The Curse of Class Imbalance and Conflicting Metrics with Machine Learning for Side-channel Evaluations

Stjepan Picek, Annelie Heuser, Alan Jovic, Shivam Bhasin, and Francesco Regazzoni
Big Picture

plaintext → device (training) → side-channel measurements → classifier (training) → profiled model → evaluation metric
plaintext → device (attacking) → side-channel measurements → classifier (attacking)
Big Picture

plaintext → device (training) → side-channel measurements → classifier (training) → profiled model → evaluation metric

plaintext → device (attacking) → side-channel measurements → classifier (attacking) → template evaluation + max likelihood

template building

success rate

guessing entropy
Big Picture

plaintext → device (training) → side-channel measurements → classifier (training) → profiled model → evaluation metric

plaintext → device (attacking) → side-channel measurements → classifier (attacking) → evaluation metric

template building
ML training

success rate
guessing entropy
accuracy

ML testing

template evaluation + max likelihood
Big Picture

1. template building
   ML training

2. template evaluation + max likelihood
   evaluation metric

ML testing

plaintext → device (training) → classifier (training)
side-channel measurements → labels → profiled model

plaintext → device (attacking) → classifier (attacking)
side-channel measurements → evaluation metric
Labels

- typically: intermediate states computed from plaintext and keys
- Hamming weight (distance) leakage model commonly used
- problem: introduces imbalanced data
- for example, occurrences of Hamming weights for all possible 8-bit values:

<table>
<thead>
<tr>
<th>HW value</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
</tr>
</thead>
<tbody>
<tr>
<td>Occurrences</td>
<td>1</td>
<td>8</td>
<td>28</td>
<td>56</td>
<td>70</td>
<td>56</td>
<td>28</td>
<td>8</td>
<td>1</td>
</tr>
</tbody>
</table>
Why do we use HW?

- often does not reflect realistic leakage model
Why do we use HW?

- often does not reflect realistic leakage model
Why do we use HW?

- reduces the complexity of learning
- works (sufficiently good) in many scenarios for attacking
Why do we care about imbalanced data?

• most machine learning techniques rely on loss functions that are “designed” to maximise accuracy

• in case of high noise: predicting only HW class 4 gives accuracy of 27%

• but is not related to secret key value and therefore does not give any information for SCA
What to do?

• in this paper: transform dataset to achieve balancedness?
• how?
  • throw away data
  • add data
  • (or choose data before ciphering)
Random under sampling

- only keep # of samples equal to the least populated class
- binomial distribution: many unused samples
Random under sampling

• only keep # of samples equal to the least populated class

• binomial distribution: many unused samples
Random oversampling with replacement

- randomly selecting samples from the original dataset until amount is equal to largest populated
- simple method, in other context comparable to other methods
- may happen that some samples are not selected at all

Class 1
7 samples

Class 2
13 samples
Random oversampling with replacement

- randomly selecting samples from the original dataset until amount is equal to largest populated
- simple method, in other context comparable to other methods
- may happen that some samples are not selected at all

Class 1

Class 2

“13” samples

13 samples
SMOTE

- synthetic minority oversampling technique
- generating synthetic minority class instances
- nearest neighbours are added (corresponding to Euclidean distance)

Class 1: 7 samples

Class 2: 13 samples
SMOTE

- synthetic minority oversampling technique
- generating synthetic minority class instances
- nearest neighbours are added (corresponding to Euclidean distance)

Class 1
13 samples

Class 2
13 samples
SMOTE+ENN

- Synthetic Minority Oversampling Technique with Edited Nearest Neighbor
- SMOTE + data cleaning
- oversampling + undersampling
- removes data samples whose class different from multiple neighbors
SMOTE+ENN

- Synthetic Minority Oversampling Technique with Edited Nearest Neighbor
- SMOTE + data cleaning
- oversampling + undersampling
- removes data samples whose class different from multiple neighbors

Class 1: 10 samples
Class 2: 10 samples
Experiments

• in most experiments SMOTE most effective

• data argumentation without any specific knowledge about the implementation / dataset / distribution to balance datasets

• varying number of training samples in the profiling phase
  • Imbalanced: 1k, 10k, 50k
  • SMOTE: (approx) 5k, 24k, 120k
Dataset 1

- Low noise dataset - DPA contest v4 (publicly available)
- Atmel ATmega-163 smart card connected to a SASEBO-W board
- AES-256 RSM (Rotating SBox Masking)
- In this talk: mask assumed known
Data sampling techniques

- dataset 1: low noise unprotected
Dataset 2

- high noise dataset
- AES-128 on Xilinx Virtex-5 FPGA of a SASEBO GII evaluation board.
- publicly available on github: https://github.com/AESHD/AES HD Dataset
Data sampling techniques

- dataset 2: high noise unprotected
Dataset 3

- AES-128: Random delay countermeasure => misaligned
- 8-bit Atmel AVR microcontroller
- publicly available on github: https://github.com/ikizhvatov/randomdelays-traces
Data sampling techniques

- dataset 3: high noise with random delay
Further results

- additionally we tested SMOTE for CNN, MLP, TA:
  - also beneficial for CNN and MLP
  - not for TA (in this settings):
    - is not “tuned” regarding accuracy
    - may still benefit if #measurements is too low to build stable profiles (lower #measurements for profiling)
  - in case available: perfectly “natural”/chosen balanced dataset leads to better performance
  - … more details in the paper
Big Picture

1. Template building
   ML training

2. Template evaluation +
   max likelihood

   ML testing

   - Success rate
   - Guessing entropy

   - Accuracy
Evaluation metrics

- SR: average estimated probability of success
- GE: average estimated secret key rank
- depends on the number of traces used in the attacking phase
- average is computed over number of experiments
- ACC: average estimated probability (percentage) of correct classification
- average is computed over number of experiments
Evaluation metrics

- **SR**: average estimated probability of success
  - depends on the number of traces used in the attacking phase
  - *average* is computed over number of experiments

- **GE**: average estimated secret key rank

- **ACC**: average estimated probability (percentage) of correct classification
  - *average* is computed over number of experiments
Evaluation metrics

- **SR**: average estimated probability of success
- **GE**: average estimated secret key rank
- depends on the number of traces used in the attacking phase
- **average** is computed over number of experiments

- **ACC**: average estimated probability (percentage) of correct classification
- **average** is computed over number of experiments

indication: if acc high, GE/SR should "converge quickly"
SR/GE vs acc

Global acc vs class acc

- relevant for non-bijective function between class and key (e.g. class involved the HW)
- the importance to correctly classify more unlikely values in the class may be more significant than others
- accuracy is averaged over all class values

Label vs fixed key prediction

- relevant if attacking with more than 1 trace
- accuracy: each label is considered independently (along #measurements)
- SR/GE: computed regarding fixed key, accumulated over #measurements
- low accuracy may not indicate low SR/GE

more details, formulas, explanations in the paper...
Take away

• HW (HD) + ML is very likely to go wrong in noisy data!
  • data sampling techniques help to increase performances
  • more effective to collect less real sample + balancing techniques than collect more imbalanced samples
• ML metrics (accuracy) do not give a precise SCA evaluation!
  ✴ global vs class accuracy
  ✴ label vs fixed key prediction