# Deep Learning to Evaluate Secure RSA Implementations

Mathieu Carbone, Vincent Conin, Marie-Angela Cornélie, François Dassance, Guillaume Dufresne, Cécile Dumas, Emmanuel Prouff and Alexandre Venelli

CEA LETI, France
Thales ITSEF, France
SERMA Safety and Security, France
ANSSI, France

**CHES 2019**

# Context

**ANSSI** asked french **ITSEFs** to evaluate several secure **RSA** implementations against various attacks based on **Machine Learning**

- software developed by CryptoExperts
- hardware implements Montgomery Arithmetic
- evaluations should include horizontal attacks and machine learning techniques
- **only the Deep Learning aspects are discussed here**

# RSA in Secure Elements

$$m^d \bmod N$$

Exponentiation done at *software* (CPU) level
Modular Operations done at *hardware* level (Montgomery Accelerator)

Main Physical Attacks:

- Simple Power Analysis (SPA – Kocher96) -> Execution Flow independent of the private exponent (e.g. [AFT+08,CMCJ04, Joy09a,Mon87])
- Chosen Message Attacks ([Yen01,FV03])-> Message blinding

$$m^d \bmod N \rightarrow (m + rN)^d \bmod r'N$$

- DPA-like attacks (DPA – MDS99) and Statistical attacks (AFV07) -> Exponent blinding

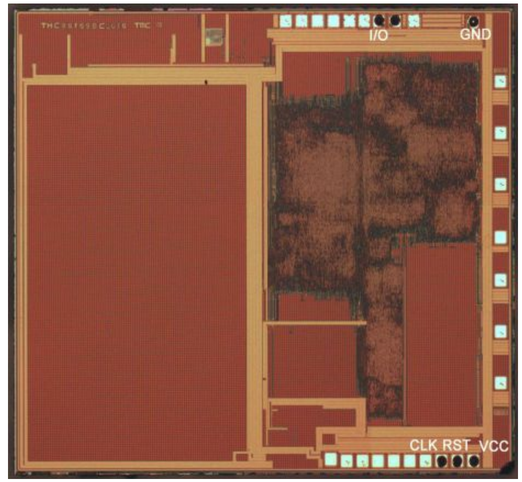
$$(m + rN)^d \bmod r'N \rightarrow (m + r_0 N)^{d + r_2 \varphi(N)} \bmod r_1 N$$

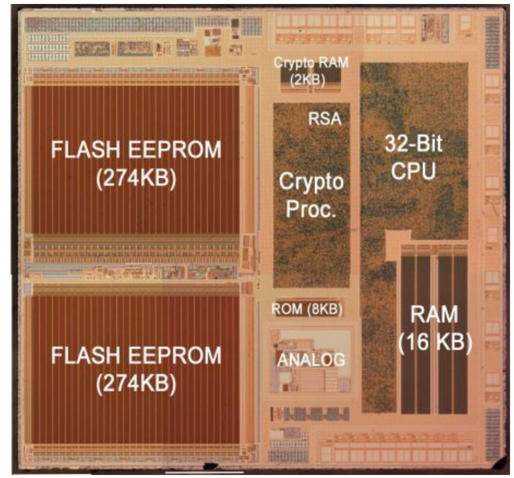
Other attacks (often assumed to be difficult to apply in practice)

- Adress-bit Attacks ([IIT02]), Horizontal Collision Attacks (Wal01,CFGRV10)

# Hardware Specifications

| | |
|---|---|
| **Product name** | BOUMBO |
| **Product versions** | Version 1<br>Version 2<br>Version 3 |
| **Technology** | 32-bit ARM core SC 100 |
| **RAM size** | 18 KB |
| **ROM size** | 8 KB |
| **FLASH size** | 548 KB |
| **Co-processing units** | DES/TDES, RSA, CRC, TRNG |
| **Cryptographic Library (list of provided algorithms)** | RSA SFM developed by CryptoExperts |
| **Form factor(s)** | Smart Cards |
| **Communication protocols** | ISO 7816 T=0/T=1 protocol |



# Software Specifications

**RSA_SFM** (u32* *output*, u32* *input*, u32* *modulus*, u32* *exponent*, u32* *euler_totient*, int *len* )

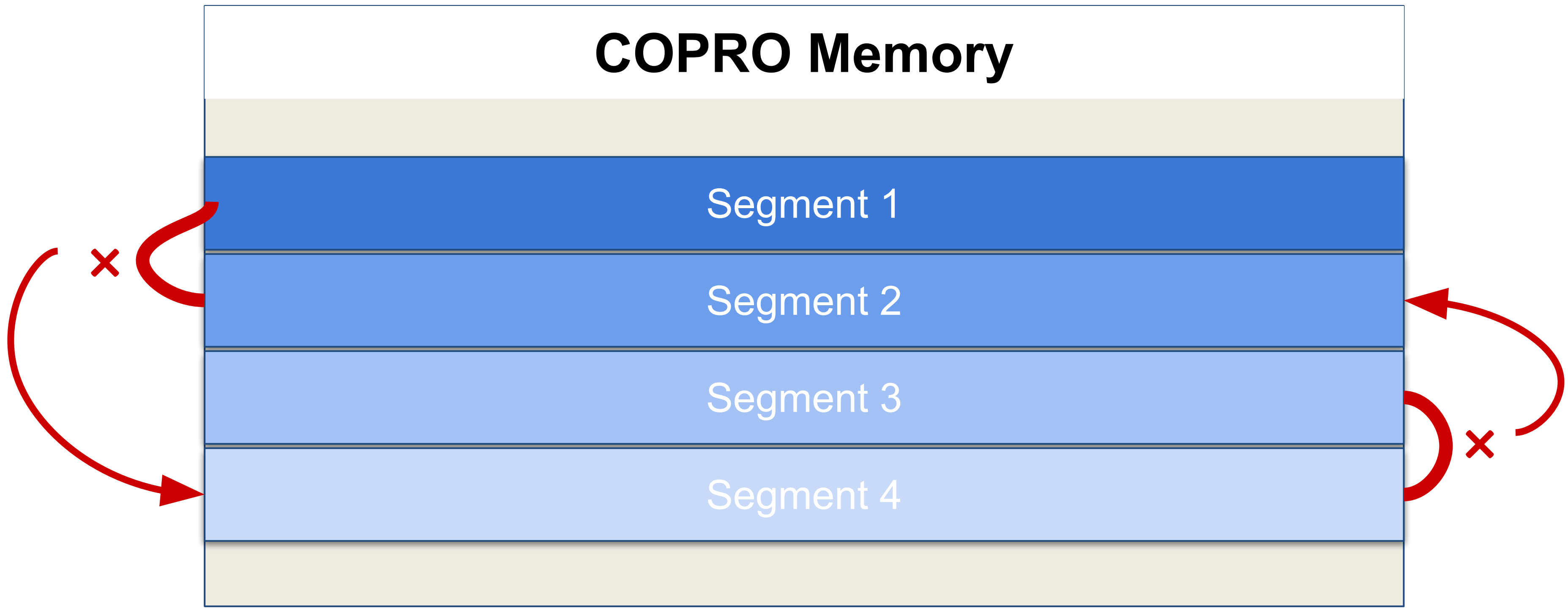
- *output* is the memory address where the output is written on *len* words,
- *input* is the memory address where the input is stored on *len* words,
- *modulus* is the memory address where the modulus is stored on *len* words,
- *exponent* is the memory address where the modulus is stored on *len* words,
- *Euler totient* is the memory address where the Euler totient of the modulus is stored on *len* words,
- *len* is the word-length of the RSA modulus.

**Summing up the three randomization techniques, the implementation processes**:

$$((m + \mathbf{r_1} * N)^{d + \mathbf{r_2} * \varphi(N)} \ mod \ \mathbf{r_0} * N) \ mod \ N$$

for three independent **random** integers $r_0, r_1$ and $r_2$ of length 64 bits.

# Memory Organization



# SQUARE & MULTIPLY ALWAYS

```
seg_1  = 1;                                   // input
seg_2 = 2;                                    // accumulator
seg_3 = 3;                                    // dummy register

//--- Exponentiation loop ---//
// MMM = Montgomery Modular Multiplier
FOR i = len-1 TO i = 0

    exp_bit = exponent [i]

    MMM (seg_4, seg_2, seg_2)                 //--- Square accumulator ---//
    seg_2  = seg_4

    seg_4 = 9 - seg_2 - seg_3                 //--- Multiply accumulator and Input ---//
    MMM (seg_4, seg_2, seg_1))

    seg_2    = exp_bit * seg_4   +  (1-exp_bit) * seg_2     //--- Assign Result wrt current exp bit ---//
    seg_3    = exp_bit*seg_3    +  (1-exp_bit) * seg_4

ENDFOR
```

# Operations Sequence

| bit | | 1 | | 0 | | 1 | | 1 | | 0 | | 1 | | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| op | | Square | mult | Square | mult | Square | mult | Square | mult | Square | mult | Square | mult | Square |
| op A | seg | 2 | 4 | 2 | 4 | 4 | 3 | 4 | 3 | 4 | 3 | 3 | 2 | 3 |
| | val | 1 | 1 | m | $m^2$ | $m^2$ | $m^4$ | $m^5$ | $m^{10}$ | $m^{11}$ | $m^{22}$ | $m^{22}$ | $m^{44}$ | $m^{45}$ |
| op B | seg | 2 | 1 | 2 | 1 | 4 | 1 | 4 | 1 | 4 | 1 | 3 | 1 | 3 |
| | val | 1 | m | m | m | $m^2$ | m | $m^5$ | m | $m^{11}$ | m | $m^{22}$ | m | $m^{45}$ |
| res | seg | 4 | 2 | 4 | 2 | 3 | 4 | 3 | 4 | 3 | 4 | 2 | 3 | 2 |
| | val | 1 | m | $m^2$ | $m^3$ | $m^4$ | $m^5$ | $m^{10}$ | $m^{11}$ | $m^{22}$ | $m^{23}$ | $m^{44}$ | $m^{45}$ | $m^{90}$ |

# Operations Sequence

| bit | | 1 | | 0 | | 1 | | 1 | | 0 | | 1 | | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| op | | Square | mult | Square | mult | Square | mult | Square | mult | Square | mult | Square | mult | Square |
| op A | seg | 2 | 4 | 2 | 4 | 4 | 3 | 4 | 3 | 4 | 3 | 3 | 2 | 3 |
| | val | 1 | 1 | m | $m^2$ | $m^2$ | $m^4$ | $m^5$ | $m^{10}$ | $m^{11}$ | $m^{22}$ | $m^{22}$ | $m^{44}$ | $m^{45}$ |
| op B | seg | 2 | 1 | 2 | 1 | 4 | 1 | 4 | 1 | 4 | 1 | 3 | 1 | 3 |
| | val | 1 | m | m | m | $m^2$ | m | $m^5$ | m | $m^{11}$ | m | $m^{22}$ | m | $m^{45}$ |
| res | seg | 4 | 2 | 4 | 2 | 3 | 4 | 3 | 4 | 3 | 4 | 2 | 3 | 2 |
| | val | 1 | m | $m^2$ | $m^3$ | $m^4$ | $m^5$ | $m^{10}$ | $m^{11}$ | $m^{22}$ | $m^{23}$ | $m^{44}$ | $m^{45}$ | $m^{90}$ |

$$seg \textbf{ for } Square_i = seg \textbf{ for } Square_{i+1} \iff exponent_i = 1$$

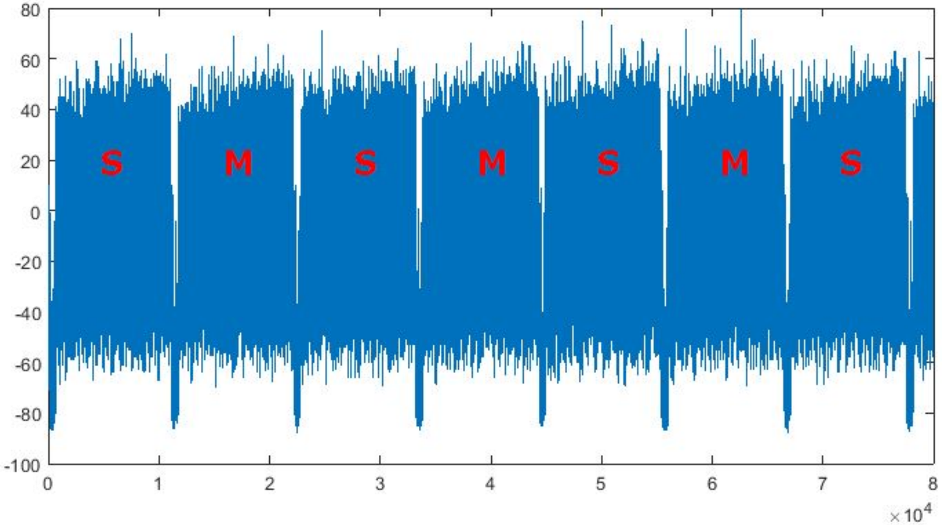
# Operands Sequence

| bit | | 1 | | 0 | | 1 | | 1 | | 0 | | 1 | | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| op | | Square | mult | Square | mult | Square | mult | Square | mult | Square | mult | Square | mult | Square |
| op A | seg | 2 | 4 | 2 | 4 | 4 | 3 | 4 | 3 | 4 | 3 | 3 | 2 | 3 |
| | val | 1 | 1 | m | $m^2$ | $m^2$ | $m^4$ | $m^5$ | $m^{10}$ | $m^{11}$ | $m^{22}$ | $m^{22}$ | $m^{44}$ | $m^{45}$ |
| op B | seg | 2 | 1 | 2 | 1 | 4 | 1 | 4 | 1 | 4 | 1 | 3 | 1 | 3 |
| | val | 1 | m | m | m | $m^2$ | m | $m^5$ | m | $m^{11}$ | m | $m^{22}$ | m | $m^{45}$ |
| res | seg | 4 | 2 | 4 | 2 | 3 | 4 | 3 | 4 | 3 | 4 | 2 | 3 | 2 |
| | val | 1 | m | $m^2$ | $m^3$ | $m^4$ | $m^5$ | $m^{10}$ | $m^{11}$ | $m^{22}$ | $m^{23}$ | $m^{44}$ | $m^{45}$ | $m^{90}$ |

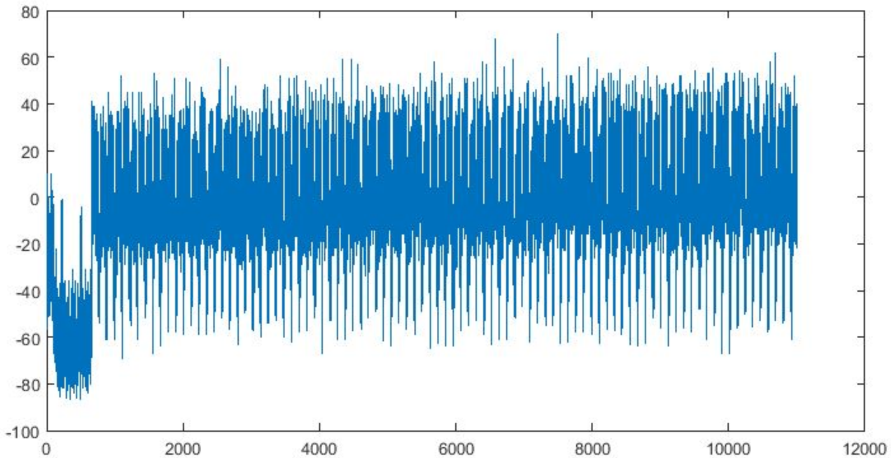
$$Op\ A \textbf{ for } Square_i = Op\ A \textbf{ for } Mult_{i+1} \iff exponent_i = 0$$

# Power Consumption Measurements

Exponent of size $n = 1088$ = 1024 + 64.
Measured at 50 MS/s using a Lecroy WaveRunner 625Zi oscilloscope.
25, 000, 000 time samples per trace



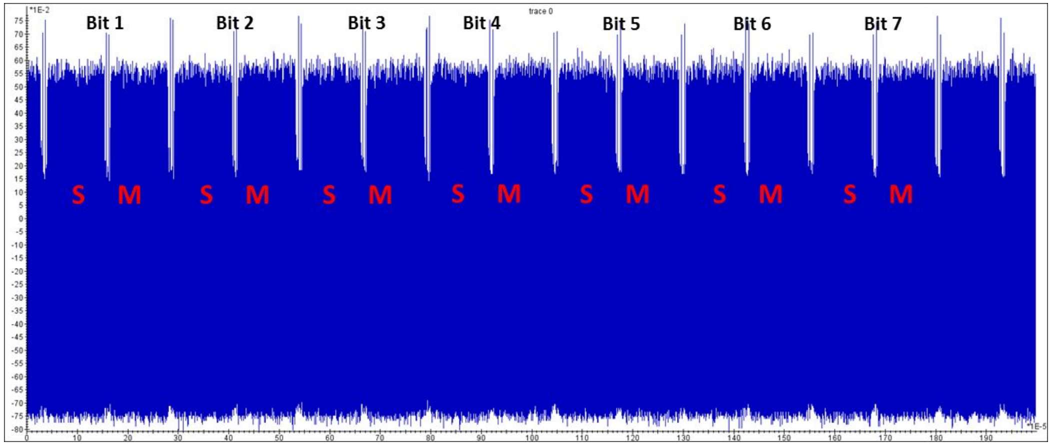

*Succession of Square and Mult with MMM*



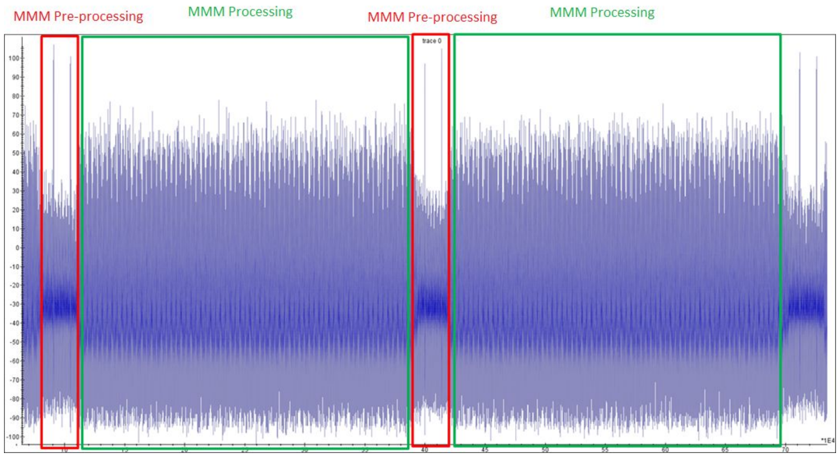

*Single MMM*

# Electromagnetic Measurements (EM)

Signal acquired at 2.5 $GS/s$ sampling rate over 200 $\mu s$
Each trace is composed of 5, 000, 000 time samples which correspond to the 7 MSB of the masked exponent
Lecroy WaveRunner 625Zi oscilloscope and Langer ICR EM probe



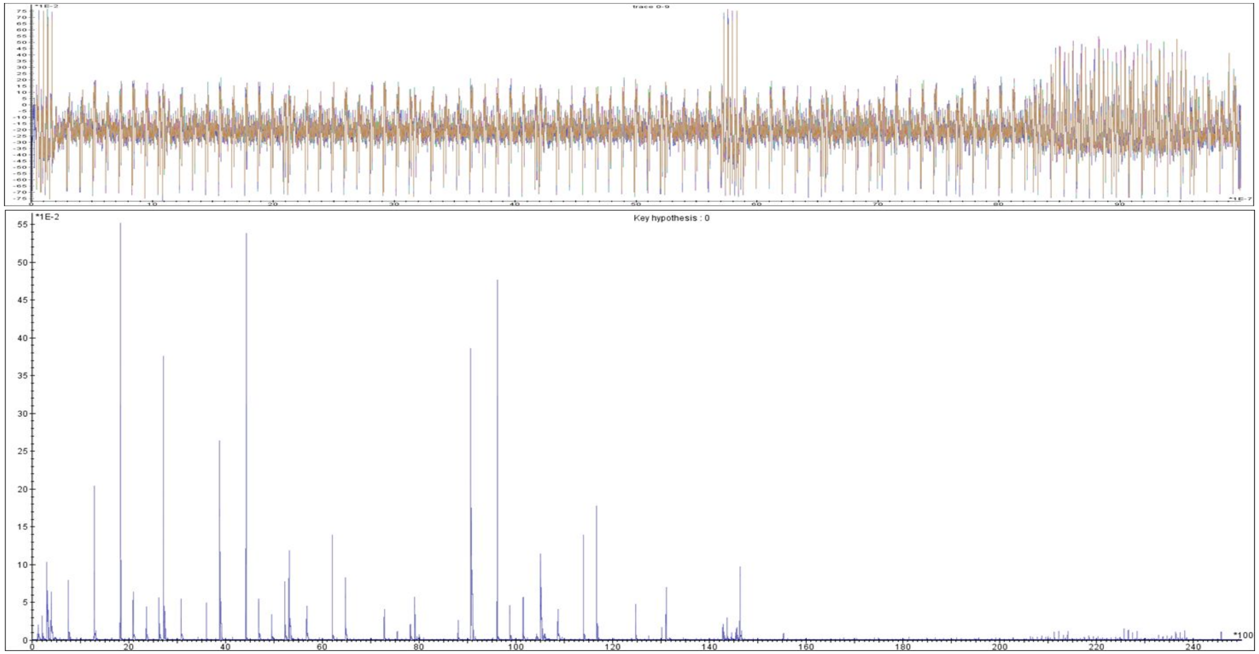

*Succession of Squares and Mults*




*Square followed by Mult*

# Leakage Assessment Phase (EM)

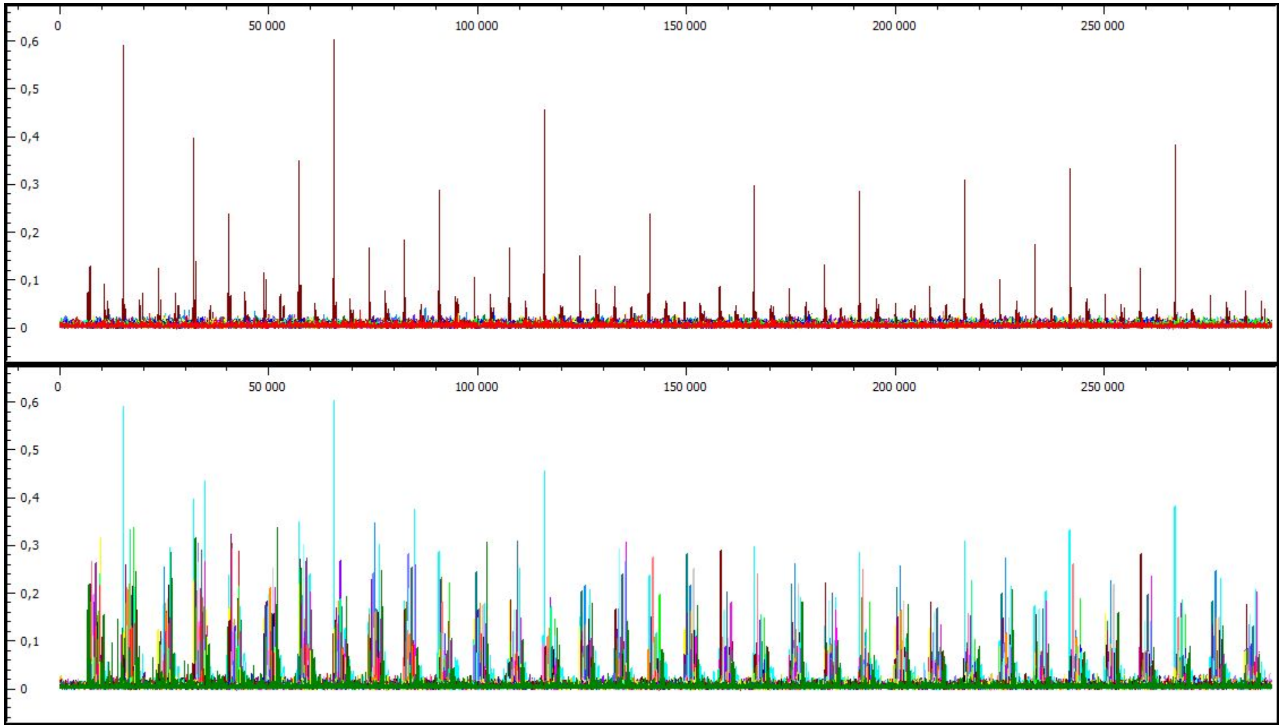
**Goal:** detect time samples that statistically depend on register index




**EM** Campaign - **SNR** for **seg_4** *versus* the squaring initialization (bottom) and the original EM trace (top)

# Leakage Assessment Phase (EM)

**Goal:** detect time samples that statistically depend on **operand** bits

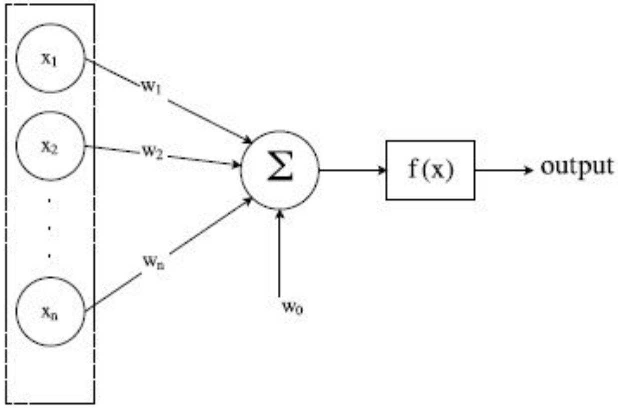



*Monobit SNRs (on 50, 000 traces) for the first operand of the MMM.*

# Deep Neural Networks (Perceptron)

**Goal:** from observations associated to labels, build an algorithm/model which correctly associates a label to a new observation

**Fundamental Example**: the Perceptron




**Assume** we have obs. $\vec{x}^j = (x_1, x_2, \dots, x_n)$ associated to labels $y^j$

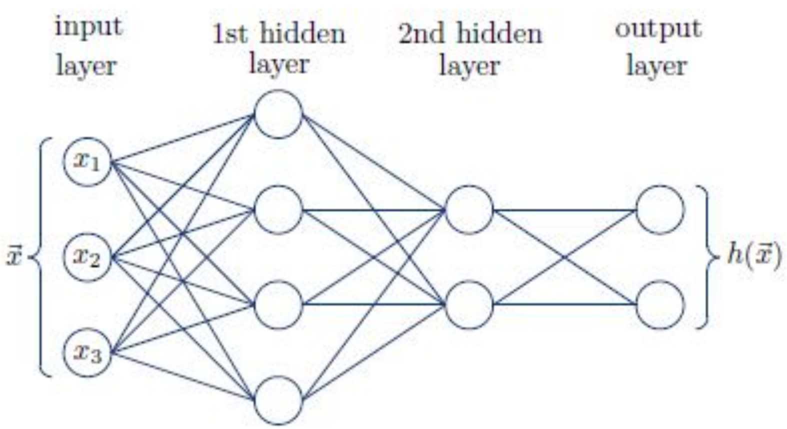
**Learning Phase:** find weights $\vec{w}^j = (w_1, w_2, \dots, w_n)$ such that for every j:

$$\Pr\left[\text{label}(\vec{x}^j) = \mathrm{y}^j \mid \text{Perceptron output} = \mathrm{f}(\vec{x}^j \cdot w^j)\right] \approx 1$$

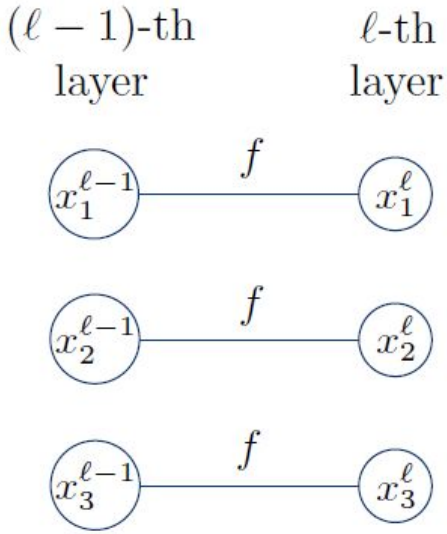
# Deep Neural Networks (MLP)

**Goal:** extend to non-linear classification problems

**Combine** several **perceptrons** in **layers**

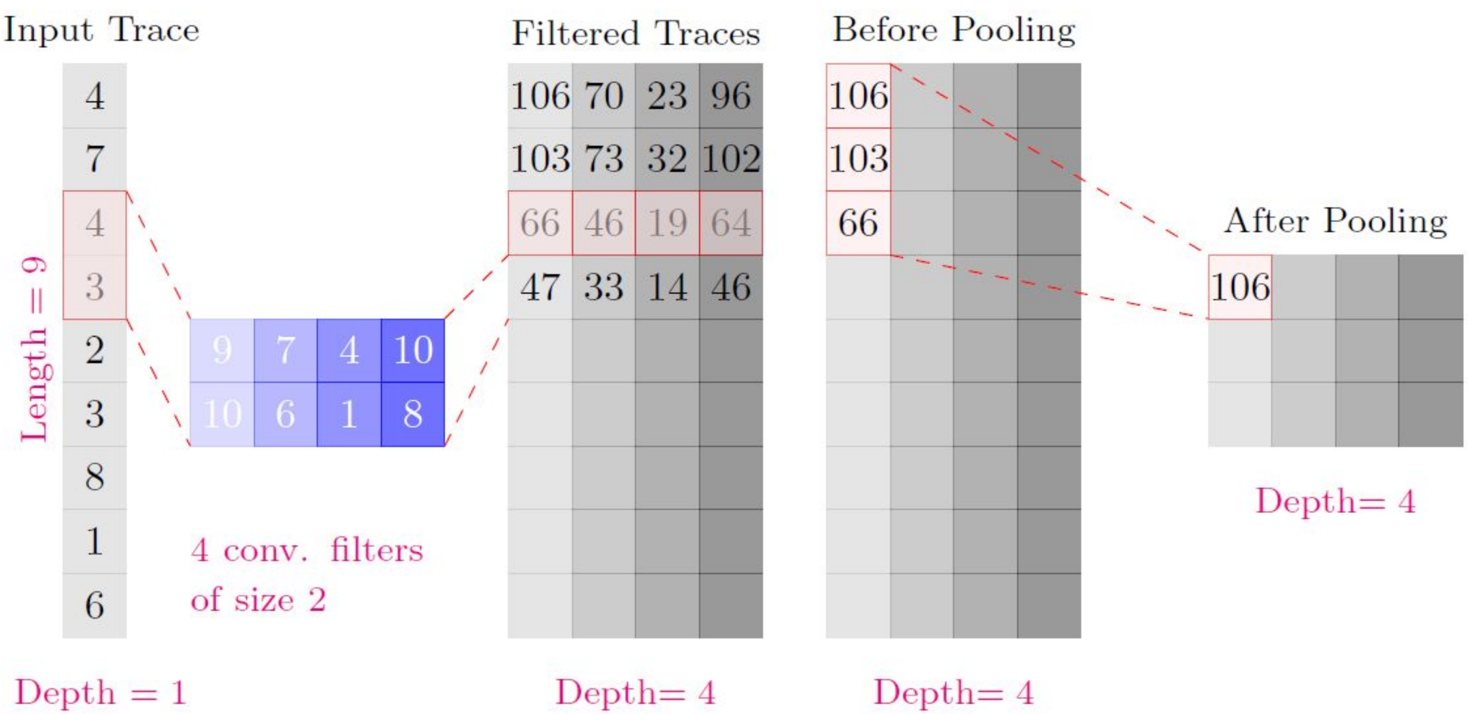



**Use** the same non-linear **activation function** to add non-linearity btw consecutive layers



# Deep Neural Networks (CNN)

**Goal:** extend to non-linear classification, **while being robust to some geometrical changes**



# Deep Neural Networks vs RSA

An input will be a leakage during a square (or a mult) MMM operation

The associated label will be:

- the value of **seg_4** index
- or a tuple composed of some bits of the **Operand A**

**Goal**: train an algorithm to correctly associate a new MMM trace to the corresponding **seg_4** (or **Operand A**) label

# Register Index Recovery

## Template Attack (EM Case)

| Set | Card | Number of traces | N | M | d | k0 (64-bit) | k1 (64-bit) | k2 (64-bit) |
|---|---|---|---|---|---|---|---|---|
| C0 | #2 | 2,016 | known varying one-shot | known varying one-shot | known varying one-shot | known varying one-shot random | known varying one-shot random | known varying one-shot random |
| C1 | #3 | 30 | known varying one-shot | known varying one-shot | known varying one-shot | known varying one-shot random | known varying one-shot random | known varying one-shot random |
| C2 | #1 | 1 | known varying one-shot | known varying one-shot | known varying one-shot | known varying one-shot random | known varying one-shot random | known varying one-shot random |

| | |
|---|---|
| **Used sets** | C0 for profiling phase, C2 for exploitation phase. |
| **Attack target** | @sfree. |
| **Leakage model** | Identity (three class labels). |
| **Normalization** | Scaling, i.e. transforming linearly data in range [0;1] for C0 and C1 independently. |
| **Compression method** | Sample selection, i.e. POI such that time samples exceeding a threshold greater than chosen value (see 3.2.5). |

| | |
|---|---|
| **Candidate selection** | Maximum. |
| **Parameters tuning** | None. |
| **Attack metric** | Success rate over 1,599 exponent bits extracted from C2 (last bit is not taken into account). |

| Covariance matrix form | Success rate |
|---|---|
| Identity matrix | 81,6% |
| Individual | 83,4% |
| Common | 83,4% |

# Register Index Recovery

## MLP (EM Case)

| Set | Card | Number of traces | N | M | d | k0 (64-bit) | k1 (64-bit) | k2 (64-bit) |
|---|---|---|---|---|---|---|---|---|
| C0 | #2 | 2,016 | known varying one-shot | known varying one-shot | known varying one-shot | known varying one-shot random | known varying one-shot random | known varying one-shot random |
| C1 | #3 | 30 | known varying one-shot | known varying one-shot | known varying one-shot | known varying one-shot random | known varying one-shot random | known varying one-shot random |
| C2 | #1 | 1 | known varying one-shot | known varying one-shot | known varying one-shot | known varying one-shot random | known varying one-shot random | known varying one-shot random |

| | |
|---|---|
| **Used sets** | C0 for training in learning phase,<br>C1 as an evaluation set in learning phase,<br>C2 for prediction / exploitation phase. |
| **Attack target** | `@sfree`. |
| **Leakage model** | Identity (three class labels). |
| **Normalization** | Scaling, i.e. transforming linearly data in range [0;1] for C0, C1 and C2 independently. |
| **Compression method** | Sample selection, i.e. POI such that time samples exceeding a threshold greater than chosen value (see 3.2.5). |
| **Candidate selection** | Maximum. |
| **Neural network** | MLP, see below. |
| **Parameters tuning** | Parameter discovering and optimization via `hyperas/hyperopt` libraries.<br>• Batch_size = 128,<br>• Learning rate = 0.007 |
| **Attack metric** | Success rate over 1,599 exponent bits extracted from C2 (last bit is not taken into account). |

The MLP is trained with the patterns from C0. The patterns from C1 are used as an evaluation set. The training accuracy is 98.72%, the accuracy for the evaluation set is 98.57%.

Finally the MLP is applied to the patterns from C2. The success rate of the attack is is 98.38%.

# Register Index Recovery

## CNN (EM Case)

| Set | Card | Number of traces | N | M | d | k0 (64-bit) | k1 (64-bit) | k2 (64-bit) |
|---|---|---|---|---|---|---|---|---|
| C0 | #2 | 2,016 | known varying one-shot | known varying one-shot | known varying one-shot | known varying one-shot random | known varying one-shot random | known varying one-shot random |
| C1 | #3 | 30 | known varying one-shot | known varying one-shot | known varying one-shot | known varying one-shot random | known varying one-shot random | known varying one-shot random |
| C2 | #1 | 1 | known varying one-shot | known varying one-shot | known varying one-shot | known varying one-shot random | known varying one-shot random | known varying one-shot random |

| | |
|---|---|
| **Used sets** | C0 for training in learning phase,<br>C1 as an evaluation set in learning phase,<br>C2 for prediction / exploitation phase. |
| **Attack target** | `@sfree`. |
| **Leakage model** | Three class labels. |
| **Normalization** | Scaling, i.e. transforming linearly data in range [0;1] for C0 and C2 independently. |
| **Compression method** | Sample selection, i.e. POI such that time samples exceeding a threshold greater than chosen value (see 3.2.5). |
| **Candidate selection** | Maximum. |
| **Neural network** | CNN, see below. |
| **Parameters tuning** | Parameter discovering and optimization via `hyperas/hyperopt` libraries.<br>• Batch_size = 512<br>• Learning rate = 0.005 |
| **Attack metric** | Success rate over 1,599 exponent bits extracted from C2 (last bit is not taken into account). |

Finally the CNN is applied to the patterns from C2. The success rate of the attack is 99.31%.

# Register Index Recovery

## Power Consumption Case

| Attack type | Best score |
|---|---|
| Template attack | 83.4% |
| Random Forest | 93.1% |
| K-Nearest Neighbors | 98.7% |
| Extreme Gradient Boosting | 98.7% |
| Support Vector Machine | 97.1% |
| Multi-Layer Perceptron | 98.38% |
| Convolutional Neural Network | 99.31% |

In the best case 99.31% of the randomized exponent are retrieved.

The wrong guesses can be corrected thanks to [SW14]. Simulations performed by the evaluator show that it is necessary to retrieve 99.31% of around 15 randomized exponents to reveal the secret exponent in clear based on this approach.

[SW14]: W. Schindler et al. - Power attacks in the presence of exponent blinding (2014)

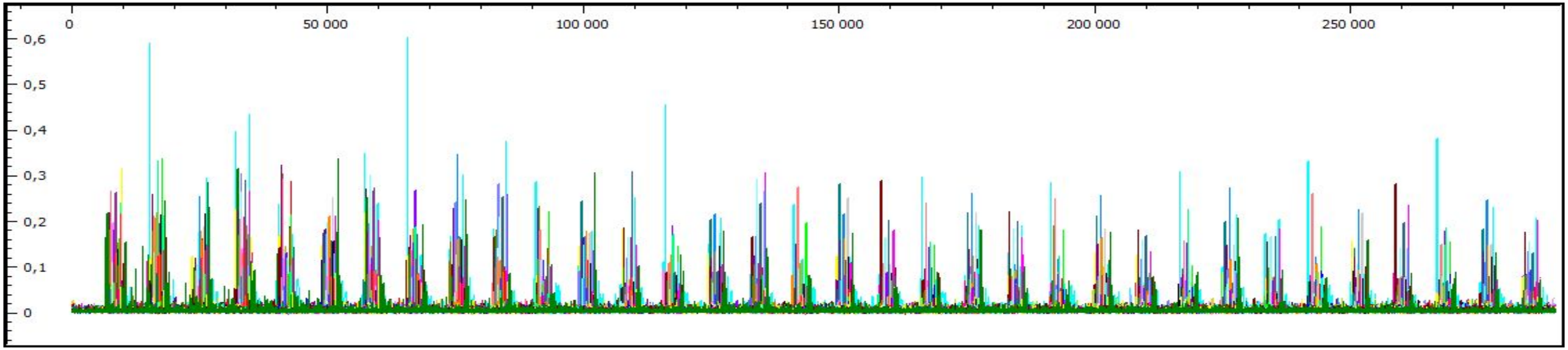
# Profiling the Operand Collisions

Targeted Sensitive Data: operand A in **mult** then **square**

**If** collision, **then** exponent bit is 0

→ recover information on the operand A values

→ decide whether they are equal or not




Initial Step: get leakages on the twelve bit of each 32-bits word of A

- Since |A| = 1088 for the tests, **34** bits are targeted by operation.

# Profiling the Operand Collisions

- 34 attacks/matchings for each operand A
- 10,000 traces for profiling and 1,400 traces for matching

Template Attacks

→ success rate for each bit: 93%

CNN

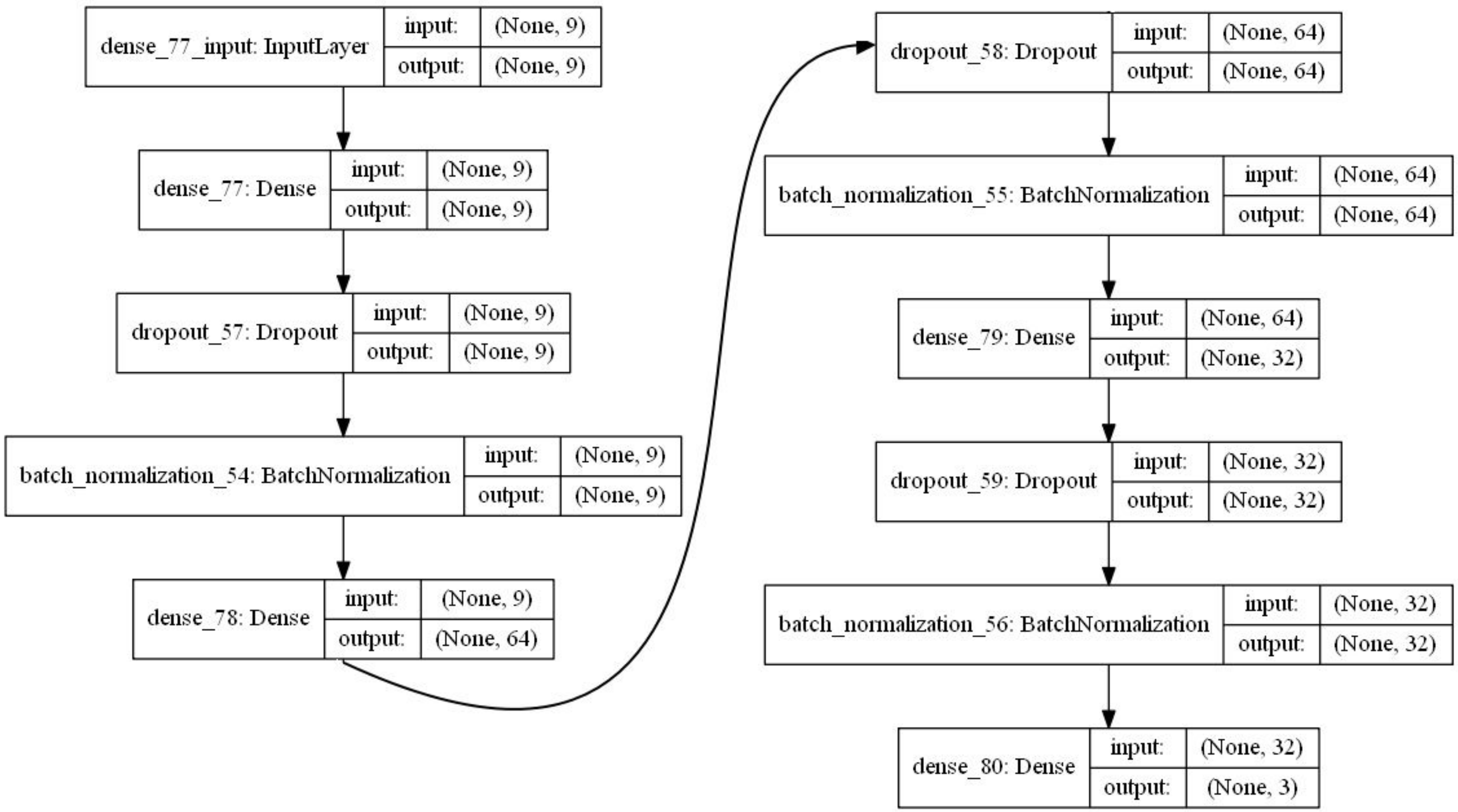
→ success rate for each bit: 97%

# Conclusions

- Deep learning may be very efficient against secure RSA implementations
- Selection of POI is less important than in TA attacks
- Deep Learning techniques currently used are very basic and attacks can be greatly improved
- Reported tests are for a Toy Implementation (RSA evaluated in CC should be much more resistant)

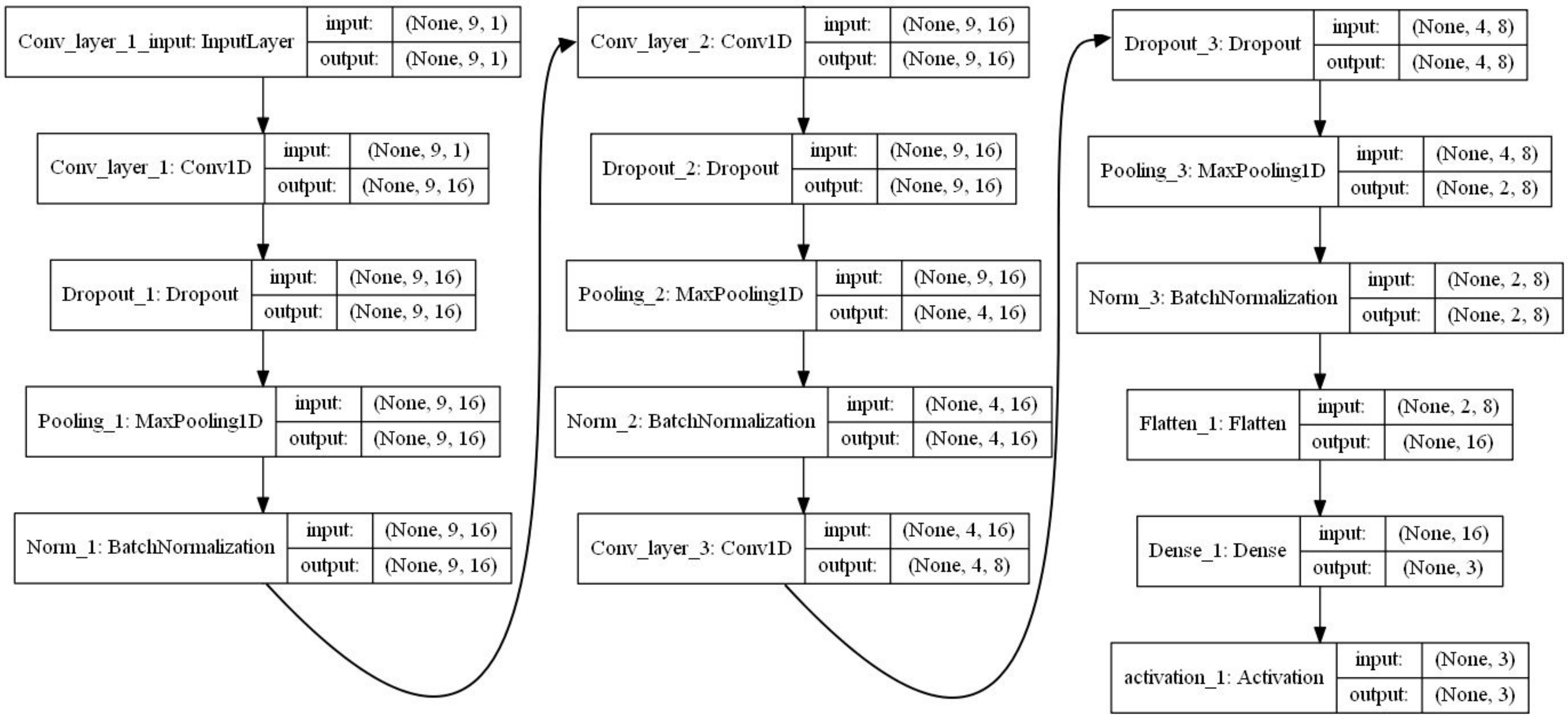
# Register Index Recovery

## Best MLP Model



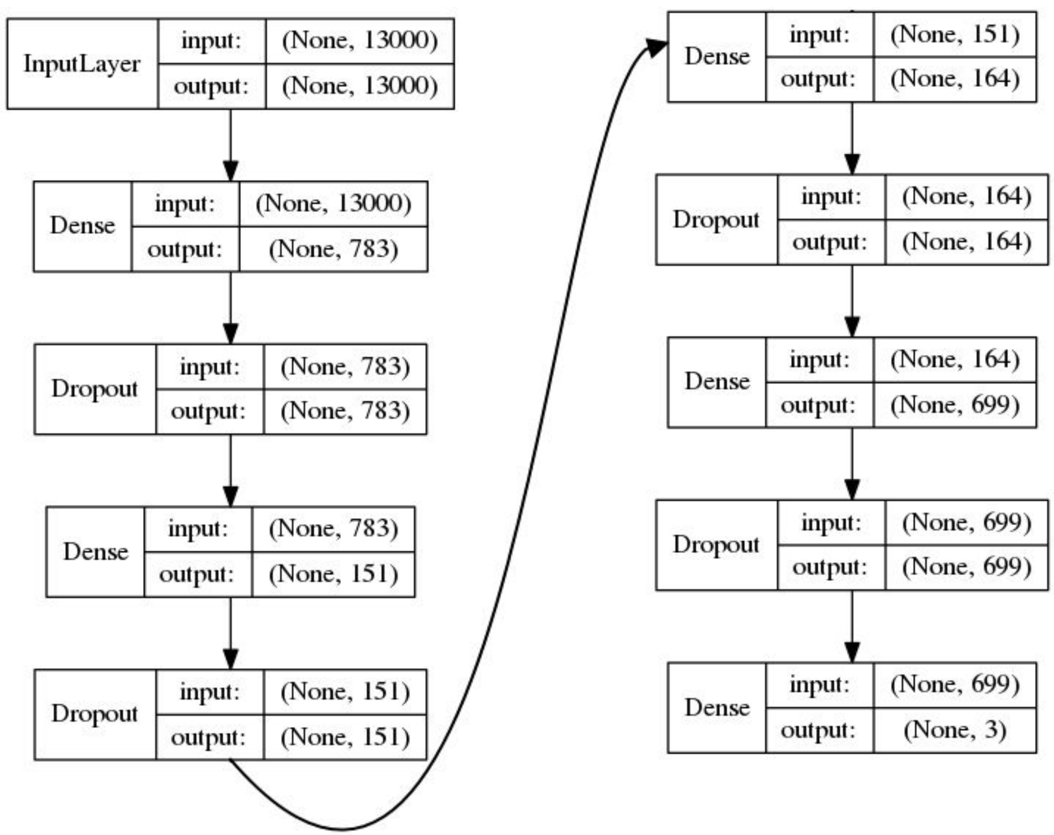
# Register Index Recovery

## Best CNN Model



# Partial Operand A Recovery

## Best MLP Model



# Partial Operand A Recovery

## Best CNN Model