https://cryptoexperts.com/maskverif/


Tool Versions

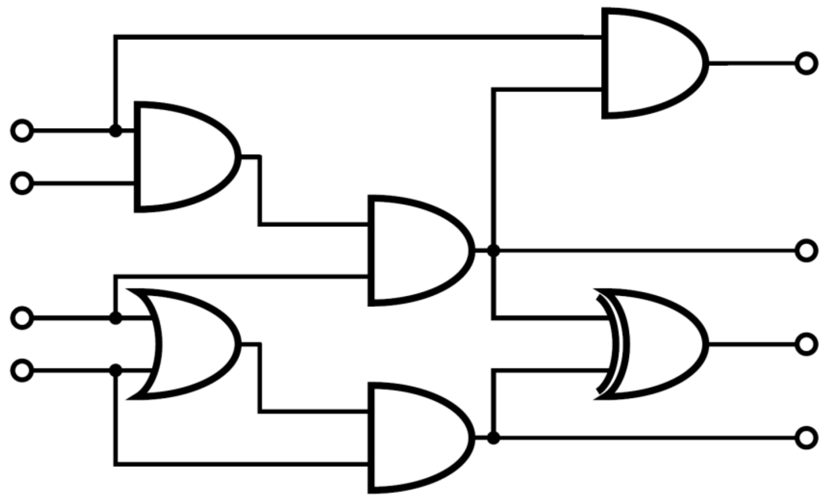
2019 **- 3 - Latest version of the tool**
- Hardware probing security, NI, SNI with/without glitches
- Verilog implementation as a possible input
- Software probing security, NI, SNI with/without transition
- Practical optimizations to improve verification time

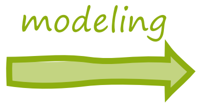
2016 **- 2 - Second version of the tool***
- Verification of the non-interfering (NI) and strong non-interfering security property (SNI)

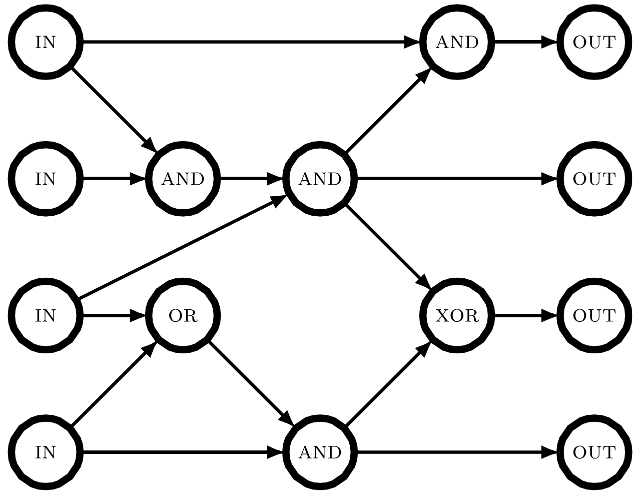
2015 **- 1 - First version of the tool***
- Verification of the probing security property
- Verification of the probing security in the presence of transitions leakage

Verification approach

Modeling Circuits & Netlists







NETLIST:

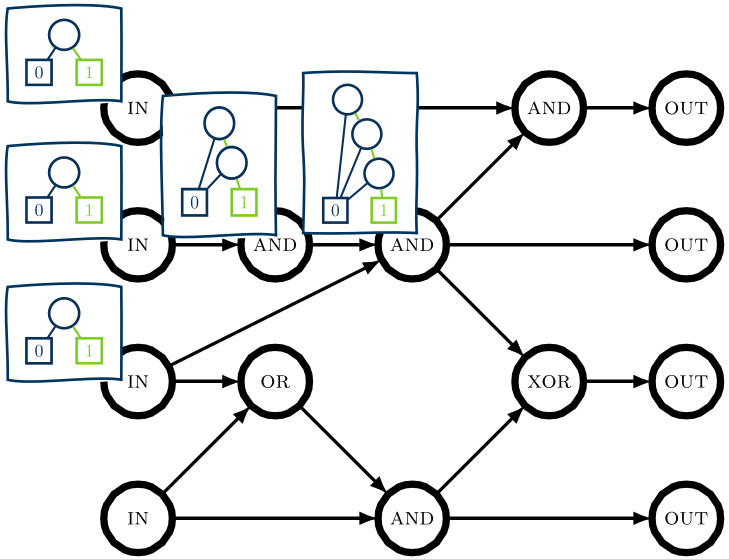
- *Hardware Description Language (Verilog or VHDL)*
- *constructed over cell library (e.g., NAND45)*
- *list of gates (sequential & combinational) and wires*
- *flatten vs. hierarchical (modules)*

CIRCUIT MODEL:

- *Directed Acyclic Graph (DAC)*
  - $\longrightarrow$ *vertices:* gates, *edges:* wires
- *limited set of combinational gates:*
  - $\longrightarrow$ NOT, AND, NAND, OR, NOR, XOR, XNOR

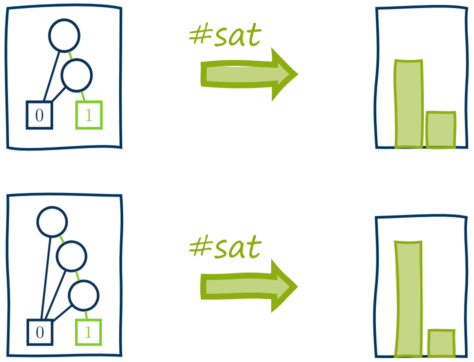
Verification approach

Checking Probability Distributions and Statistical Independence



Reduced Ordered Binary Decision Diagrams:

- *recursive generation of BDDs for each node*
- *requires topologically sorted graph*



Probability distributions:

- *counting satisfiable solutions (#P-complete)*
- *implicit assumption on i.i.d. inputs*
- *statistical independence on binary events*

Verification approach

Analysis of Security and Composability Notions

RECAP OF DEFINITIONS

- *probing security:* $Pr[\boldsymbol{Q}|\boldsymbol{X}] = Pr[\boldsymbol{Q}]$
- *composability notions:* $Pr[\boldsymbol{Q}|\boldsymbol{S}] = Pr[\boldsymbol{Q}|Sh(\boldsymbol{X})]$

ANALYSIS

1. *construction of probing sets **Q** (all combinations of d probes)*
2. *construction of set of unshared secrets **X** (probing security)*
3. *construction of set of partial inputs **S** (according to composability notion)*
4. *generation of joint distributions for all sets (using BDDs & #sat)*
5. *checking statistical independence according to definitions*

COMPLEXITY AND LIMITATIONS

- *higher-order testing limited by circuit size and order d:* $\#\mathbf{sets} = \binom{\#gates}{\#probes}$
- *BDD sizes limited by number of variables (primary circuit inputs)*

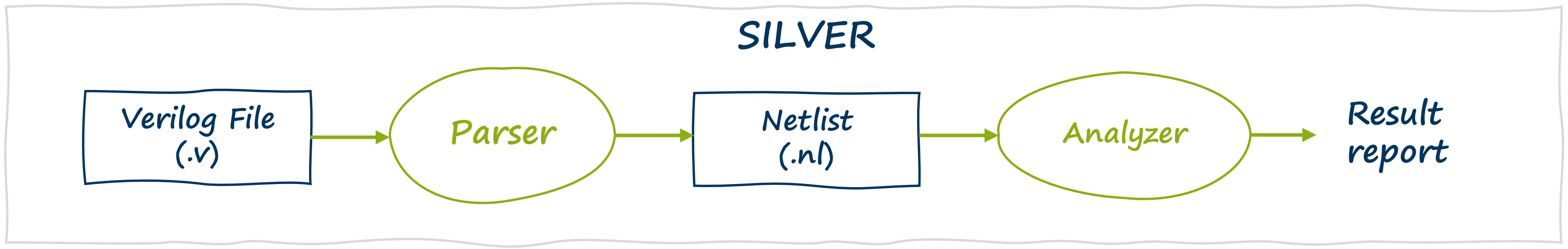
image sources: https://www.onlinewebfonts.com, https://www.flaticon.com

SILVER

Statistical Independence and Leakage Verification

SILVER

The General Verification Flow



PARSER:

- reads synthesized netlist (Verilog)
  - reduced number of gates supported
  - annotations for inputs/outputs
- creates (custom) intermediate representation
  - topologically sorted list of instructions
  - adds input, output, and refresh instructions

ANALYZER:

- constructs DAG from instructions
- elaborates DAG (ROBDD generation)
  - ROBDD for each gate in graph
  - recursive construction from input ROBDDs
- Generates probe combinations & performs analysis of security notions

SILVER

List of Features

PROBING SECURITY

- *current version limited to Boolean masking*
- *reports first failing combination of up to d probes*

→ *standard (SW) and glitch-extended (HW) model*

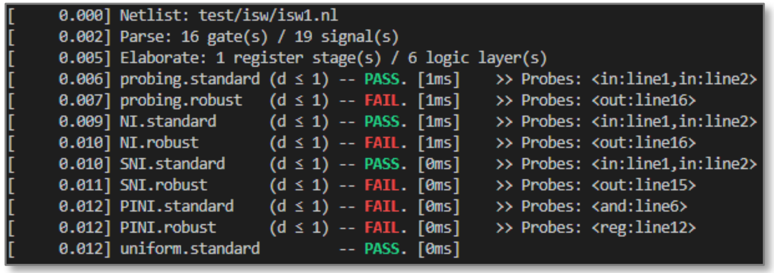
COMPOSABILITY NOTIONS

- *d-Non-Interference (d-NI)*
- *d-Strong-Non-Interference (d-SNI)*
- *d-Probe-Isolating-Non-Interference (d-PINI)*

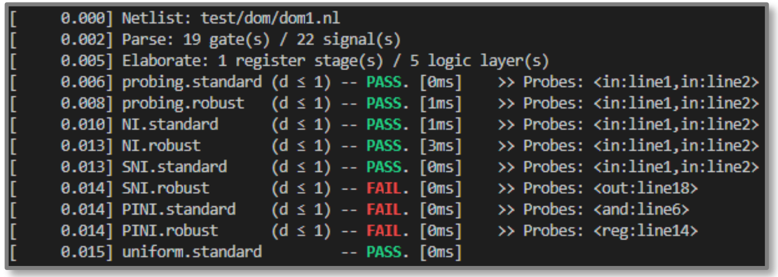
→ *standard (SW) and glitch-extended (HW) model*

OUTPUT UNIFORMITY

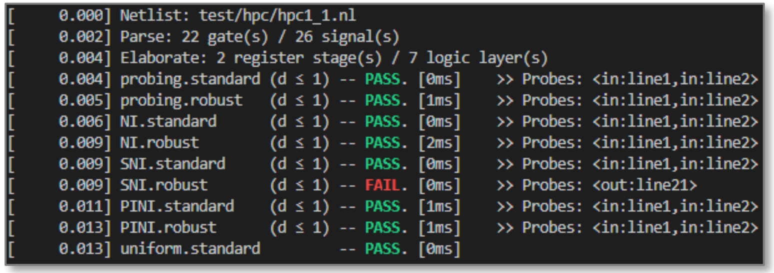
- *redefinition of uniformity over balanced outputs*
- *not covered in this talk (cf. paper for more details)*



```
[     0.000] Netlist: test/isw/isw1.nl
[     0.002] Parse: 16 gate(s) / 19 signal(s)
[     0.005] Elaborate: 1 register stage(s) / 6 logic layer(s)
[     0.006] probing.standard (d ≤ 1) -- PASS. [1ms]    >> Probes: <in:line1,in:line2>
[     0.007] probing.robust   (d ≤ 1) -- FAIL. [1ms]    >> Probes: <out:line16>
[     0.009] NI.standard      (d ≤ 1) -- PASS. [1ms]    >> Probes: <in:line1,in:line2>
[     0.010] NI.robust        (d ≤ 1) -- FAIL. [1ms]    >> Probes: <out:line16>
[     0.010] SNI.standard     (d ≤ 1) -- PASS. [0ms]    >> Probes: <in:line1,in:line2>
[     0.011] SNI.robust       (d ≤ 1) -- FAIL. [0ms]    >> Probes: <out:line15>
[     0.012] PINI.standard    (d ≤ 1) -- FAIL. [0ms]    >> Probes: <and:line6>
[     0.012] PINI.robust      (d ≤ 1) -- FAIL. [0ms]    >> Probes: <reg:line12>
[     0.012] uniform.standard         -- PASS. [0ms]
```



```
[     0.000] Netlist: test/dom/dom1.nl
[     0.002] Parse: 19 gate(s) / 22 signal(s)
[     0.005] Elaborate: 1 register stage(s) / 5 logic layer(s)
[     0.006] probing.standard (d ≤ 1) -- PASS. [0ms]    >> Probes: <in:line1,in:line2>
[     0.008] probing.robust   (d ≤ 1) -- PASS. [1ms]    >> Probes: <in:line1,in:line2>
[     0.010] NI.standard      (d ≤ 1) -- PASS. [1ms]    >> Probes: <in:line1,in:line2>
[     0.013] NI.robust        (d ≤ 1) -- PASS. [3ms]    >> Probes: <in:line1,in:line2>
[     0.013] SNI.standard     (d ≤ 1) -- PASS. [0ms]    >> Probes: <in:line1,in:line2>
[     0.014] SNI.robust       (d ≤ 1) -- FAIL. [0ms]    >> Probes: <out:line18>
[     0.014] PINI.standard    (d ≤ 1) -- FAIL. [0ms]    >> Probes: <and:line6>
[     0.014] PINI.robust      (d ≤ 1) -- FAIL. [0ms]    >> Probes: <reg:line14>
[     0.015] uniform.standard         -- PASS. [0ms]
```



```
[     0.000] Netlist: test/hpc/hpc1_1.nl
[     0.002] Parse: 22 gate(s) / 26 signal(s)
[     0.004] Elaborate: 2 register stage(s) / 7 logic layer(s)
[     0.004] probing.standard (d ≤ 1) -- PASS. [0ms]    >> Probes: <in:line1,in:line2>
[     0.005] probing.robust   (d ≤ 1) -- PASS. [1ms]    >> Probes: <in:line1,in:line2>
[     0.006] NI.standard      (d ≤ 1) -- PASS. [0ms]    >> Probes: <in:line1,in:line2>
[     0.009] NI.robust        (d ≤ 1) -- PASS. [2ms]    >> Probes: <in:line1,in:line2>
[     0.009] SNI.standard     (d ≤ 1) -- PASS. [0ms]    >> Probes: <in:line1,in:line2>
[     0.009] SNI.robust       (d ≤ 1) -- FAIL. [0ms]    >> Probes: <out:line21>
[     0.011] PINI.standard    (d ≤ 1) -- PASS. [1ms]    >> Probes: <in:line1,in:line2>
[     0.013] PINI.robust      (d ≤ 1) -- PASS. [1ms]    >> Probes: <in:line1,in:line2>
[     0.013] uniform.standard         -- PASS. [0ms]
```

SILVER

Discussion: Advantages & Limitations

ADVANTAGES:

→ **Completeness**
SILVER supports all recent and established security and composability notions.

→ **Soundness**
SILVER avoids false negatives & overhead due to direct conformity to the security notions.

→ **Verbosity**
SILVER directly reports failing probes, allowing an easier fix of masked design.

LIMITATIONS:

→ **Performance:**
Due to its direct conformity with security notions, SILVER is slower than language-based approaches (e.g., maskVerif).

DEMO

SILVER

CONCLUSION

Your free takeaway for today :)

Conclusion

Formal Verification of Masking Schemes in Hardware

1. **Secure implementation of masking schemes in hardware is a challenging:**

$\longrightarrow$ *manual, complex, and error-prone task requiring longstanding expertise*

$\longrightarrow$ *downstream testing & evaluation results in long development cycles*

2. **Formal verification provides early feedback to designers (shorten development cycles)**

$\longrightarrow$ *ISW d-probing model to define adversarial capabilities*

$\longrightarrow$ *composability notions reduce complexity in verification and construction*

$\longrightarrow$ *direct conformity with security notions avoids false negatives*

SILVER
is an easy-to-use tool for sound and complete verification
of the security and composability of masked implementations.

Conclusion

Open Challenges and Future Work

1. **HANDLING COMPLEXITY**

   → *verification of larger circuits and complex systems*

   → *verification of higher-order security*

2. **IMPROVING SECURITY NOTIONS**

   → *efficient construction of gadgets (randomness, area, latency, performance, power, …)*

3. **NON-CRYPTOGRAPHIC HARDWARE DESIGNS**

   → *verification of micro-controllers, CPUs, etc.*

4. **BEYOND FORMAL VERIFICATION**

   → *automated generation of secure designs*

   → *automated repairing and optimization of designs (computer-aided design)*

5. **AND MUCH MORE…**

FURTHER DETAILS

Source code, documentation, contact details, references

Further Details

SOURCE CODE & TOOL:

https://github.com/Chair-for-Security-Engineering/SILVER

DOCUMENT:

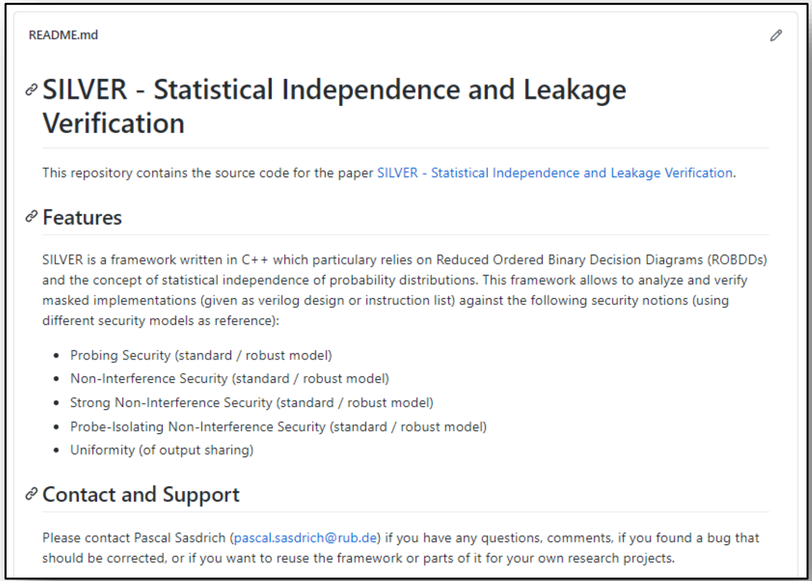
https://eprint.iacr.org/2020/634.pdf

CONTACTS:

David Knichel (david.knichel@rub.de)

Pascal Sasdrich (pascal.sasdrich@rub.de)

Amir Moradi (amir.moradi@rub.de)



README.md

SILVER - Statistical Independence and Leakage Verification

This repository contains the source code for the paper SILVER - Statistical Independence and Leakage Verification.

Features

SILVER is a framework written in C++ which particulary relies on Reduced Ordered Binary Decision Diagrams (ROBDDs) and the concept of statistical independence of probability distributions. This framework allows to analyze and verify masked implementations (given as verilog design or instruction list) against the following security notions (using different security models as reference):

- Probing Security (standard / robust model)
- Non-Interference Security (standard / robust model)
- Strong Non-Interference Security (standard / robust model)
- Probe-Isolating Non-Interference Security (standard / robust model)
- Uniformity (of output sharing)

Contact and Support

Please contact Pascal Sasdrich (pascal.sasdrich@rub.de) if you have any questions, comments, if you found a bug that should be corrected, or if you want to reuse the framework or parts of it for your own research projects.

References

[ISW03] *Yuval Ishai, Amit Sahai, David A. Wagner: Private Circuits: Securing Hardware against Probing Attacks. (CRYPTO 2003: 463-481)*

[BBD+15] *Gilles Barthe, Sonia Belaïd, François Dupressoir, Pierre-Alain Fouque, Benjamin Grégoire, Pierre-Yves Strub: Verified Proofs of Higher-Order Masking. (EUROCRYPT 2015: 457-485)*

[BBD+16] *Gilles Barthe, Sonia Belaïd, François Dupressoir, Pierre-Alain Fouque, Benjamin Grégoire, Pierre-Yves Strub, Rébecca Zucchini: Strong Non-Interference and Type-Directed Higher-Order Masking. (CCS 2016: 116-129)*

[BDF+17] *Gilles Barthe, François Dupressoir, Sebastian Faust, Benjamin Grégoire, François-Xavier Standaert, Pierre-Yves Strub: Parallel Implementations of Masking Schemes and the Bounded Moment Leakage Model. (EUROCRYPT 2017: 535-566)*

[FGP+18] *Sebastian Faust, Vincent Grosso, Santos Merino Del Pozo, Clara Paglialonga, François-Xavier Standaert: Composable Masking Schemes in the Presence of Physical Defaults & the Robust Probing Model. (IACR TCHES 2018(3): 89-120)*

[BBC+19] *Gilles Barthe, Sonia Belaïd, Gaëtan Cassiers, Pierre-Alain Fouque, Benjamin Grégoire, François-Xavier Standaert: maskVerif: Automated Verification of Higher-Order Masking in Presence of Physical Defaults. (ESORICS 2019: 300-318)*

[CS20] *Gaëtan Cassiers, François-Xavier Standaert: Trivially and Efficiently Composing Masked Gadgets With Probe Isolating Non-Interference. (IEEE Transactions on Information Forensics & Security 15, 2020: 2542-2555)*

Thank you!

pascal.sasdrich@rub.de

Chair for Security Engineering
Electrical Engineering & Information Technology
Ruhr University Bochum