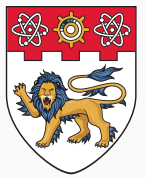
Peek into the Black-Box: Interpretable Neural Network using SAT equations in Side-Channel Analysis

Trevor Yap, Adrien Benamira, Shivam Bhasin, Thomas Peyrin

11 September, 2023

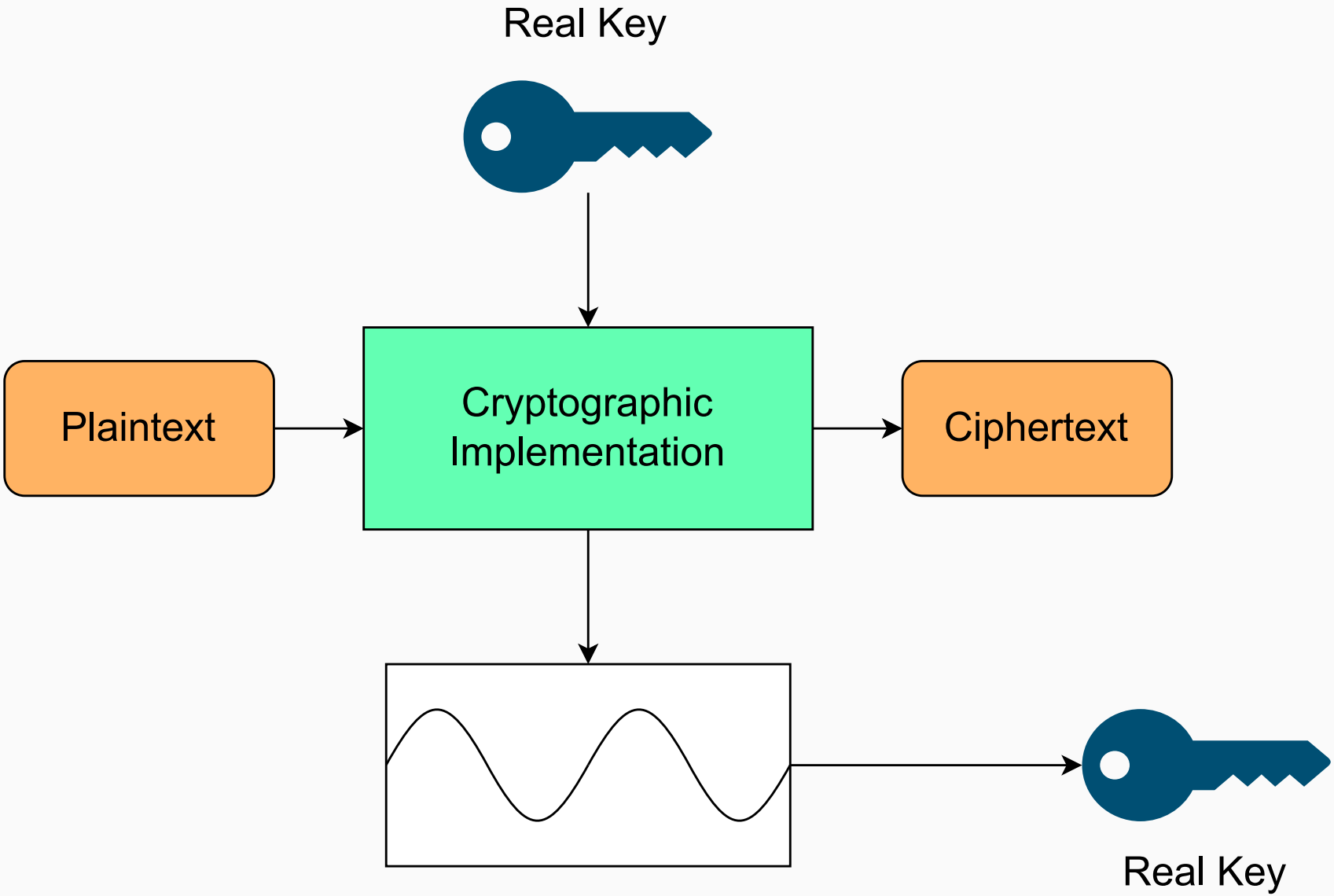


NANYANG TECHNOLOGICAL UNIVERSITY SINGAPORE

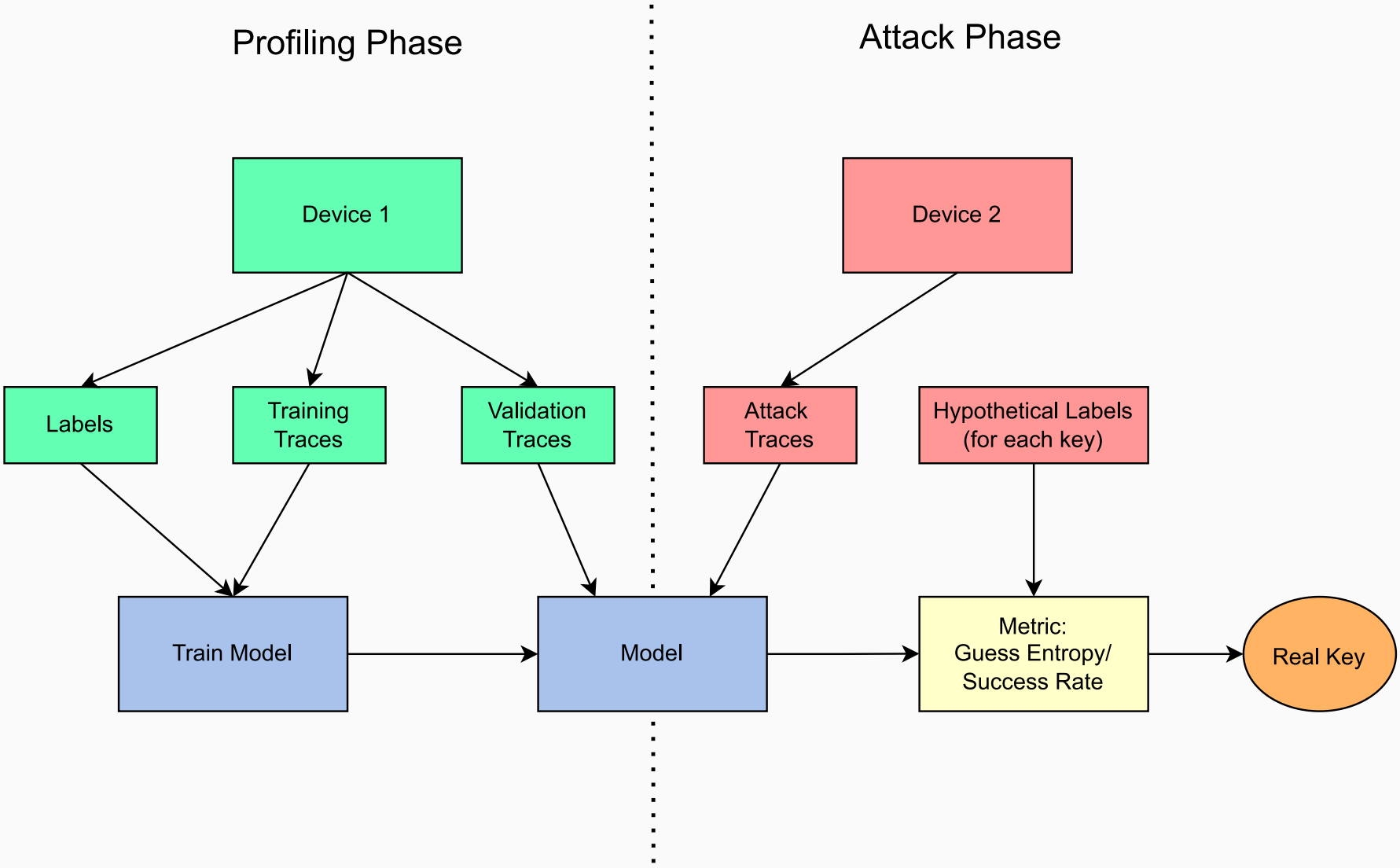
Table of contents

Table of contents

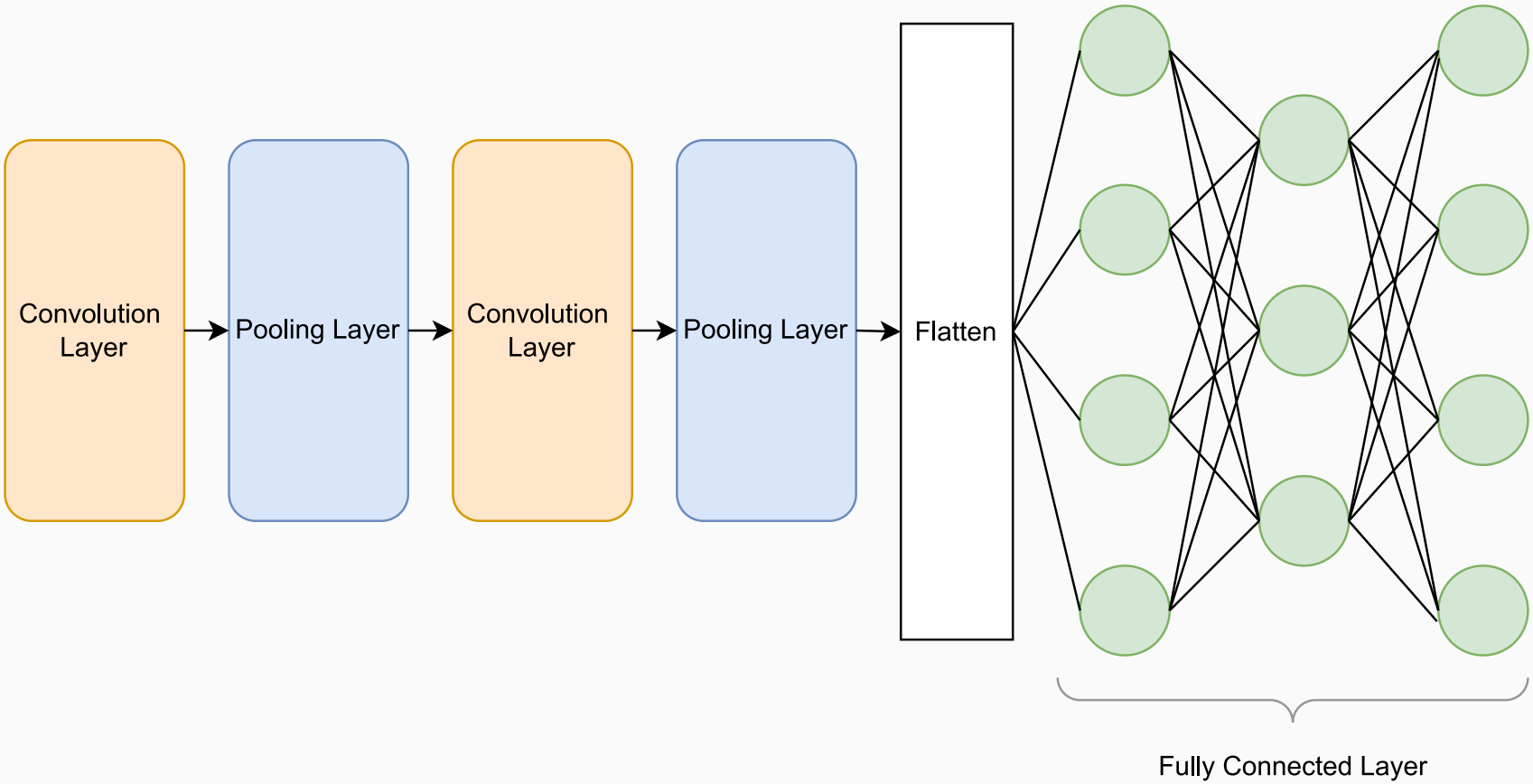
Overview of side channel analysis



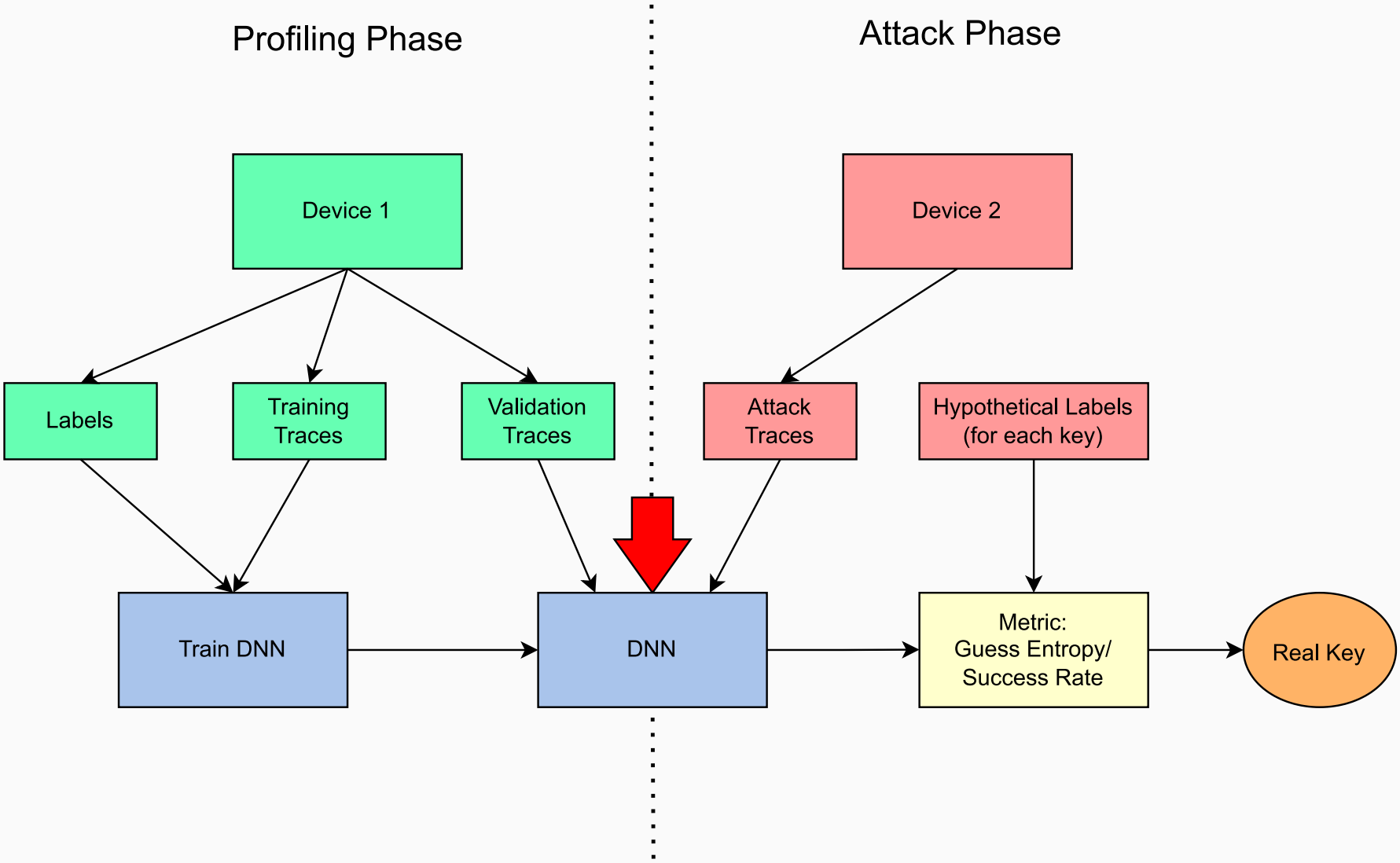
Profiling attack



Deep neural network (DNN)



Profiling attack with DNN



Advantage of using DNN-based SCA

Classic SCA (i.e. Template Attack)	DNN-based attack
More traces to attack	Significantly lesser traces to attack
Preprocessing to remove any desynchronization in the traces	No preprocessing needed
Preprocessing to obtain Points of Interest (PoIs)	No preprocessing needed

Disadvantages of DNN

- Currently, Deep Neural Networks (DNNs) are seen as a **black-box tool**.
- The DNNs have far **more hyperparameters** to tune compared to classical SCA techniques.
- Unable to tell **which architecture to use**.
- Presently, auto-tune methods to find hyperparameters and architecture are **time consuming**.
- The evaluator wants to help the developer **localize and understand where the vulnerability** comes from in order to remove or at least reduce it.

Problem:

Lesser works in interpretability/explainability of DNNs within SCA.

Explainability of DNN

We want to explore what the neural network is learning in side-channel analysis.

Goal:

Propose a neural network that is **easy to interpret**.

The proposed architecture is **not for efficiency**.

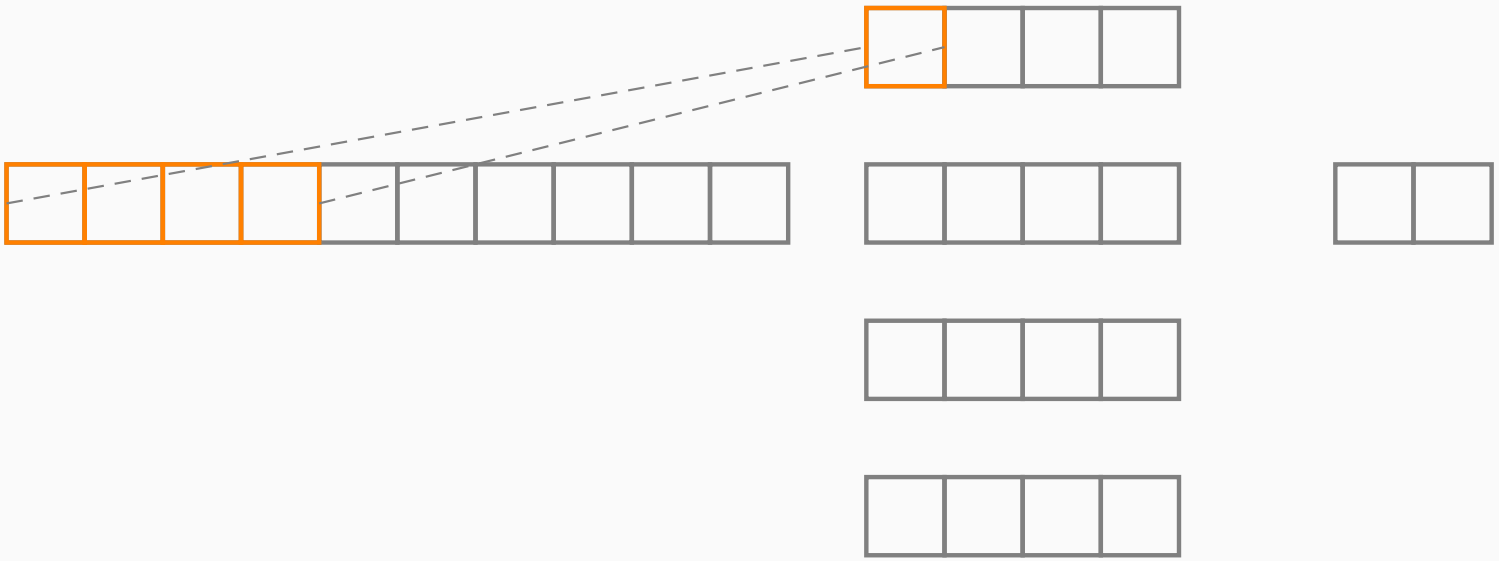
Table of contents

Neural network and truth table

Convolution of two layers.

Layer 1: kernel size $= 4$, stride $= 2$

Layer 2: kernel size $= 2$, stride $= 2$

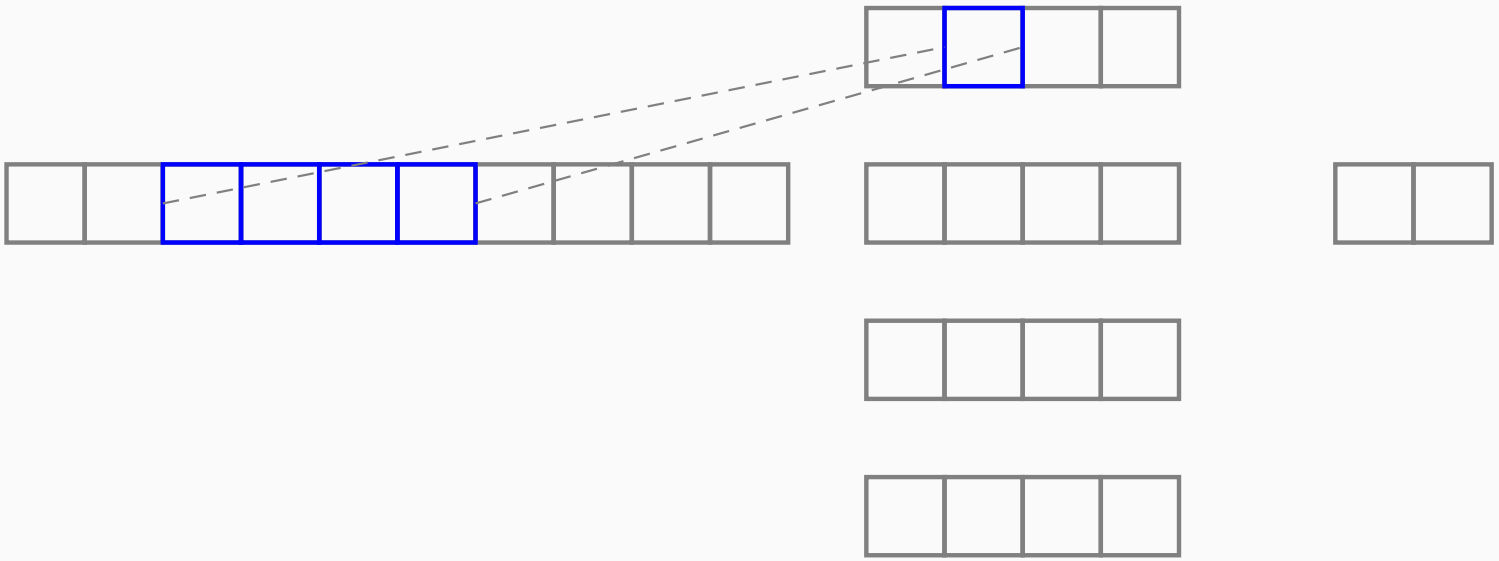


Neural network and truth table

Convolution of two layers.

Layer 1: kernel size $= 4$, stride $= 2$

Layer 2: kernel size $= 2$, stride $= 2$

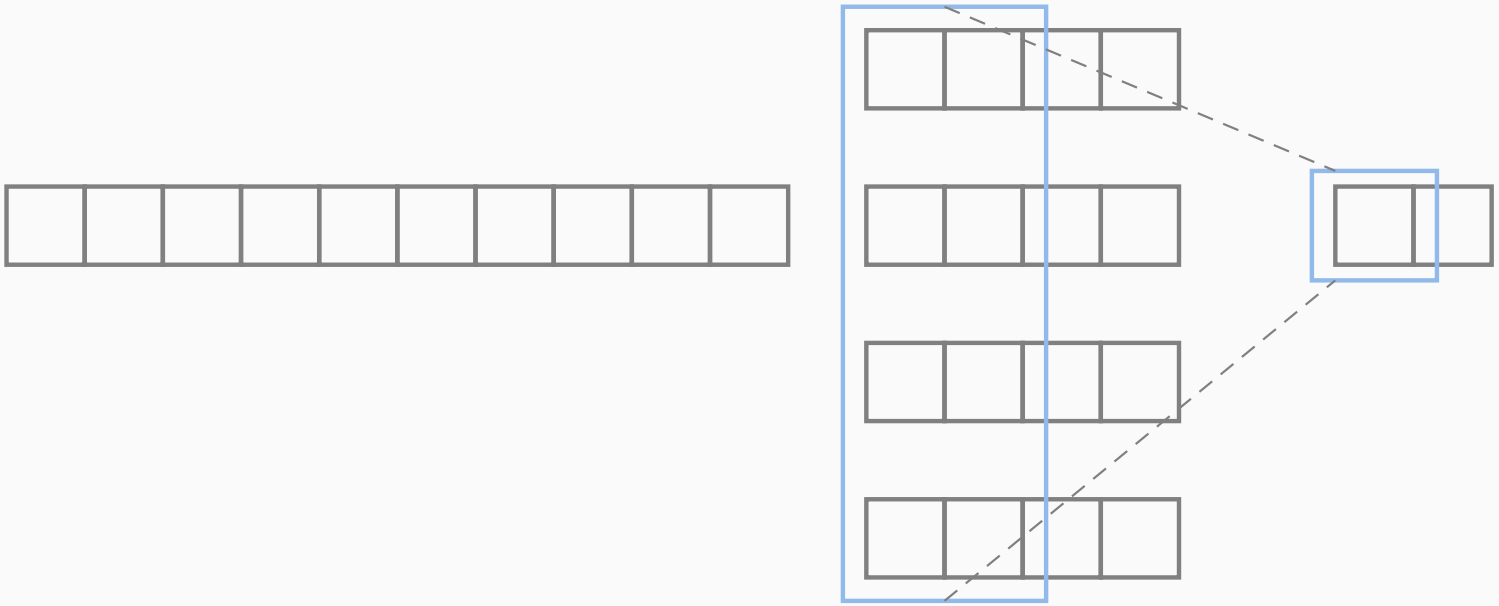


Neural network and truth table

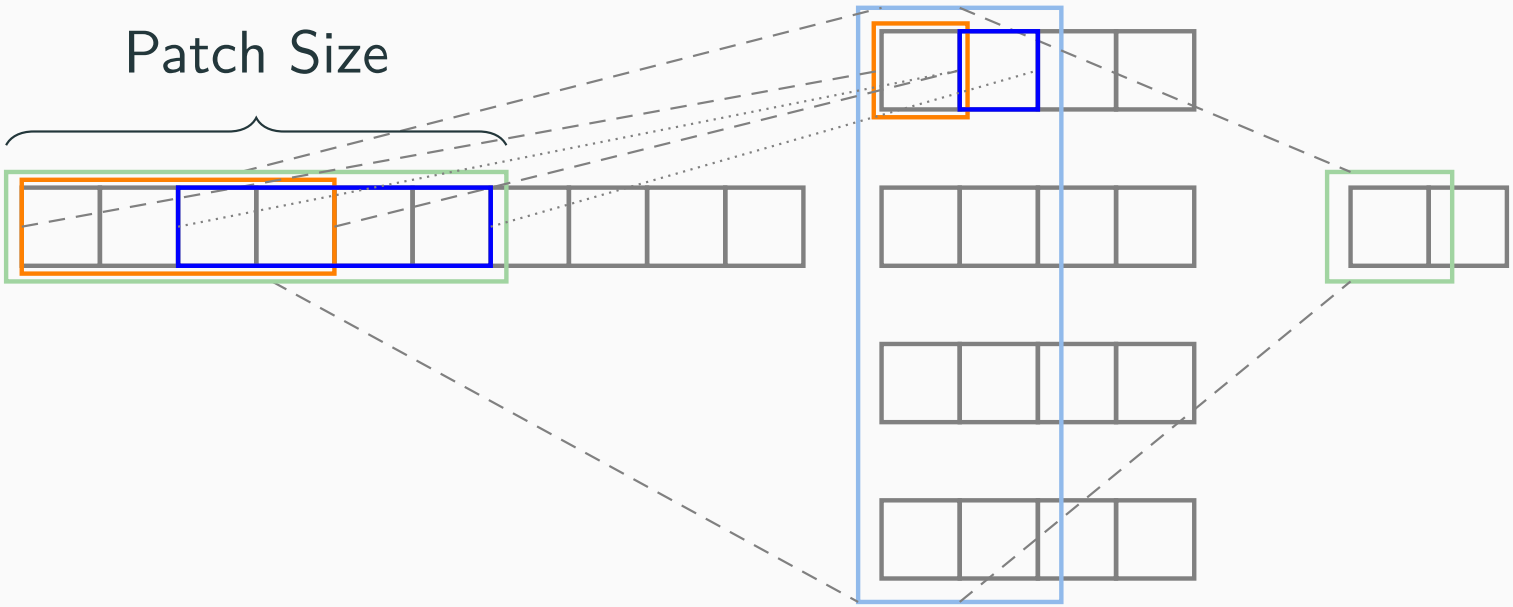
Convolution of two layers.

Layer 1: kernel size $= 4$, stride $= 2$

Layer 2: kernel size $= 2$, stride $= 2$

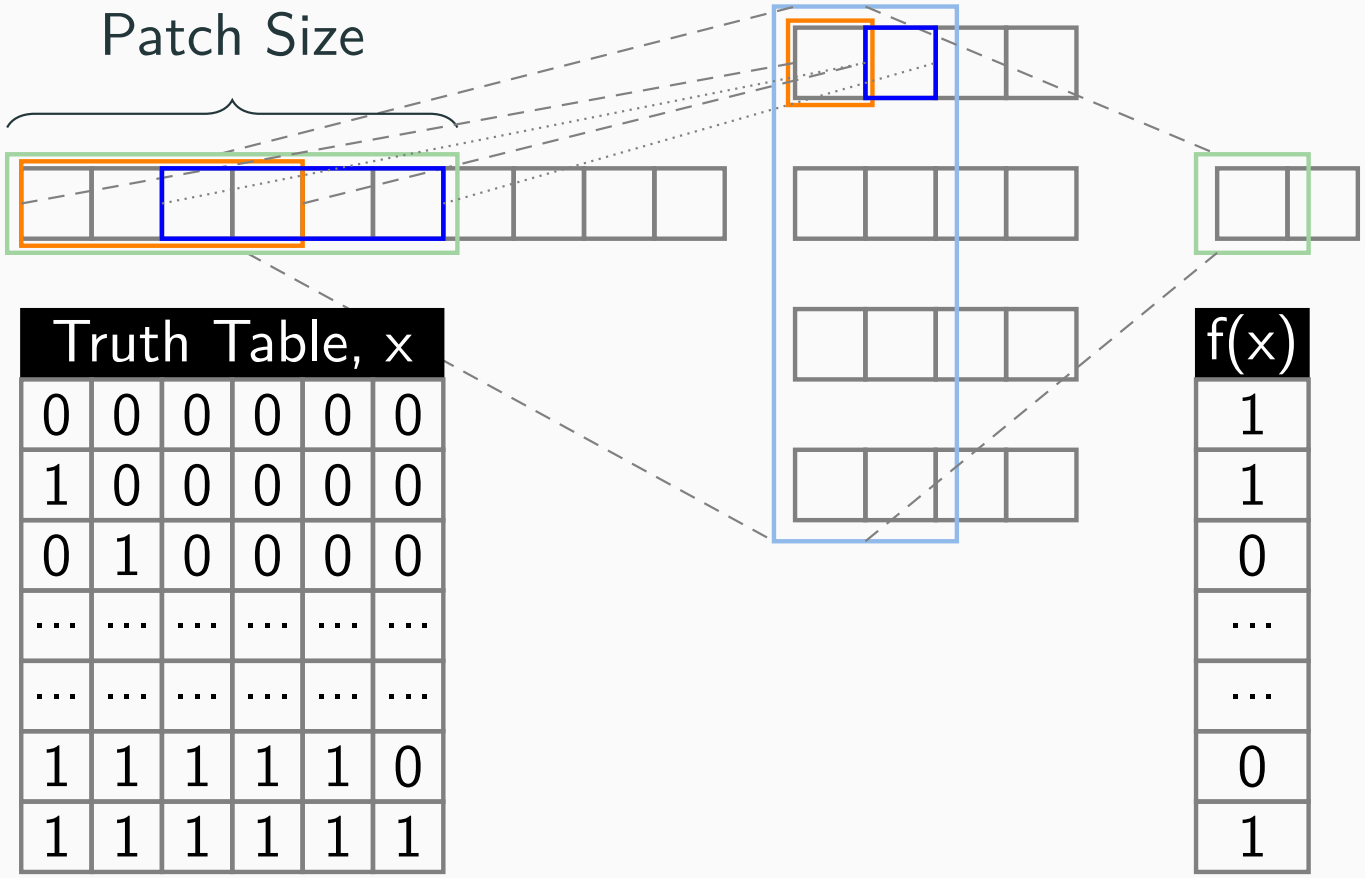


Neural network and truth table



Observe that first 6 features of the input is used to calculate the first output of the last layers.

Neural network and truth table



Binarized input and output.

- Can create a truth table by enumerating all possible input.
- This truth table can be converted into **SAT equations for interpretation**.

Truth table deep convolution neural Network (TT-DCNN)

- It is possible to interpret the Convolution layers as **SAT equations** from the truth table.
- Convert to **Disjunctive Normal Form (DNF)** using the Quine-Mccluskey Algorithm. (e.g. $(x_0 \wedge x_1) \vee (x_2 \wedge \neg x_1)$)
- Application of the TT-DCNN to SCA: Able to **pinpoint location on the Point of Interest** for the leakages in the traces and combine the leakages.

Table of contents

Heuristic used to analyse the SAT equations

We want to know which are the **important disjuncts/literals**, as most of the disjuncts are unnecessary for key recovery. We proposed three types of heuristic:

Heuristic used to analyse the SAT equations

We want to know which are the **important disjuncts/literals**, as most of the disjuncts are unnecessary for key recovery. We proposed three types of heuristic:

1. **Sieving** disjuncts based on their size
   (e.g. $(x_1 \wedge \neg x_3 \wedge x_5 \wedge \neg x_6)$ is a disjunct of size 4),

Heuristic used to analyse the SAT equations

We want to know which are the **important disjuncts/literals**, as most of the disjuncts are unnecessary for key recovery. We proposed three types of heuristic:

1. **Sieving** disjuncts based on their size (e.g. $(x_1 \wedge \neg x_3 \wedge x_5 \wedge \neg x_6)$ is a disjunct of size 4),
2. **Separating** disjuncts based on their combinations of literals (CoLs) (e.g. CoL of x_1, x_3, x_5, x_6, examples of disjuncts of these combination are $(x_1 \wedge \neg x_3 \wedge x_5 \wedge \neg x_6)$),

Heuristic used to analyse the SAT equations

We want to know which are the **important disjuncts/literals**, as most of the disjuncts are unnecessary for key recovery. We proposed three types of heuristic:

1. **Sieving** disjuncts based on their size
   (e.g. $(x_1 \wedge \neg x_3 \wedge x_5 \wedge \neg x_6)$ is a disjunct of size 4),
2. **Separating** disjuncts based on their combinations of literals (CoLs) (e.g. CoL of x_1, x_3, x_5, x_6, examples of disjuncts of these combination are $(x_1 \wedge \neg x_3 \wedge x_5 \wedge \neg x_6)$),
3. **Trimming** disjuncts based on the literals.
   (e.g. trimming the literal (x_3) then the $(x_1 \wedge \neg x_3 \wedge x_5 \wedge \neg x_6)$ become $(x_1 \wedge x_5 \wedge \neg x_6)$.)

Heuristic used to analyse the SAT equations

We want to know which are the **important disjuncts/literals**, as most of the disjuncts are unnecessary for key recovery. We proposed three types of heuristic:

1. **Sieving** disjuncts based on their size
   (e.g. $(x_1 \wedge \neg x_3 \wedge x_5 \wedge \neg x_6)$ is a disjunct of size 4),
2. **Separating** disjuncts based on their combinations of literals (CoLs) (e.g. CoL of x_1, x_3, x_5, x_6, examples of disjuncts of these combination are $(x_1 \wedge \neg x_3 \wedge x_5 \wedge \neg x_6)$),
3. **Trimming** disjuncts based on the literals.
   (e.g. trimming the literal (x_3) then the $(x_1 \wedge \neg x_3 \wedge x_5 \wedge \neg x_6)$ become $(x_1 \wedge x_5 \wedge \neg x_6)$.)

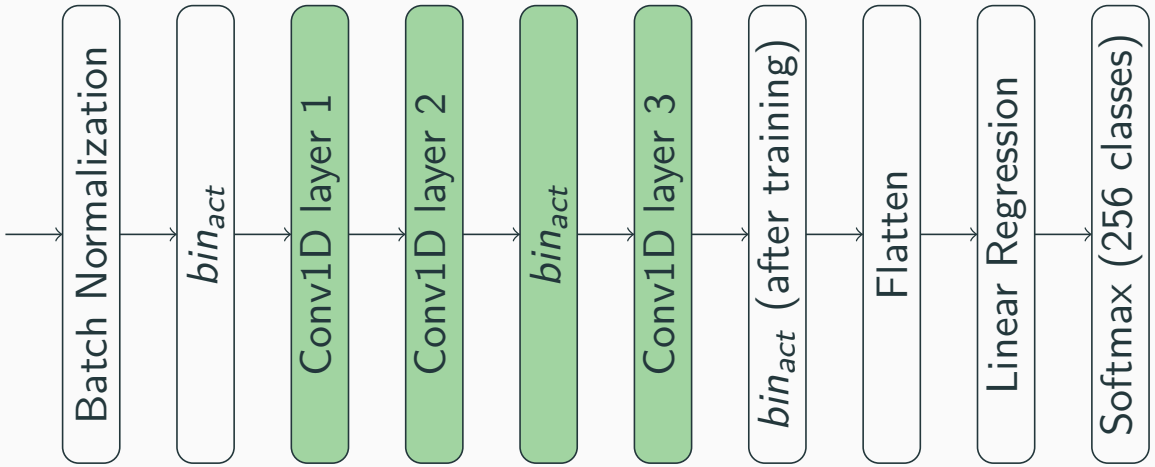
Goal:

Find the most miniature set of rules that the neural network needs for key recovery.

Table of contents

TT-DCNN architecture, $TTSCA_{small}$

We found out that the neural network overfits very fast due to it being a perfect world. Therefore, we proposed the following architecture called $TTSCA_{small}$.



$TTSCA_{small}$ architecture

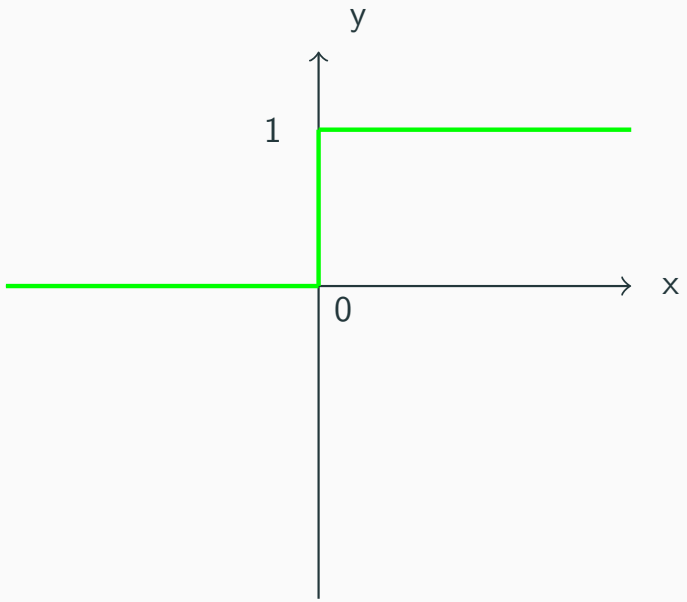


Fig: bin_{act}

Results of $TTSCA_{small}$ on simulated data with masking order 1

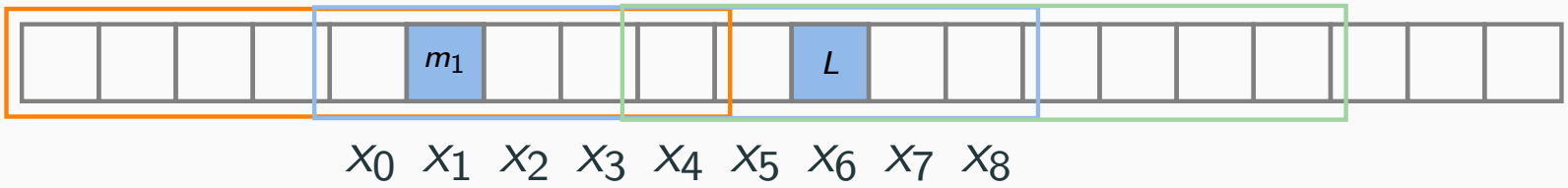
Masking

Supposed there are $d+1$ leakages points, $L_1, \ldots L_{d+1}$, then the secret variable, Z, is

$$Z = g(L_1, L_2, \ldots, L_{d+1}).$$

A common function g is the Boolean XOR of each leakage points (aka Boolean Masking).

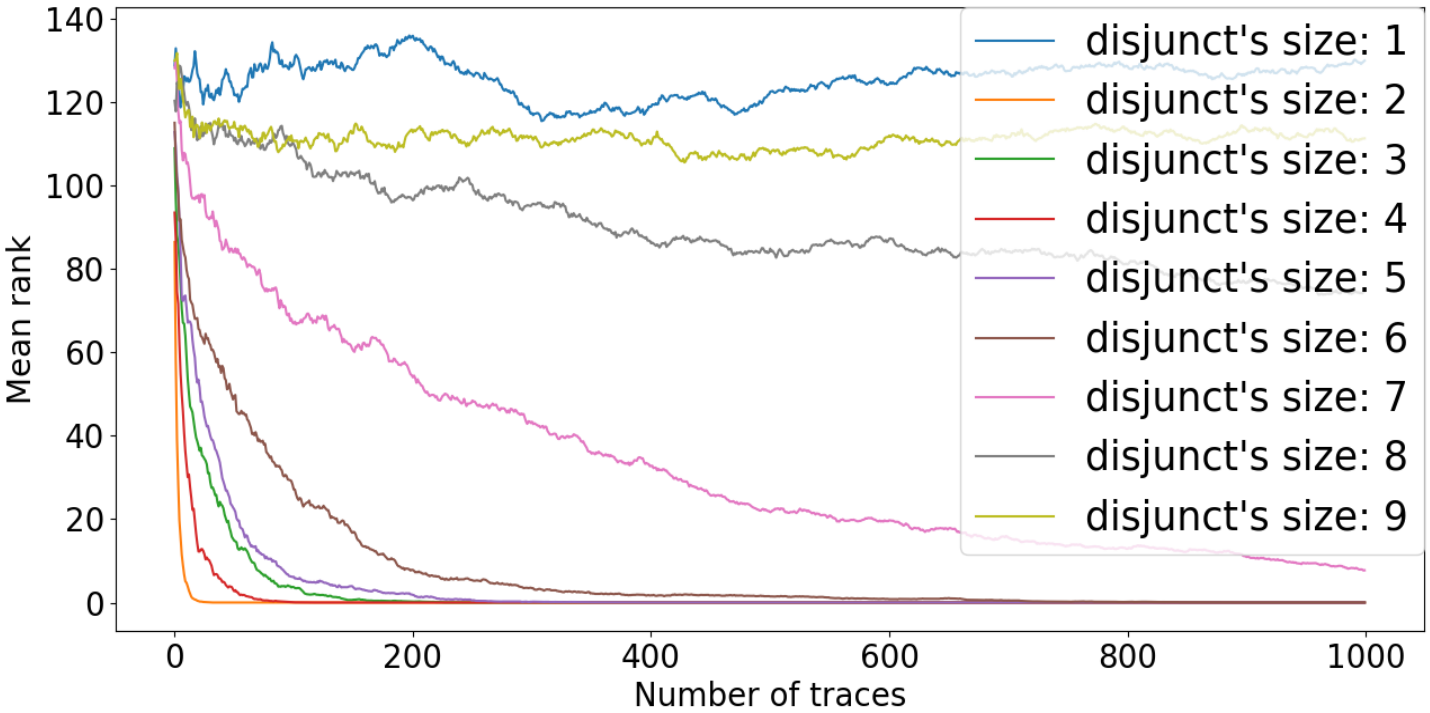
A visualization of the simulated traces are shown below and we observed that the neural network sees the following:



where $L = Z \oplus m_1$.

Results for simulated data with masking order 1

The Guess Entropy for order 1 for different sizes is as follows:

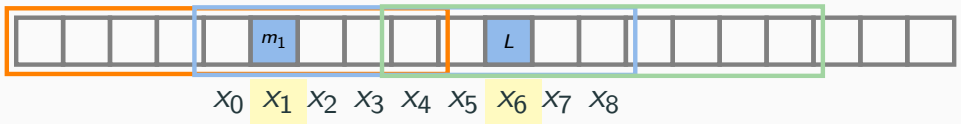


Conclusion: The disjuncts of size 2 is the smallest size with mean rank going to 0 (see orange line).

Results for simulated data with masking order 1

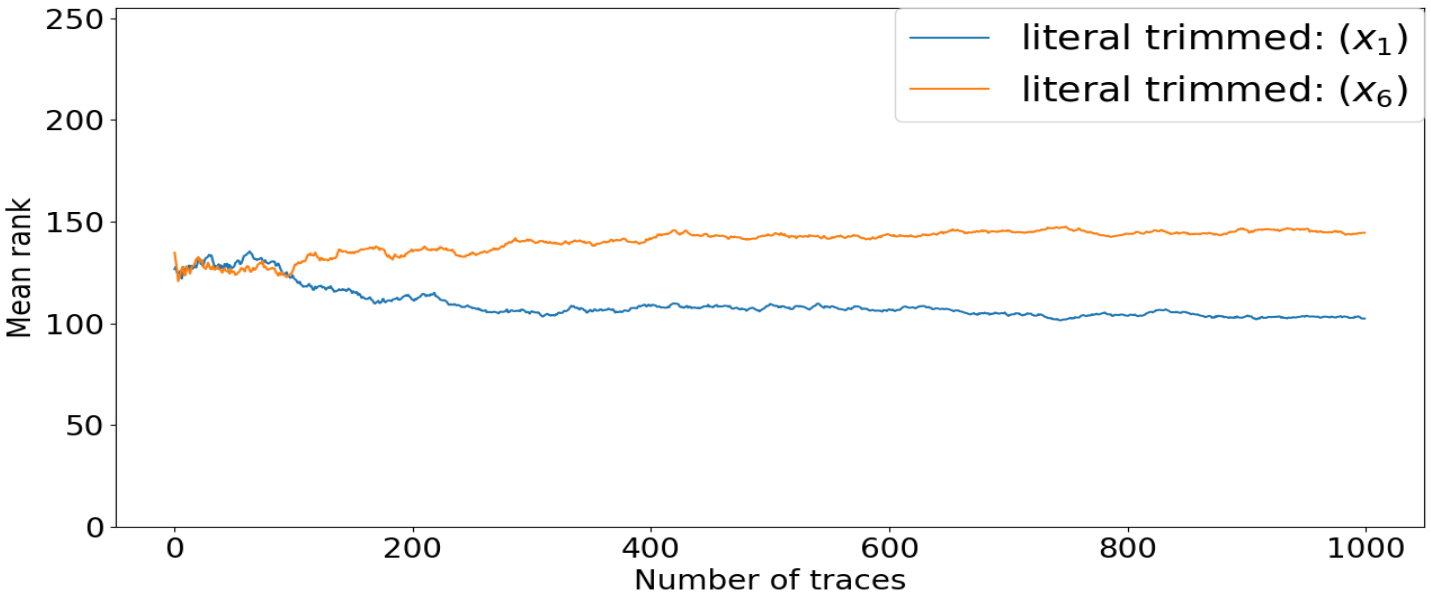
The disjuncts of size 2 are

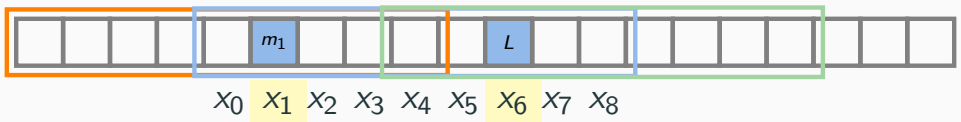
$$(x_1 \wedge \neg x_6), (x_6 \wedge \neg x_1), (\neg x_6 \wedge \neg x_1).$$



Results for simulated data with masking order 1

Since there is only one CoL; the CoL for (x_1, x_6). We trim based on (x_1) and (x_6) individually.

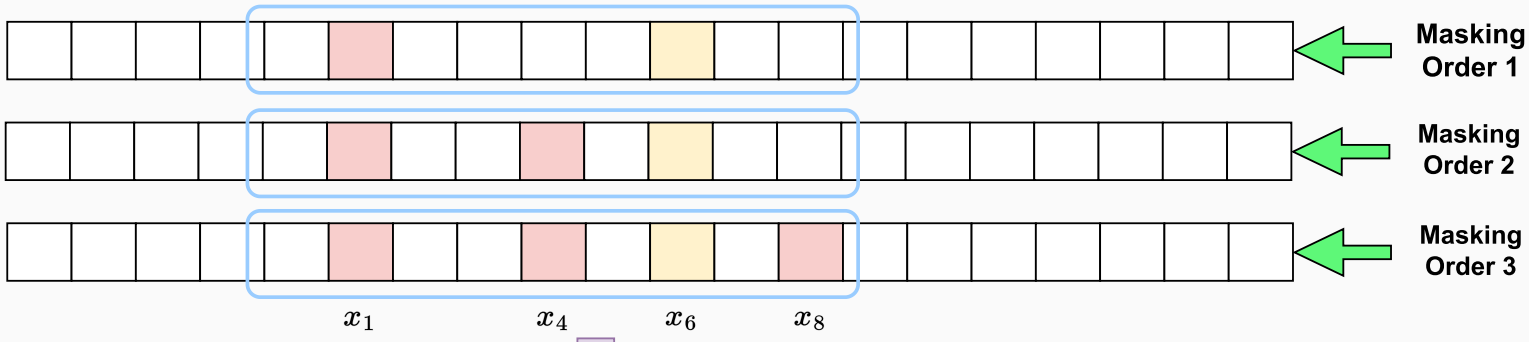




The most important literals are x_1 and x_6.

Results for simulated data with masking order 1

The $TTSCA_{small}$ can pinpoint the leakage's position and use it to retrieve the key.

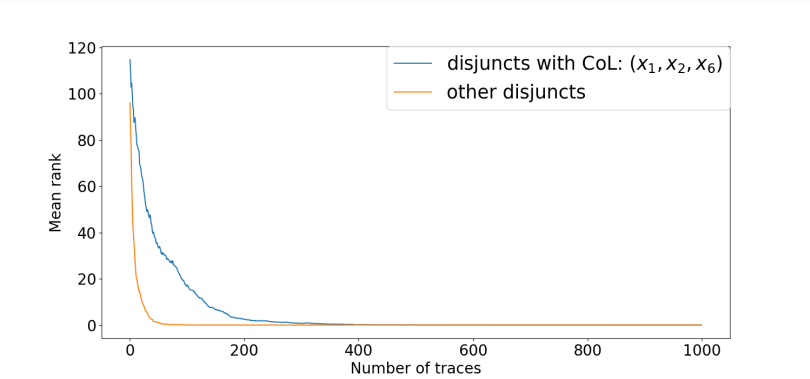


Masking Order	Recovered Disjuncts
1	$(x_1 \wedge \neg x_6)$
2	$(x_1 \wedge \neg x_4 \wedge \neg x_6)$
3	$(x_1 \wedge x_4 \wedge x_6 \wedge x_8)$

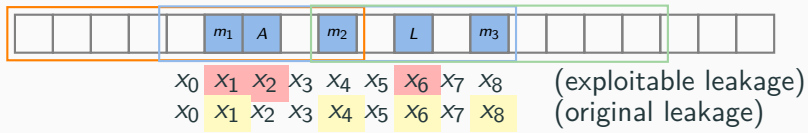
We obtained similar results for masking orders of $0, 2$ and 3.

Masking with Flaws

Suppose we consider simulated traces with masking order 3. Furthermore, we consider the existence of the leak $A = m_2 \oplus m_3$ at x_2.



(a) CoL of (x_1, x_2, x_6).

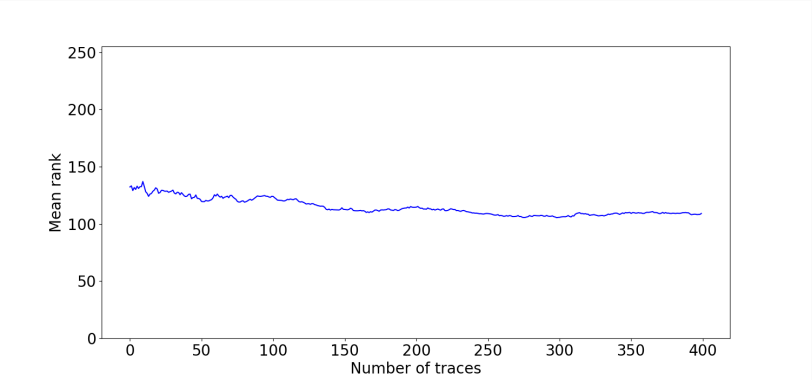


(b) Denote $L = Z \oplus m_1$ and $A = m_2 \oplus m_3$ the mask combination.

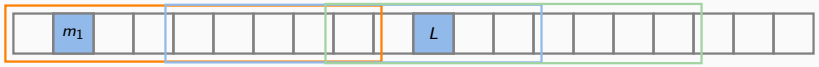
We observe that the $TTSCA_{small}$ is able to detect the exploitable leakage of x_1, x_2 and x_6 to recover the key (blue line in Figure (a)).

Further investigation

We train $TTSCA_{small}$ on new simulated traces of masking order 1; where we place m_1 and $L = Z \oplus m_1$ place at $trace[1]$ and $trace[10]$ respectively.



(a) Guessing entropy of $TTSCA_{small}$ trained on the new simulated traces of masking order 1.



(b) Orange, light blue and light green boxes shows that patch size when it is at timestamp $1, 2$ and 3 respectively. Denote $L = Z \oplus m_1$.

Conclusion: At any point in time, both m_1 and $L = Z \oplus m_1$ need to lie within the same patch for $TTSCA_{small}$ to recover the key.

Table of contents

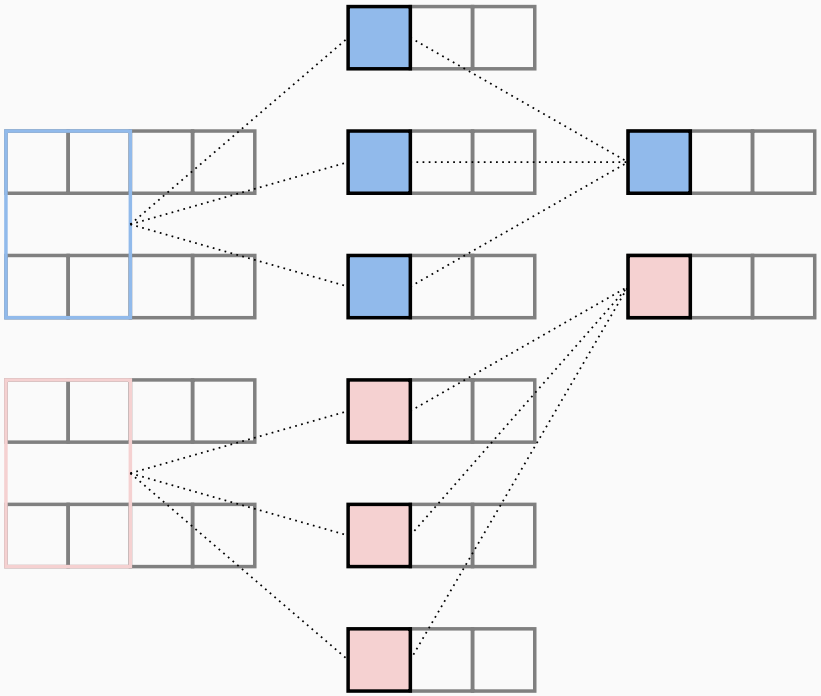
Challenges when finding an architecture

- Most obvious way to apply $TTSCA_{small}$ onto longer traces is to increase its patch size
- However, the size of the patch **cannot be more than 12**, due to solving an NP complete problem of simplifying SAT equations.
- Because of this **limitation on the patch size**, it is harder for us to apply onto longer traces, especially in practical traces.

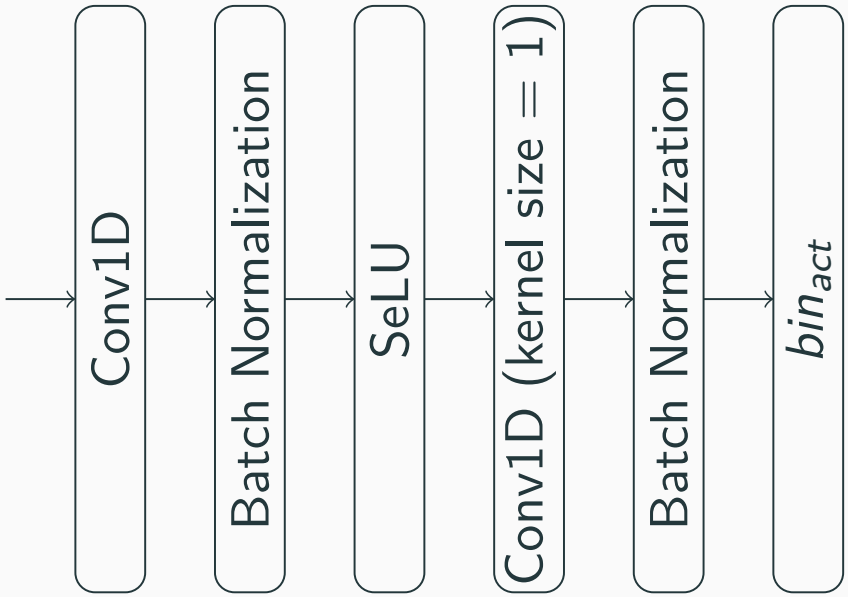
LTT block

In order to increase the learning capacity of the TT-DCNN, we use an **extra** convolutional layer of **kernel size** 1 as proposed in Benamira et al. [1].

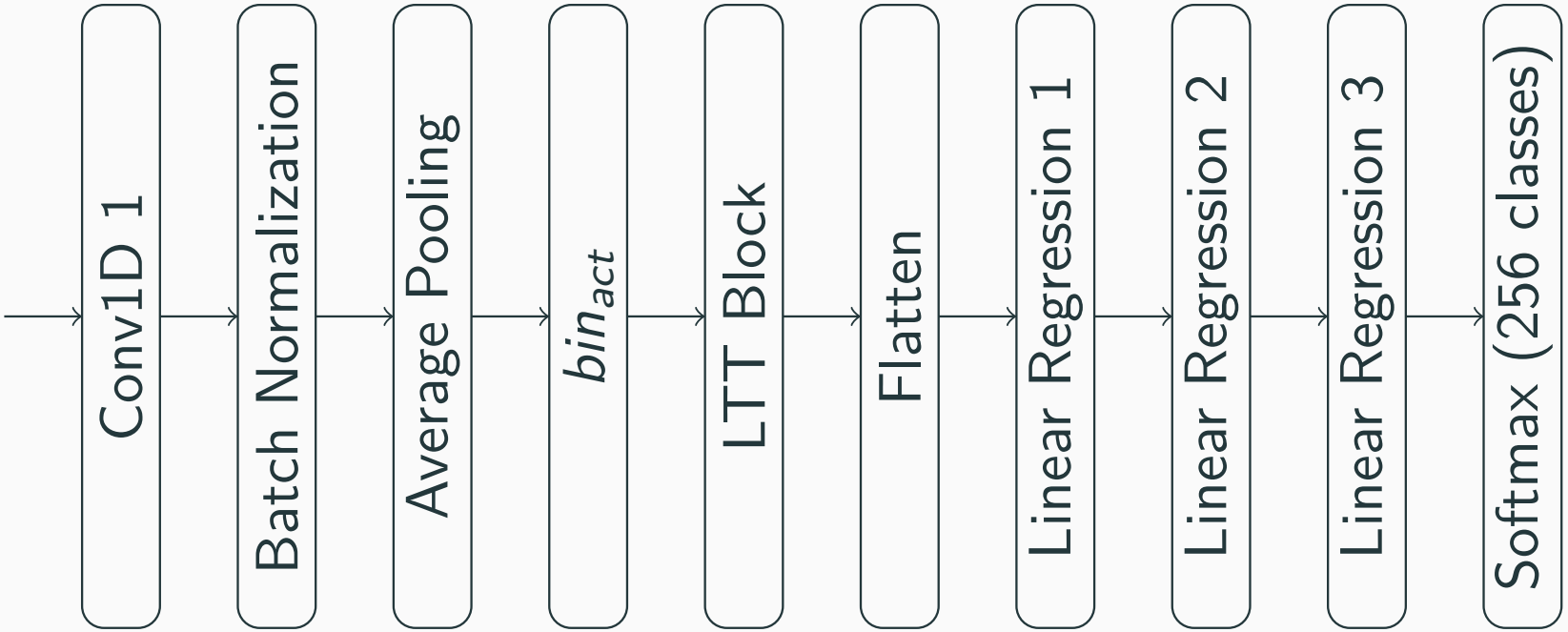
This is known as Learning Truth Table (LTT) Block.



First 1D-CNN layer of kernel size 2, stride 1 and group $g = 2$. The second 1D-CNN layer have an amplification parameter, $t = 3$ of kernel size 1 and stride 1.



$TTSCA_{big}$ architecture



TT-DCNN architecture $TTSCA_{big}$

- We consider the first convolution layer and average pool as a **preprocessing block on the traces**.
- The preprocessing block allows us to have a **patch size of** ≤ 12.
- The LTT Block Layer is then converted into SAT Equations with each literals corresponds to a window of PoIs.

Results for ASCADv1_f

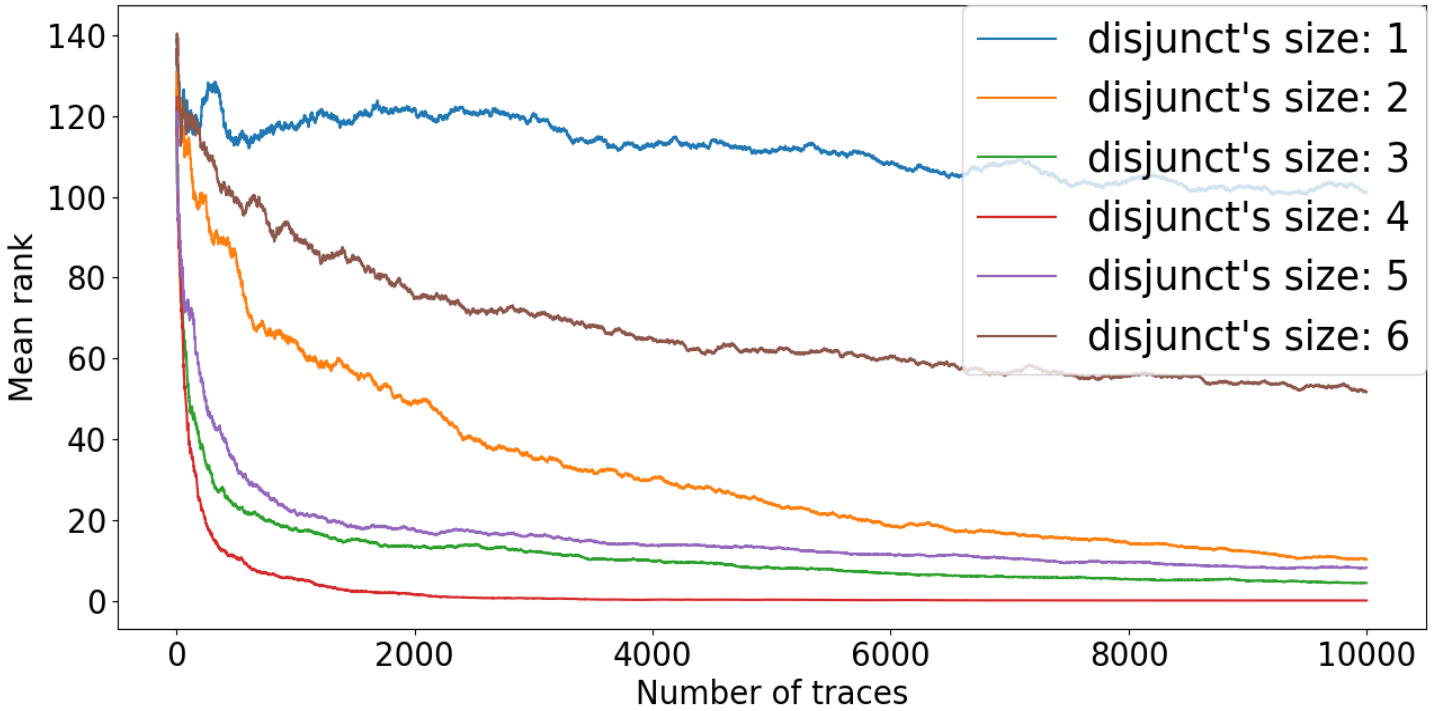
The preprocessing block provided the following windows of PoIs for each literal.

Sample Points	0 to 99	100 to 199	200 to 299	300 to 399	400 to 499	500 to 599	600 to 699
Literal	x_0	x_1	x_2	x_3	x_4	x_5	x_6

Table 1: Sample points for each literals of $TTSCA_{big}$ on ASCADv1_f.

Results for ASCADv1_f

The Guess Entropy for ASCADv1 for the different sizes is given as follows:



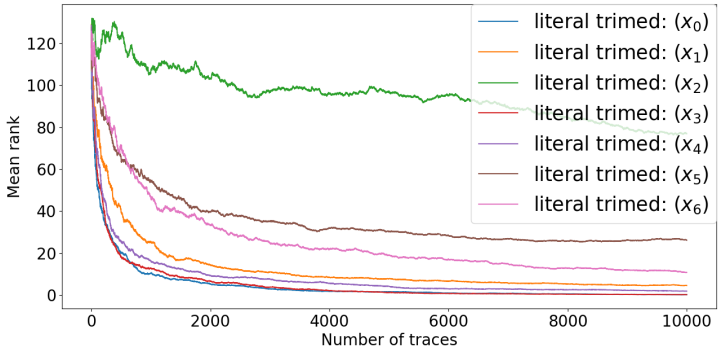
Conclusion: The disjuncts of size 4 is the smallest size with mean rank going to 0 (see red line).

Results for ASCADv1_f

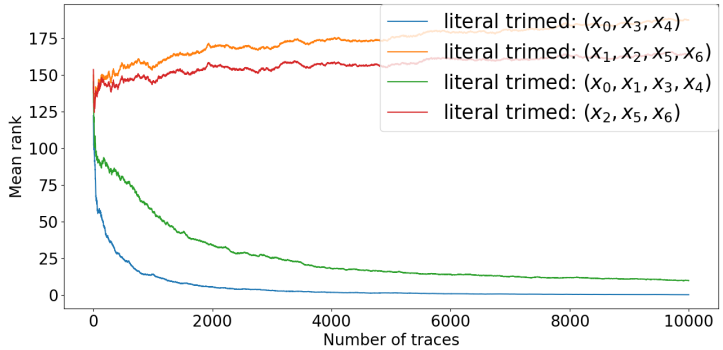
We obtained the Critical CoLs (aka CoLs relevant for key recovery) using Algorithm 1 in our paper:

List Of Critical CoLs	
(x_0, x_3, x_4, x_5)	(x_0, x_1, x_2, x_3)
(x_2, x_3, x_5, x_6)	(x_0, x_1, x_2, x_5)
(x_0, x_3, x_5, x_6)	(x_1, x_2, x_4, x_6)
(x_0, x_3, x_4, x_6)	(x_0, x_4, x_5, x_6)
(x_0, x_1, x_2, x_4)	(x_3, x_4, x_5, x_6)
(x_1, x_2, x_3, x_4)	(x_1, x_2, x_4, x_5)

Results for ASCADv1_f



(a) Trimming with every individual literals.



(b) Trimming to verify the relevancy of x_1 and x_4.

Conclusion: literals x_1, x_2, x_5 and x_6 are important.

Results for ASCADv1_f

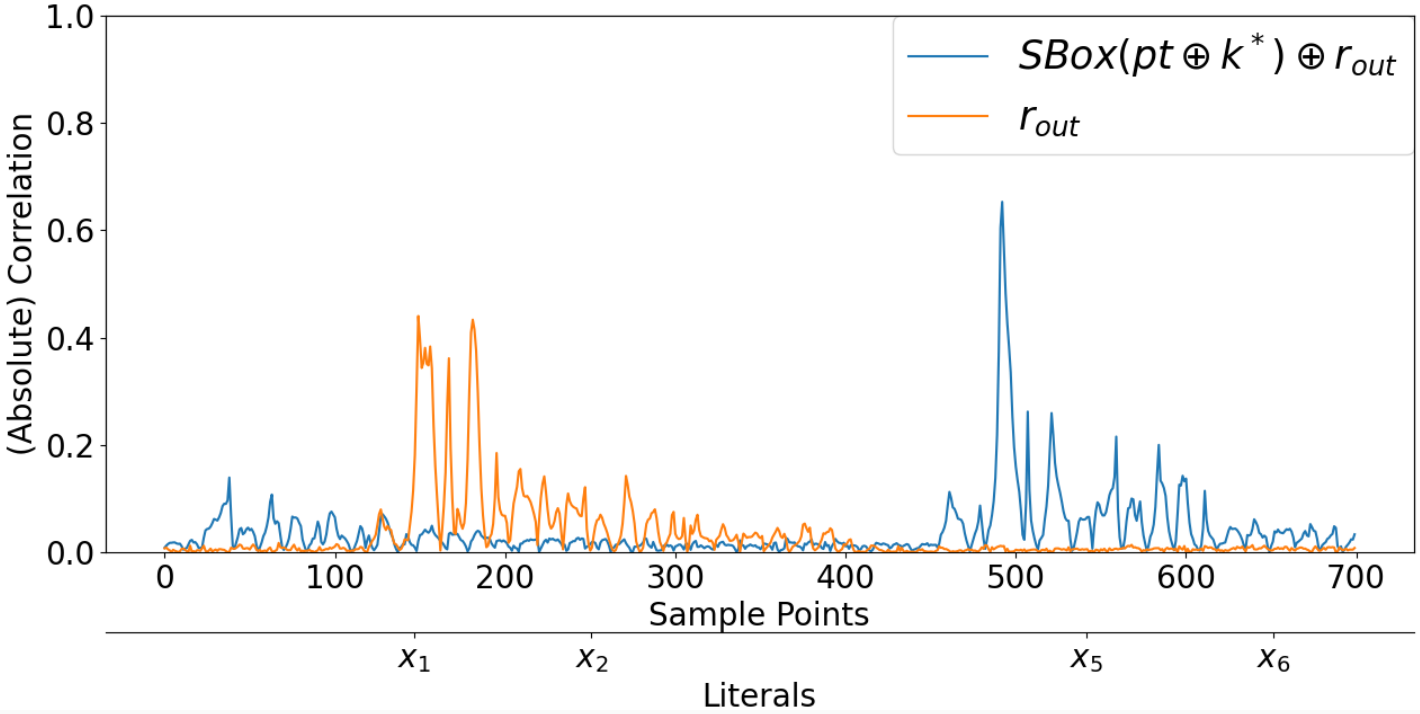
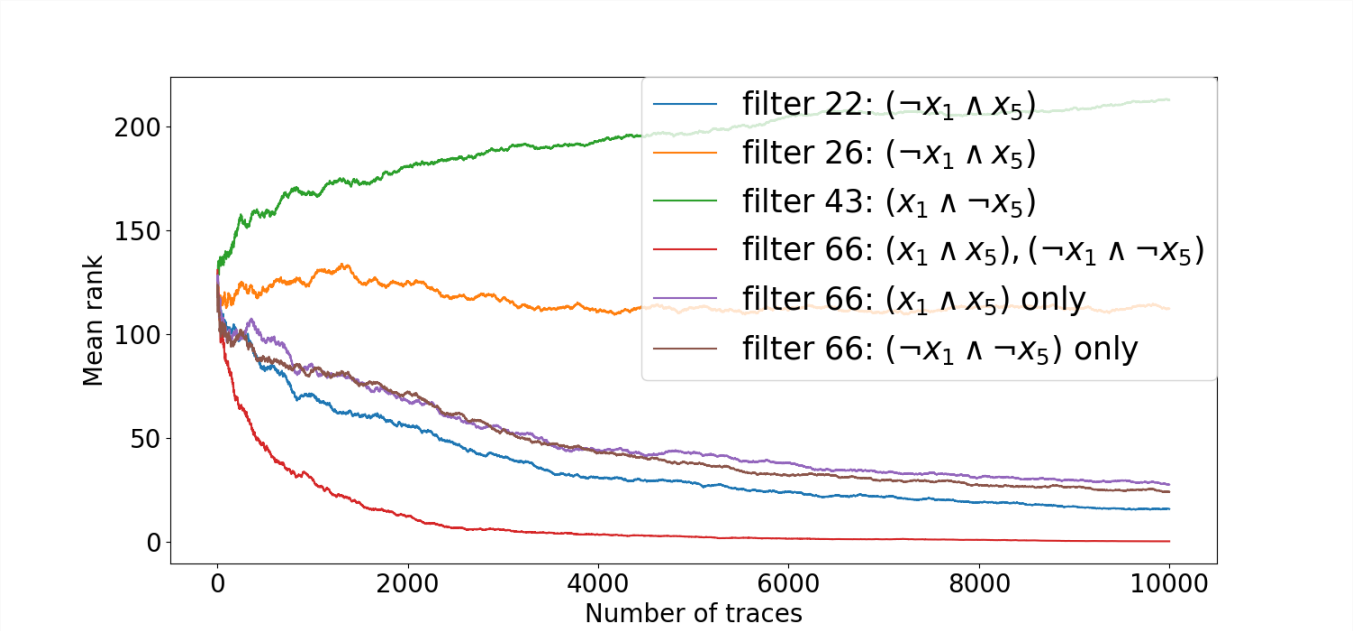


Figure 4: CPA on the attack traces of ASCADv1_f.

The $TTSCA_{big}$ is able to use the PoIs to retrieve the key.

Interesting Result

Each training is non-deterministic, this means each neural network will have slight different interpretation.

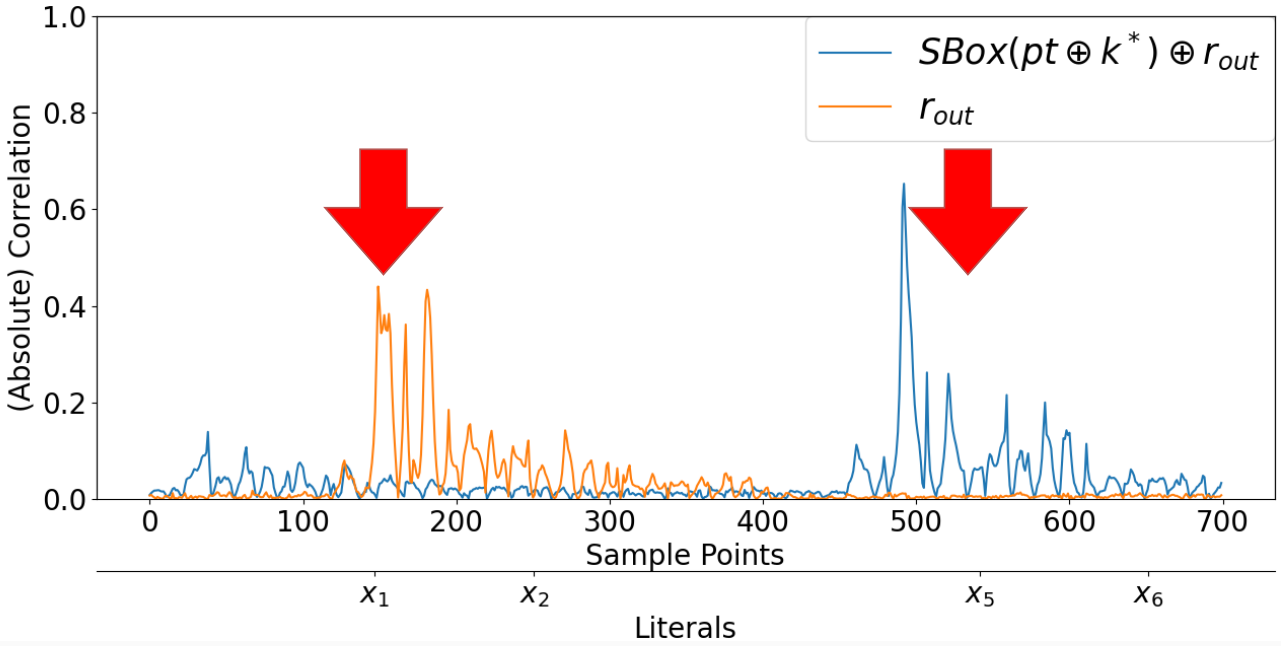


In the best case, we are able to recover the key with only filter 66:

$$x_1 \oplus \neg x_5 = (x_1 \wedge x_5) \vee (\neg x_1 \wedge \neg x_5).$$

This is exactly the Boolean masking.

Interesting Result



$$x_1 \oplus \neg x_5 = (x_1 \wedge x_5) \vee (\neg x_1 \wedge \neg x_5).$$

Overall: Retrieve the key with just *one preprocessing block*, *one XOR gate* (i.e., the masking function), and *one linear regression*.

Datasets used

Here, we have presented the results ASCADv1_f.

We also validated our approach on other 2 public datasets:

- ASCADv1_r,
- AES_HD_ext.

Table of contents

Conclusion

- Proposed two type of architecture $TTSCA_{small}$ and $TTSCA_{big}$ for SCA.
- Proposed heuristics to analyse the SAT equations for SCA.
- Detect bad implementation.
- Able to pinpoint Points of Interest and combine the leakages found. Best case: found the masking function.

Thank You!