12.03.2019

The Pitfalls of Threshold Cryptography in Hardware

Marco Macchetti, Karine Villegas, Claudio Favi

© 2019 KUDELSKI GROUP - All rights reserved

Outline

- Brief review of relevant theoretical models
- The reality of HW implementations
  - Conceptual vs. real circuit
  - Designer’s perspective
  - Automated synthesis tools
- Possible issues and possible solutions
  - Parallel computation on shares
  - Logic reuse
  - Control signals
- Conclusions and problem statements

NAGRA KUDELSKI GROUP

Brief Literature overview

- *Towards Sound Approaches to Counteract Power-Analysis Attacks*
- Seminal paper by Chari et al. @ CRYPTO'99
- Introduction of the noisy leakage model
- Highlights the difference between ad-hoc countermeasures against SCA and provably secure ones
  - Ad-hoc: shuffling, "dual" logic, current filtering, shields, etc…
  - Provable: secret sharing on ***d*** random shares (against a ***d-1*** adversary)
- Derives bounds on the distinguishing power of a differential attacker in terms of number of samples
- Samples leak information on all shares, but in a noisy way

NAGRA KUDELSKI GROUP

Brief Literature overview

- *Masking against Side-Channel Attacks:A Formal Security Proof*
- Paper by Prouff and Rivain @ EUROCRYPT 2013
- Extends and builds upon Chari's paper
- Chari makes static analysis while here the analysis is on computations
- A basic assumption is that computations are split in basic computations which are performed sequentially (e.g. CPU instructions)
- Obtain bounds on adversarial advantages for full computations
- Chari : adversary observes all shares with noise
  - proves lower bound on samples, meaning the adversary can always succeed but needs a certain number of traces

Brief Literature overview

- *Private Circuits: Securing Hardware against Probing Attacks*
- T-probing model : introduced by Ishai et al. @ CRYPTO ‘03
- Attacker has access to at most ***t*** wires of the circuit at each “time period” (e.g. clock cycle)
- Access via physical probes
- Assumed costly to switch position of probes, but possible between different “time periods”
- Cost also increases with number of probes
- Obtains lower bounds on how big a circuit with ***n*** gates must become to be resistant against ***t*** probes ⇨ $O(nt^2)$

NAGRA KUDELSKI GROUP

Brief Literature overview

- *Unifying Leakage Models: from Probing Attacks to Noisy Leakage*
- Paper by Duc et al. @ EUROCRYPT 2014
- The two previous leakage models have been then shown to be related by reducing security in one model to security in the other one
- “A ***t***-order (noisy) SC attack is equivalent to placing ***t*** probes on the circuit”
- Aims at unification of leakage models to simplify analysis of countermeasures
- But we have to remind the basic assumptions of these papers
  - This is true in the considered models

NAGRA KUDELSKI GROUP

Brief Literature overview

- It has then become common to refer to a first order SC attack as equivalent to placing a single probe in the circuit
- *Masking AES with just two random bits*, Gross et al. 2018

> For convenience reasons, the security is often expressed in the so-called t-probing model [23] which assumes that an attacker can make up to t observations in the circuit (place up to t probe on the circuit). It has been verified in the past that this formal model accurately models the abilities of a differential side-channel analysis attacker that has access to noisy side-channel leakage traces [17]. We assume in the following a first-order attacker, i.e., an attacker that can place a single probe on the circuit.

NAGRA KUDELSKI GROUP

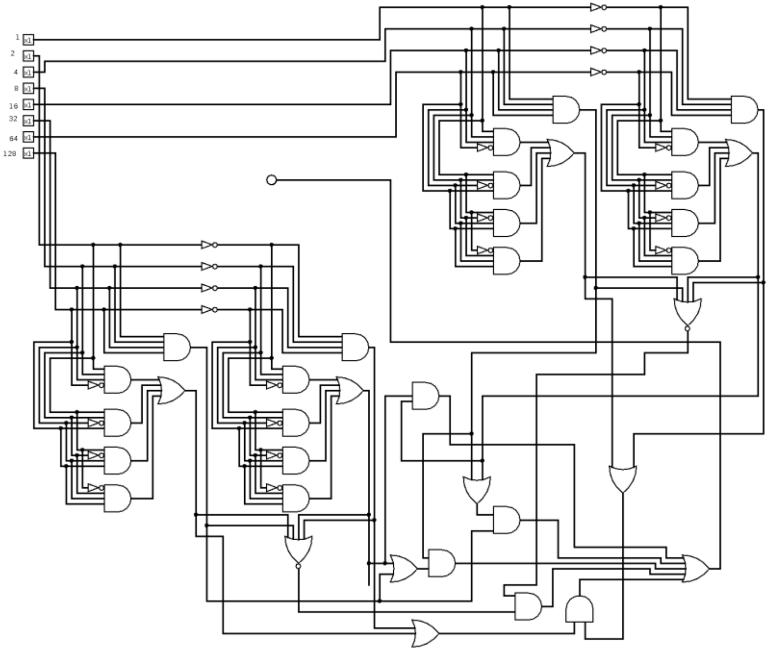
Discussion

- Is all this really closely modeling reality?
- In the models, a single probe means probing a single signal in the circuit
  - E.g. a single bit
- In reality even the smallest EM probe collects the leakage corresponding to many logic events in the circuit at the same time.
- What exactly do we mean when we talk about a *circuit* in these papers?
- How is it related to a real chip?
- How is the model related to a real attacker?

NAGRA KUDELSKI GROUP

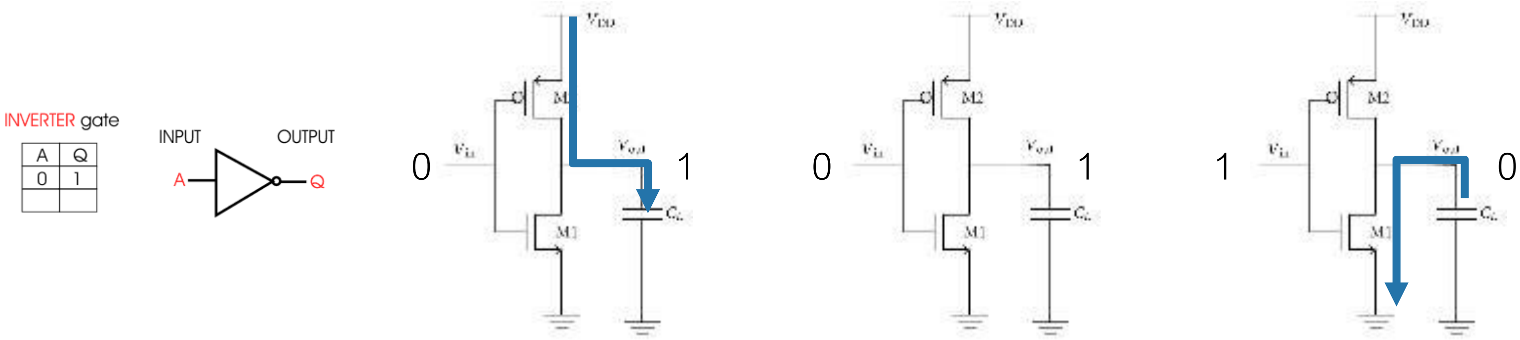
Circuit in complexity theory

- A Boolean circuit in computational complexity theory is a model of a digital circuit, consisting in a directed acyclic graph built of bounded-fan-in AND, OR and NOT gates.
- E.g. a conceptual circuit scheme
- Suppose we *prove* some statement about this circuit.
- Will it still hold for the manufactured circuit on a real silicon chip?



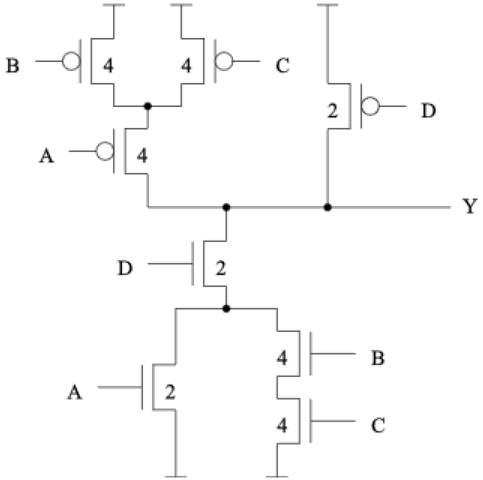
Side Channel Leakage

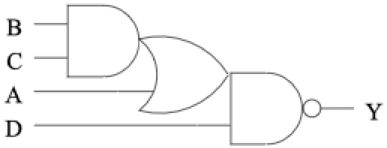
- Every circuit computing a given function consumes energy
  - c = f(a, b)
- In CMOS digital logic, basic units are transistors, used as switches
  - First approximation: energy is consumed when switches change state (0 ⇨ 1, 1 ⇨ 0)
  - Energy necessary to charge / discharge capacitances



Circuits in reality

- Real logic gates differ from ideal Boolean gates
  - They can be more complex (compound)
  - Or they can be simpler (limited fan-in)
  - They can have additional hidden variables

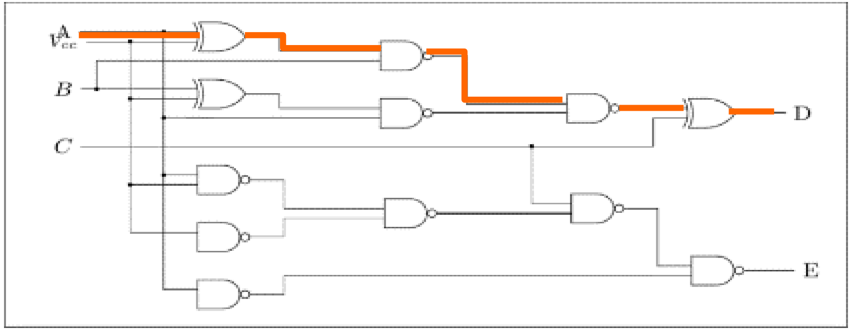




NAGRA KUDELSKI GROUP

Circuits in reality

- Different timing characteristics imply different switching activity patterns, which imply different side channel leakages
- Switching activity cannot be determined without examining the previous state of the circuit.
  - Reset? Previous vector? Random?
- Which gates switch at the same time? One? A single layer? All?
  - Has also to do with sampling rate of the attacker…



NAGRA KUDELSKI GROUP

Circuits in reality

- Models often consider a precise sequence of computation in a circuit
- Almost implicit in papers about HW threshold schemes
- But in HW implementations, this is often not the case
  - One explicit goal of specialized HW is to parallelize in order to speed up w.r.t software implementations
  - Designer may not even be aware of the precise order of gate switching

> An algorithm is modelled by a sequence of *elementary calculations* $(\mathsf{C}_i)_i$ that are Turing machines augmented with a common random access memory called *the state*. Each elementary calculation reads its input and writes its output on the state. When an elementary calculation C_i is invoked, its input is written from the state to its input tape, then C_i is executed, afterwards its output is written back to the state.

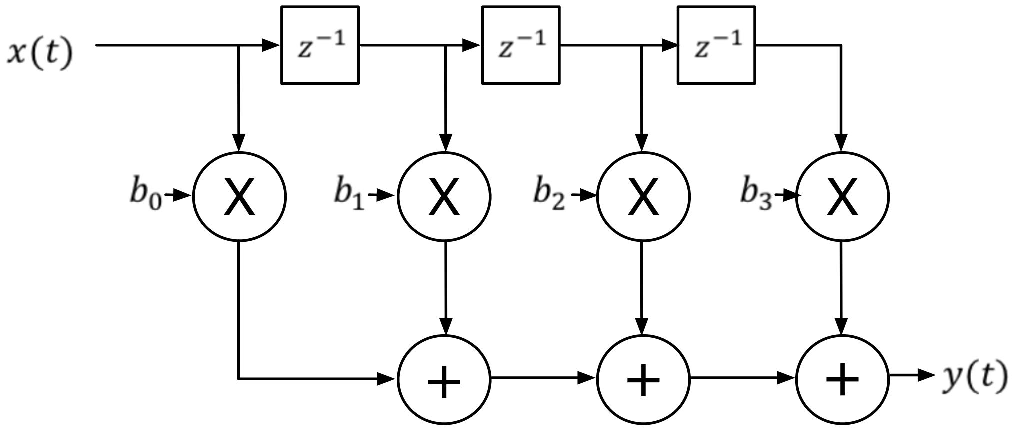
NAGRA KUDELSKI GROUP

Reality of HW Designer's job

- The HW designer not only has to consider security constraints, but also timing, power and area ones.
- Designer examines scientific literature to find a suitable method which
  - Meets (in practice!) security robustness expectations (often difficult to quantify e.g. number of traces? Order of attack?
  - Meets all other constraints (area / performance / power consumption)
- Circuit is designed, functionally verified and taped-out
- Chip is manufactured
- Chip undergoes security lab evaluation
- If weaknesses are found -> need to analyze, fix, and iterate (if possible!)
- Full cycles can take up to 1-2 years

Reality of HW Designer's job

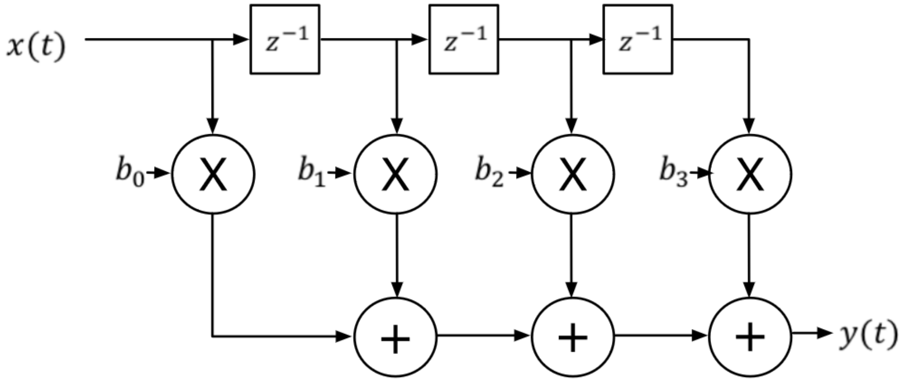
- Fact #1: HW designer works at RTL level of abstraction.
- He starts from a conceptual circuit and applies his skills to derive the best circuit architecture under all constraints.
- This is a rather high-level representation, equivalent to a high level language (e.g. C++) for SW

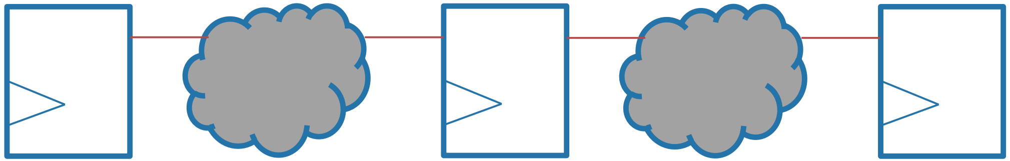


NAGRA KUDELSKI GROUP

Reality of HW Designer's job

- HW designer can apply several architectural design patterns to an ideal circuit (RTL):
  - Round loops
  - Unrolling
  - Pipelining
  - resource sharing
  - Clock gating
  - etc...



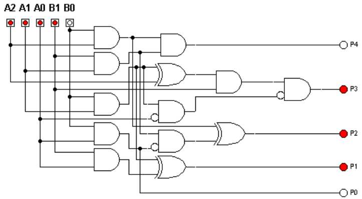


NAGRA KUDELSKI GROUP

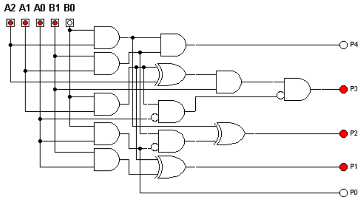
Reality of HW Designer’s job

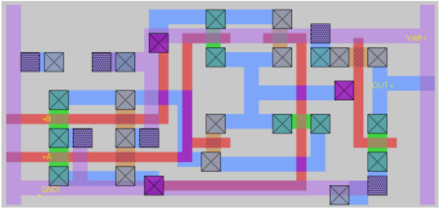
- Fact #2 The synthesis flow then derives the real design in terms of silicon library cells….

```
c <= a * b;
```



- And layout tools then derive the actual circuit topology on silicon





- Equivalent to a SW toolchain C++ ⇨ compiler ⇨ object files ⇨ linker ⇨ binary code

NAGRA KUDELSKI GROUP

Reality of HW Designer’s job

- Synthesis tools can apply a wide range of optimizations automatically, much like a compiler optimizes C code
- Register sharing (different variables are mapped to the same register, or derived as a Boolean function of another register)
- Combinational logic reuse
    - E.g. a XOR logic is reused on different variables at different cycles.
- Register re-timing
    - boolean logic is moved/split across a register
- Constant optimization
    - logic optimizations are pushed up to the next register
- Etc…

NAGRA KUDELSKI GROUP

Questions

- With previous items in mind, is it possible to make a claim about a real chip adhering to a given theoretical model?
- In papers about HW implementations of threshold cryptography, circuit schemes are often given as a reference for HW implementation.
- Are the proposed schemes to be intended as idealized or real circuits? Before or after optimizations?
- Is it possible to deviate from the reference schemes by using the optimizations discussed above?

Paper examples

- *Masking AES With d+1 Shares in Hardware*
  - By Rijmen et al. @ CHES 2016
- *A more efficient AES Threshold Implementation*
  - By Rijmen et al. AFRICACRYPT 2014

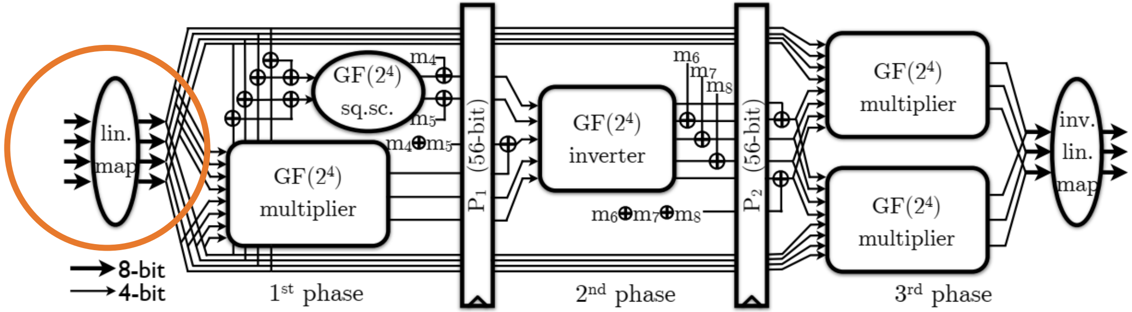


Fig. 2: The Sbox of our implementation.

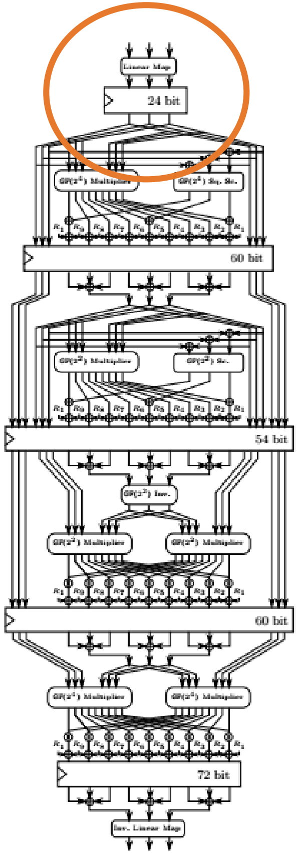


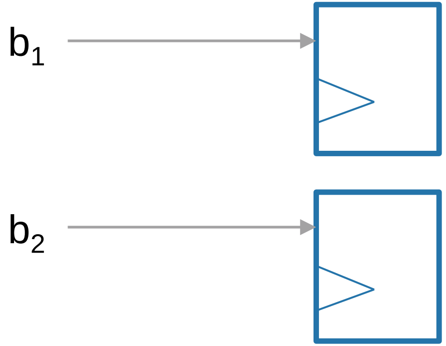
Fig. 3: Structure of the second-order TI of the AES S-box

NAGRA KUDELSKI GROUP

Examples of issues

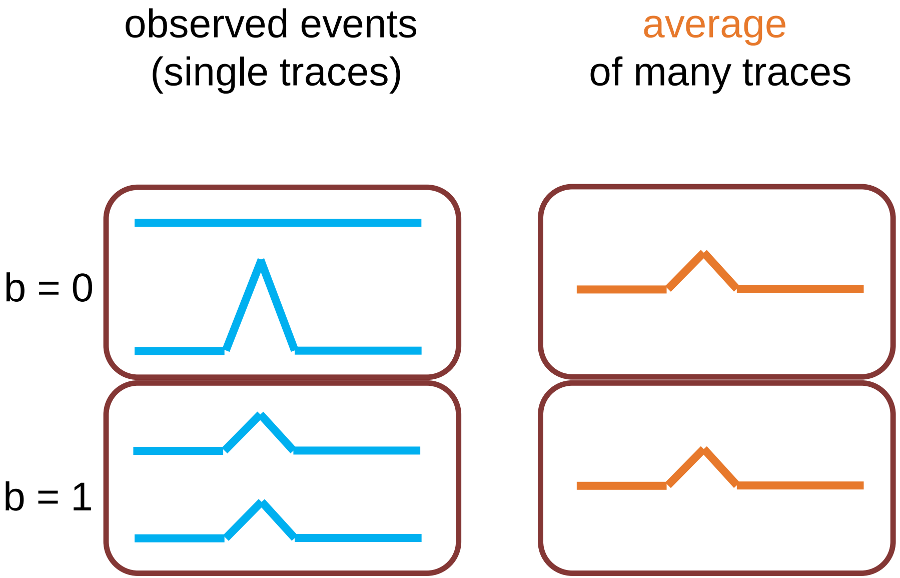
- Why could we have problems?
- Simple model: single bit is split in two Boolean shares $\boldsymbol{b} \rightarrow (\boldsymbol{b_1}, \boldsymbol{b_2})$
  - First order resistant
- parallel transfer of all shares to registers, which were previously reset to 0

simplest case to analyze, no computation



CMOS dynamic power consumption is due to changing state of logic cells (first order approximation)

Examples of issues



b	b1	b2	Power consumption
0	0	0	0
0	1	1	2
1	0	1	1
1	1	0	1

<table>
<tr><th>b</th><th>b1</th><th>b2</th><th>Average consumption</th></tr>
<tr><td>0</td><td>0</td><td>0</td><td rowspan="2">1</td></tr>
<tr><td>0</td><td>1</td><td>1</td></tr>
<tr><td>1</td><td>0</td><td>1</td><td rowspan="2">1</td></tr>
<tr><td>1</td><td>1</td><td>0</td></tr>
</table>

NAGRA KUDELSKI GROUP

Examples of issues

- Attacker who looks at average of traces (1st order) is incapable of extracting information on ***b***
- However, observing a single trace, it is trivial to obtain ***b***
  - 0th order or SPA
- Simple SPA inspection or machine learning would trivially break the implementation
- The problem is inherently due to the fact that all shares are manipulated in parallel

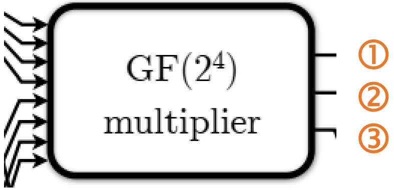
Examples of issues

- Is it a violation of the proposed models?
  - Sequential calculation of shares…
- It is not detectable by attacks which just look at average of trace sets
- It is really a 0th order problem
- The problem is mitigated by noise, but can persist when many bits are manipulated in parallel
  - Example, a null-coordinate-point arising from ECC scalar-point multiplication, which is simply blinded by 1 bit and split in two shares
  - 0x0000000000000 or 0xFFFFFFFFFFFFFF…
- High speed HW threshold implementations can be sensitive to machine learning / SPA / template attacks

Examples of issues

- To solve the problem, we can pre-load the registers with random values
- This is OK for sequential logic
- But what about combinational logic? Looks like an extremely complex problem in the generic instance (timing/activity/logic cells)
- Generic solution: never manipulate all shares at the same time
  - easy for linear functions ⇨ compute independently on single shares
  - Non-linear: they are already shared, but compute single bits individually

first-order resistant
4-shares input
3-shares output
GF(2^4) multiplier



Examples of issues

- Another example: Boolean to arithmetic masking switching algorithm, proposed by Goubin in 2001
- Proven to be first order resistant, under the implicit assumption of a sequential SW implementation.
- Potential problem if logic resources are shared between algorithm steps
- Poses hard constraints on possible circuit, to stick to the model

Algorithm 1. BooleanToArithmetic

Require: (x', r) such that $x = x' \oplus r$

Ensure: (A, r) such that $x = A + r$

Initialize Γ to a random value γ

$T \Leftarrow x' \oplus \Gamma$

$T \Leftarrow T - \Gamma$

$T \Leftarrow T \oplus x'$

$\Gamma \Leftarrow \Gamma \oplus r$

$A \Leftarrow x' \oplus \Gamma$

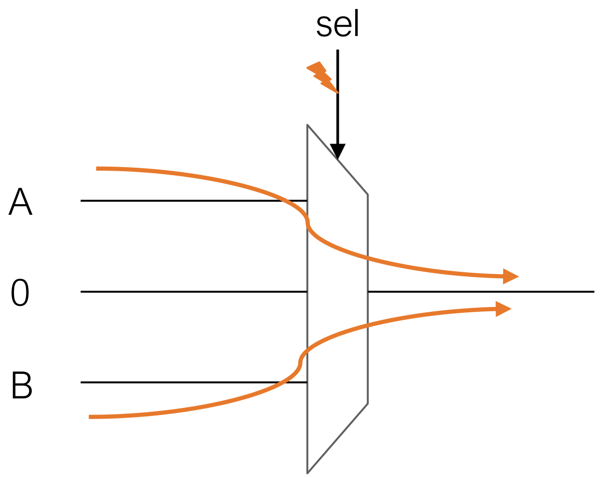
$A \Leftarrow A - \Gamma$

$A \Leftarrow A \oplus T$

NAGRA KUDELSKI GROUP

Examples of issues

- Third example: multiplexer with glitchy selector



- Control path must also be inspected as can be source of problems as well as data path

Conclusions

- Models are good, but they do not always fully adhere to reality
- Hard lesson : **models are never complete**
- Generic solutions for HW implementations:
  - Always pre-charge registers with random values
  - Always register control signals
  - Always compute sequentially on single shares
- Of course, all this has **severe impact on performance**.

NAGRA KUDELSKI GROUP

Future work

- Two open problems statements:
- A rather complex one for future research:
  - Devise an high-speed HW threshold implementation which is also resistant against all attacks under a certain order, including profiled attacks (templates, machine-learning, etc…)
- A (really?) less complex one related to standardization:
  - At which level should HW threshold schemes be described in the standard(s) and at which level should we certify?
  - Regarding HW, should we formalize requirements or standardize techniques for circuit implementation/optimization?

NAGRA KUDELSKI GROUP

Thank you!