Efficient Implementations of a Key Enumeration Algorithm

N. Veyrat-Charvillon, B. Gérard, M. Renauld and F-X. Standaert

UCL Crypto Group, Université catholique de Louvain

COSADE 2012, May 4
Standard DPA Attacks

Side-channel traces
Standard DPA Attacks

\[
\rho_{X,Y} = \frac{\text{cov}(X,Y)}{\sigma_X \sigma_Y}
\]
Standard DPA Attacks

- Side-channel traces
- Some kind of statistics
- Rank subkeys
**Standard DPA Attacks**

- Side-channel traces
- Some kind of statistics
- Rank subkeys
- Output key(s)
Key is incorrect!
Abort, Retry, Fail?
Let’s try again

Output another key, or two, or $2^{32}$!

Wait... how do we choose them?
Let’s try again

Output another key, or two, or $2^{32}$!

Wait... how do we choose them?
Let’s try again

Output another key, or two, or $2^{32}$!

Wait... how do we choose them?

![Diagram of side-channel cryptanalysis]

- Side-Channel Cryptanalysis
- Let’s try again
- Output another key, or two, or $2^{32}$!
- Wait... how do we choose them?

4/11
Let’s try again

Output another key, or two, or $2^{32}$!

Wait... how do we choose them?
Random sampling

\[ \begin{array}{c|c|c}
0 & k_2^{(j)} & 1 \\
\hline
k_1^{(j)} & & \\
\hline
1 & & \\
\end{array} \]
Random sampling

\[ k_1^{(j)} \]
Random sampling
Random sampling
Random sampling
Random sampling
Random sampling

The image shows a diagram with two rows and three columns. The first row is labeled as $k_1^{(j)}$ and the second row as $k_2^{(j)}$. Each column is labeled as 0, $k_2^{(j)}$, and 1. The diagram contains points scattered across the grid.
Random sampling

\[ k_1^{(j)} \]

\[ k_2^{(j)} \]
Random sampling
Random sampling
Random sampling

\[ k_1^{(j)} \]

\[ k_2^{(j)} \]

\[ 0 \quad 1 \]

\[ 0 \quad 1 \]
Optimal enumeration

\[ k_1^{(j)} \]

\[ k_2^{(j)} \]

\[ 0 \]

\[ 1 \]
Optimal enumeration

\[ k_1^{(j)} \downarrow \]

\[ k_2^{(j)} \]

\[ \begin{array}{c|c|c}
0 & k_2^{(j)} & 1 \\
\hline
1 & 2 & \end{array} \]

\[ \begin{array}{c|c|c|c}
0 & k_2^{(j)} & 1 & 2 \\
\hline
1 & 2 & \end{array} \]
Optimal enumeration

\[
\begin{array}{ccc}
0 & k_{2}^{(j)} & 1 \\
1 & \downarrow & \downarrow \\
\end{array}
\]

\[
\begin{array}{ccc}
1 & 3 \\
2 \\
1 \\
\end{array}
\]

\[
k_{1}^{(j)} \quad k_{2}^{(j)}
\]
**Optimal enumeration**

\[
\begin{array}{c|cc|c}
0 & \vdots & 1 \\
\hline
k_1^{(j)} & 2 & 4 & 1 \\
k_2^{(j)} & 1 & 3 & 1 \\
\hline
1 &  &  & \\
\end{array}
\]
Optimal enumeration
Success rate: optimal enumeration

![Graph showing the success rate for optimal enumeration against traces. The x-axis represents the number of traces, ranging from 50 to 250, and the y-axis represents the success rate, ranging from 0.0 to 1.0. The graph demonstrates an increasing success rate as the number of traces increases, with a shaded area indicating the success for 1 candidate.]
Success rate: optimal enumeration

The graph shows the success rate of optimal enumeration as a function of the number of traces. The x-axis represents the number of traces ranging from 50 to 250, and the y-axis represents the success rate ranging from 0.0 to 1.0. The graph indicates that the success rate increases as the number of traces increases, approaching 1.0 at approximately 200 traces for $2^4$ candidates.
Success rate: optimal enumeration

2^8 candidates

traces

success
Success rate: optimal enumeration

2^{12} candidates
Success rate: optimal enumeration

2^{16} candidates

success

traces
Success rate: optimal enumeration

The graph shows the success rate as a function of the number of traces. The x-axis represents the number of traces, ranging from 50 to 250, and the y-axis represents the success rate, ranging from 0.0 to 1.0. The number of candidates is $2^{20}$.
Success rate: optimal enumeration

2^{24} candidates
Success rate: optimal enumeration

2^{28} candidates

traces

success
Success rate: optimal enumeration

2^{32} candidates

success

traces
An enumeration tool

Open source C++11 efficient implementation

```
#include "Enumeration.h"
[...]
    Enumeration<> myEnum(keyPMF); // enumeration object
    bool found = false;
    unsigned long long rank = 0;
    pair<double, array<unsigned char, 0x10>> candidate; // key candidate
    while (!found) {
        candidate = myEnum.nextCandidate(); // enumerate next candidate
        found = testKey(candidate.second);
    }
```

Check out our website:

http://perso.uclouvain.be/fstandae/source_codes/enumeration/
## Enumeration performances

<table>
<thead>
<tr>
<th>#trials</th>
<th>$2^{16}$</th>
<th>$2^{20}$</th>
<th>$2^{24}$</th>
<th>$2^{28}$</th>
<th>$2^{32}$</th>
<th>$2^{33}$</th>
<th>$2^{35}$</th>
<th>$2^{37}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>[1]</td>
<td>0.96s</td>
<td>18.1s</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td></td>
<td>118MB</td>
<td>7.7GB</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Ours [2] open</td>
<td>0.01s</td>
<td>0.25s</td>
<td>5s</td>
<td>100s</td>
<td>1800s</td>
<td>3297s</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td></td>
<td>100KB</td>
<td>2MB</td>
<td>5.3MB</td>
<td>30MB</td>
<td>142MB</td>
<td>231MB</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>500KB</td>
<td>3MB</td>
<td>37MB</td>
<td>267MB</td>
<td>2.8GB</td>
<td>4.6GB</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Ours [2] optimized</td>
<td>0.01s</td>
<td>0.25s</td>
<td>5s</td>
<td>100s</td>
<td>1700s</td>
<td>4058s</td>
<td>5.4h</td>
<td>28.2h</td>
</tr>
<tr>
<td></td>
<td>100KB</td>
<td>2MB</td>
<td>3.9MB</td>
<td>30MB</td>
<td>140MB</td>
<td>190MB</td>
<td>560MB</td>
<td>0.9GB</td>
</tr>
<tr>
<td></td>
<td>500KB</td>
<td>3MB</td>
<td>12MB</td>
<td>80MB</td>
<td>530MB</td>
<td>540MB</td>
<td>1.1GB</td>
<td>+1.7HDD</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>+2.3HDD</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>+6.1HDD</td>
</tr>
</tbody>
</table>


Exploiting parallelism

Parallel implementation:
splits enumeration workload between threads

<table>
<thead>
<tr>
<th>#threads</th>
<th>speedup</th>
<th>memory overhead</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>1.3</td>
<td>1-4</td>
</tr>
<tr>
<td>4</td>
<td>2</td>
<td>1-3</td>
</tr>
<tr>
<td>6</td>
<td>2-3</td>
<td>1-3</td>
</tr>
</tbody>
</table>
Attacks versus security evaluation

Key enumeration:
1. attack tool
2. evaluation of SR and GE limited by enumeration (e.g. up to rank $2^{40}$)

But security should rely on ranks too large to enumerate ($\gg 2^{40}$)
⇒ How can an evaluator compute these ranks?

Stay Tuned!