Differential Computation Analysis
Hiding your White-Box Designs is Not Enough

Joppe W. Bos
Summer school on real-world crypto and privacy
Šibenik, Croatia
NXP Semiconductors

Operations in > 35 countries, more than 130 facilities
≈ 45,000 employees

Research & Development
≈ 11,200 engineers in 23 countries
The presence of an attacker

- Where should we assume the attacker to be? What is most realistic?
  - Is the attacker only eavesdropping on the communication channel?
  - Or did one of the (trusted/authorized) end-users become the attacker?
  - Or are there any malware/viruses installed on a trusted end-user’s device?

![Diagram of encryption and decryption process with an attacker in the middle.]
Cryptography & Security Notions

- In order to properly assess the security of (the implementation of) a cryptographic algorithm, one needs a clear definition of a security notion.

- **Security Notion = attacker’s goal + attacker model.**
  - *Attacker’s goal:* what does the attacker want to achieve?
    - This is not always key-extraction, the attacker is often satisfied with much less…
  - *Attacker model:* what are the capabilities of the attacker in order for him to achieve his goal?
    - Such a model tries to capture the capabilities of an attacker as realistically as possible, i.e., modeling the hostile environment in which the implementation of a cryptographic primitive is deployed.
Black box model

Initial cryptographic security model from the 1980s

- Endpoints are trusted parties
- Attacker “observes” data being transferred
When technology changed this model did not reflect reality any longer

Cryptographic algorithms implemented in hardware were originally thought to form a secure environment

In 1999 it was publicly shown that hardware implementations tend to leak key-correlated information

Kocher, Jaffe, Jun. Differential power analysis. In CRYPTO 1999
The research area of side-channel attacks and resistance has grown significantly: fault injections, simple power analysis, differential power analysis, correlation power analysis, template attacks, higher-order correlation attacks, mutual information analysis, linear regression analysis, horizontal analysis, vertical analysis etc. etc. See the CHES conference.
Grey box model → white box model

- When technology changed this model did not reflect reality any longer
- Increase in mobile devices without dedicated hardware support → need to rely on software solutions
- In 2002 the white-box model was introduced
  Initial focus on DRM applications.

White box model

![Diagram of encryption/decryption process with adversary capabilities]

Adversary owns the device running the software. Powerful capabilities

- has full access to the source code
- perform static analysis
- inspect and alter the memory used
- alter intermediate results
Applications of WB crypto has evolved to protection of:
- digital assets
- mobile device (from an application store)
- Host Card Emulation (HCE)
- credentials for an authentication to the cloud

White box crypto - applications

Applications of WB crypto has evolved to protection of:
- digital assets
- mobile device (from an application store)
- Host Card Emulation (HCE)
- credentials for an authentication to the cloud

How to realize a white-box implementation in practice?
"when the attacker has internal information about a cryptographic implementation, choice of implementation is the sole remaining line of defense"

White-Box basic idea – Why?

- **Entropy attack**
  - Locate the unusual high entropy of the cryptographic key in a memory dump using sliding windows for example.

  | 0-bit | 1-bit |

Shamir, van Someren: *Playing "Hide and Seek" with Stored Keys*. Financial Cryptography 1999
White-Box basic idea – Why?

- **Entropy attack**
  - Locate the unusual high entropy of the cryptographic key in a memory dump using sliding windows for example.

- **S-box blanking attack**
  - Locate the publicly defined S-boxes in the binary and overwrite it with all zeros such that \( S(x) = 0 \) for any \( x \).

Shamir, van Someren: *Playing "Hide and Seek" with Stored Keys*. Financial Cryptography 1999

Kerins, Kursawe: *A cautionary note on weak implementations of block ciphers*. WISSec, 2006
Security of WB solutions - Theory

White box can be seen as a form of code obfuscation
- It is known that obfuscation of any program is impossible

Barak, Goldreich, Impagliazzo, Rudich, Sahai, Vadhan, Yang. On the (im)possibility of obfuscating programs. In CRYPTO 2001

- Unknown if a (sub)family of white-box functions can be obfuscated
- If secure WB solution exists then this is protected (by definition!) to all current and future side-channel and fault attacks!
Security of WB solutions - Theory

White box can be seen as a form of code obfuscation

- It is known that obfuscation of any program is impossible

Barak, Goldreich, Impagliazzo, Rudich, Sahai, Vadhan, Yang. On the (im)possibility of obfuscating programs. In CRYPTO 2001

- Unknown if a (sub)family of white-box functions can be obfuscated
- If secure WB solution exists then this is protected (by definition!) to all current and future side-channel and fault attacks!

Practice

- Only results known for symmetric crypto (all academic designs broken)
- Convert algorithms to sequence of LUTs
- Embed the secret key in the LUTs
- Obfuscate the LUTs by using encodings
WB Impossible?

No! “Ideal” WB AES implementation
One big lookup table $\rightarrow 2^{92}$ TB storage required

Practical WB AES?
Network of smaller tables: $\approx 700$ kB
Encoding on intermediate values using ideas by Chow


Generic idea.
Transform a cipher into a network of randomized key-instantiated look-up tables
AES with look-up tables

- The key addition and S-box operations are merged into a single operation (8 bit $\rightarrow$ 8 bit table $\rightarrow$ 256 byte) $c_{i,j} = Sbox(a_{i,j} \oplus k_{i,j}) = T_{i,j}(a_{i,j})$

- To simplify: we omit ShiftRow operation
  - Corresponds to renumbering of indices

- The MixColumn operation can be split into four byte-to-32-bit (8 bit $\rightarrow$ 32 bit table $\rightarrow$ 1024 byte) operations:

\[
\bar{d}_j = M_0T_{0,j}(a_{0,j}) \oplus M_1T_{1,j}(a_{1,j}) \oplus M_2T_{2,j}(a_{2,j}) \oplus M_3T_{3,j}(a_{3,j})
\]

- We can now implement a round by only using the following 2 types of lookup tables:
AES with look-up tables + obfuscation

- Since S-boxes and matrix $M$ are known, the key can easily be extracted from the lookup tables.
- **Solution**: obfuscating lookup tables by encoding their input and output.
AES with look-up tables + obfuscation

- Since S-boxes and matrix $M$ are known, the key can easily be extracted from the lookup tables.

- **Solution:** obfuscating lookup tables by encoding their input and output.

- First, we apply **linear** encodings:
  - $A_i$: random 8-bit linear mapping
  - $MB$: random 32-bit linear mapping

\[
A_i^{-1} \cdot a_{i,j} \rightarrow A_i \rightarrow T_{i,j} \rightarrow MB \cdot M_i \rightarrow MB \cdot d_j
\]
AES with look-up tables + obfuscation

- Since S-boxes and matrix $M$ are known, the key can easily be extracted from the lookup tables.

- **Solution:** obfuscating lookup tables by encoding their input and output.

- First, we apply **linear** encodings:
  - $A_i$: random 8-bit linear mapping
  - $MB$: random 32-bit linear mapping

- Matrix $MB$ is removed from the computed output columns. Implemented in the same way as the MixColumn operations
  \[
  MB^{-1}(x) = MB_0^{-1}(x_0) \oplus MB_1^{-1}(x_1) \oplus MB_2^{-1}(x_2) \oplus MB_3^{-1}(x_3)
  \]

- Merge the $MB_i$-tables by the linear encodings used in the next round.
AES with look-up tables + obfuscation - Flow

\[ A_i^{-1} \cdot a_{i,j} \]

\[ A_i \]
\[ T_{i,j} \]
\[ MB \cdot M_j \]
\[ MB \cdot \bar{d}_j \]

\[ MB_j^{-1} \]
\[ A_0^{-1} A_1^{-1} A_2^{-1} A_3^{-1} \]

\[ A_i^{-1} \cdot d_{i,j} \]
Obfuscation, obfuscation, obfuscation

- In addition to the *linear* encodings, also add **non-linear** encodings \( f \).

Table

\[
\begin{array}{c}
U \\
\end{array}
\quad \begin{array}{c}
\rightarrow \\
\end{array}
\quad \begin{array}{c}
U \\
\end{array}
\]

followed by

\[
\begin{array}{c}
V \\
\end{array}
\quad \begin{array}{c}
\rightarrow \\
\end{array}
\quad \begin{array}{c}
V \\
\end{array}
\]

followed by

\[
\begin{array}{c}
f \\
\end{array}
\quad \begin{array}{c}
f' \\
\end{array}
\]

Obfuscation, obfuscation, obfuscation

- In addition to the linear encodings, also add non-linear encodings $f$.

Size implementation: $\approx 700$ kB

In practice the white box is the most essential but a **small part** of the entire software implementation:

- Strong code obfuscation
- Binary is “glued” to the environment
  - Prevent code-lifting
- Support for traitor tracing
- Mechanism for frequent updating

More details see the invited talk at EC 2016 *Engineering Code Obfuscation* by Christian Collberg

**Focus on the white-box only**
White box crypto - practice

• White-box “solutions” are known for standard symmetric crypto only

• All published (academic) designs have been theoretically broken

So what can do to improve the attacks?
Effort and expertise required

- Previous WB attacks were WB approach specific which means:
  - know the type of encodings that are applied on the *intermediate results*
  - know which *cipher operations* are implemented by which *(network of)* lookup tables
  - This implies **time-consuming reverse-engineering** of the code and then applying a sophisticated (algebraic) attack
Effort and expertise required

- Previous WB attacks were WB approach specific which means
  - know the type of encodings that are applied on the *intermediate results*
  - know which *cipher operations* are implemented by
    which *(network of)* lookup tables
  - This implies **time-consuming reverse-engineering** of the code
    and then applying a sophisticated (algebraic) attack

- Our attack allows to assess the security of a WB implementation
  - Automatically
  - Without knowledge of the implementation choices / details
    of the underlying scheme (just the scheme itself)
  - Ignores all (attempts) at code-obfuscation
  - No expertise required (execute our scripts and wait for the key)
SOFTWARE TRACES
Tracing binaries

- Academic attacks are on open design
- In practice: what you get is a binary blob
  - No design documents
  - No algorithm specification
  - No source code

Idea: Does the WB leak info?

In other words:
Verify if one can correlate guesses to some intermediate results (which use the key) using software traces with the help of

*dynamic binary instrumentation* tools
Tracing binaries

- Academic attacks are on open design
- In practice: what you get is a binary blob

Idea: create software traces using *dynamic binary instrumentation* tools

- Record all instructions and memory accesses.

Examples of the tools we extended / modified

- Intel PIN (x86, x86-64, Linux, Windows, Wine/Linux)
- Valgrind (idem+ARM, Android)
Tracing binaries

- Academic attacks are on open design
- In practice: what you get is a binary blob

Idea: create software traces using *dynamic binary instrumentation* tools

- Record all instructions and memory accesses.

Examples of the tools we extended / modified
  - Intel PIN (x86, x86-64, Linux, Windows, Wine/Linux)
  - Valgrind (idem+ARM, Android)

- Using traces:
  1. One trace: Visual identification of white-box, code-/table-lifting
  2. Few traces: data correlation, standard deviation, etc
  3. More traces: DPA-based attack
Trace visualization convention: pTra waterfall
Visual crypto identification: code
Visual crypto identification: code?
Visual crypto identification: code? data!

1 + 15
Visual crypto identification: code? data?
Visual crypto identification: stack!
Where is the key?
Differential Power Analysis and friends


Very powerful grey box attack!

Requirements
- known input or known output
- ability to trace power consumption (or EM radiations, or …)

For example in AES: $\text{SubBytes}(p \oplus \kappa)$
Differential Computation Analysis

Port the white-box to a smartcard and measure power consumption
Differential Computation Analysis

Port the white-box to a smartcard and measure power consumption
Make pseudo power traces from our software execution traces
→ this are lists of memory accesses / data + stack writes / …

E.g. build a trace of all 8-bit data reads:

→ 256 possible discrete values
Differential Computation Analysis

256 possible discrete values but bit values dominated by the MSB

→ Build Hamming weight traces?

→ 8 possible discrete values

That works but we can do better…

recall: Hamming weight was a **hardware model** for combined bit leaks
Differential Computation Analysis

Each bit of those bytes is equally important address bits represent a different way to partition the look-up tables

→ Serialize bytes in a succession of bits

→ 2 possible discrete values: 0's and 1's
DCA: DPA on software traces

HW analogue: this is like probing each bus-line individually \textit{without any error}
Results

WB implementations should not leak any side-channel information (by definition of the WB attack model): let’s check!

<table>
<thead>
<tr>
<th>WB implementation</th>
<th>Algorithm</th>
<th>#traces</th>
</tr>
</thead>
<tbody>
<tr>
<td>Wyseur challenge, 2007</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Hack.lu challenge, 2009</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SSTIC challenge, 2012</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Klinec implementation, 2013</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Chow+: Chow-based plus personal improvements by Brecht Wyseur

E. Link and W. D. Neumann. Clarifying obfuscation: Improving the security of white-box DES. In International Symposium on Information Technology: Coding and Computing (ITCC 2005)
Results

WB implementations should not leak any side-channel information (by definition of the WB attack model): let’s check!

<table>
<thead>
<tr>
<th>WB implementation</th>
<th>Algorithm</th>
<th>#traces</th>
</tr>
</thead>
<tbody>
<tr>
<td>Wyseur challenge, 2007</td>
<td>DES (Chow+)</td>
<td>65</td>
</tr>
<tr>
<td>Hack.lu challenge, 2009</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SSTIC challenge, 2012</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Klinec implementation, 2013</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Hack.lu challenge

Zoom on the stack

✓ AES-128
✓ Very easy to break
  (designed for a one-day challenge)
Results

WB implementations should not leak any side-channel information (by definition of the WB attack model): let’s check!

<table>
<thead>
<tr>
<th>WB implementation</th>
<th>Algorithm</th>
<th>#traces</th>
</tr>
</thead>
<tbody>
<tr>
<td>Wyseur challenge, 2007</td>
<td>DES (Chow+)</td>
<td>65</td>
</tr>
<tr>
<td>Hack.lu challenge, 2009</td>
<td>AES (Chow)</td>
<td>16 (no encodings)</td>
</tr>
<tr>
<td>SSTIC challenge, 2012</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Klinec implementation, 2013</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Results

WB implementations should not leak any side-channel information (by definition of the WB attack model): let’s check!

<table>
<thead>
<tr>
<th>WB implementation</th>
<th>Algorithm</th>
<th>#traces</th>
</tr>
</thead>
<tbody>
<tr>
<td>Wyseur challenge, 2007</td>
<td>DES (Chow+)</td>
<td>65</td>
</tr>
<tr>
<td>Hack.lu challenge, 2009</td>
<td>AES (Chow)</td>
<td>16 (no encodings)</td>
</tr>
<tr>
<td>SSTIC challenge, 2012</td>
<td>DES</td>
<td>16 (no encodings)</td>
</tr>
<tr>
<td>Klinec implementation, 2013</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
- AES using Karroumi’s approach (using dual ciphers)
- More difficult, not all correct key bytes are #1

M. Karroumi. Protecting white-box AES with dual ciphers. In ICISC 2011
**Balanced encodings?**

- It may become the *least* candidate, this is still standing out!

<table>
<thead>
<tr>
<th>Target bit</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>
<th>9</th>
<th>10</th>
<th>11</th>
<th>12</th>
<th>13</th>
<th>14</th>
<th>15</th>
</tr>
</thead>
<tbody>
<tr>
<td>Key byte</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1</td>
<td>256</td>
<td>256</td>
<td>239</td>
<td>256</td>
<td>1</td>
<td>1</td>
<td></td>
<td>255</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1</td>
<td>256</td>
<td>256</td>
<td>5</td>
<td>1</td>
<td>256</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>256</td>
<td>1</td>
<td>255</td>
<td>256</td>
<td>256</td>
<td>256</td>
<td>226</td>
<td>256</td>
<td>256</td>
<td>256</td>
<td>22</td>
<td>1</td>
<td>256</td>
<td>256</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>256</td>
<td>255</td>
<td>251</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>254</td>
<td>1</td>
<td>1</td>
<td>256</td>
<td>256</td>
<td>234</td>
<td>256</td>
<td>256</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>256</td>
<td>256</td>
<td>74</td>
<td>256</td>
<td>256</td>
<td>256</td>
<td>255</td>
<td>256</td>
<td>254</td>
<td>256</td>
<td>1</td>
<td>256</td>
<td>256</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>50</td>
<td>256</td>
<td>253</td>
<td>1</td>
<td>251</td>
<td>256</td>
<td>253</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td></td>
<td>254</td>
<td>1</td>
<td>1</td>
<td>256</td>
<td>254</td>
<td>256</td>
<td>248</td>
<td>256</td>
<td>252</td>
<td>256</td>
<td>1</td>
<td>14</td>
<td>255</td>
<td>256</td>
<td>250</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>256</td>
<td>1</td>
<td>1</td>
<td>252</td>
<td>256</td>
<td>253</td>
<td>256</td>
<td>256</td>
<td>255</td>
<td>256</td>
<td>1</td>
<td>251</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
</tbody>
</table>

**Correct key, ranked #256**
Table 1. DCA ranking for a Karroumi white-box implementation when targeting the output of the *SubBytes* step in the first round based on the least significant address byte on memory reads.

<table>
<thead>
<tr>
<th>target bit</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>
<th>9</th>
<th>10</th>
<th>11</th>
<th>12</th>
<th>13</th>
<th>14</th>
<th>15</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>256</td>
<td>255</td>
<td>256</td>
<td>255</td>
<td>256</td>
<td>253</td>
<td>1</td>
<td>256</td>
<td>256</td>
<td>239</td>
<td>256</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>255</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>256</td>
<td>256</td>
<td>256</td>
<td>1</td>
<td>255</td>
<td>256</td>
<td>1</td>
<td>1</td>
<td>5</td>
<td>1</td>
<td>256</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>256</td>
<td>1</td>
<td>255</td>
<td>256</td>
<td>1</td>
<td>253</td>
<td>226</td>
<td>256</td>
<td>256</td>
<td>256</td>
<td>1</td>
<td>256</td>
<td>22</td>
<td>1</td>
<td>256</td>
<td>256</td>
</tr>
<tr>
<td>3</td>
<td>256</td>
<td>255</td>
<td>251</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>254</td>
<td>1</td>
<td>1</td>
<td>256</td>
<td>256</td>
<td>253</td>
<td>1</td>
<td>254</td>
<td>256</td>
<td>256</td>
</tr>
<tr>
<td>4</td>
<td>256</td>
<td>256</td>
<td>256</td>
<td>256</td>
<td>256</td>
<td>256</td>
<td>256</td>
<td>256</td>
<td>1</td>
<td>256</td>
<td>256</td>
<td>1</td>
<td>1</td>
<td>256</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>5</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>50</td>
<td>256</td>
<td>253</td>
<td>1</td>
<td>251</td>
<td>256</td>
<td>253</td>
<td>1</td>
<td>256</td>
<td>256</td>
</tr>
<tr>
<td>6</td>
<td>254</td>
<td>1</td>
<td>1</td>
<td>256</td>
<td>254</td>
<td>256</td>
<td>248</td>
<td>256</td>
<td>252</td>
<td>256</td>
<td>1</td>
<td>14</td>
<td>255</td>
<td>256</td>
<td>250</td>
<td>1</td>
</tr>
<tr>
<td>7</td>
<td>1</td>
<td>256</td>
<td>1</td>
<td>1</td>
<td>252</td>
<td>256</td>
<td>253</td>
<td>256</td>
<td>256</td>
<td>256</td>
<td>1</td>
<td>251</td>
<td>1</td>
<td>254</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>All</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
</tbody>
</table>

Table 2. DCA ranking for a Karroumi white-box implementation when targeting the output of the multiplicative inversion inside the *SubBytes* step in the first round based on the least significant address byte on memory reads.

<table>
<thead>
<tr>
<th>target bit</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>
<th>9</th>
<th>10</th>
<th>11</th>
<th>12</th>
<th>13</th>
<th>14</th>
<th>15</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>256</td>
<td>256</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>256</td>
<td>256</td>
<td>256</td>
<td>254</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>255</td>
<td>256</td>
<td>256</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>253</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>256</td>
<td>249</td>
<td>256</td>
<td>256</td>
<td>256</td>
<td>226</td>
<td>1</td>
<td>254</td>
<td>256</td>
<td>256</td>
</tr>
<tr>
<td>2</td>
<td>256</td>
<td>256</td>
<td>1</td>
<td>1</td>
<td>255</td>
<td>256</td>
<td>256</td>
<td>256</td>
<td>251</td>
<td>1</td>
<td>255</td>
<td>256</td>
<td>1</td>
<td>1</td>
<td>254</td>
<td>256</td>
</tr>
<tr>
<td>3</td>
<td>254</td>
<td>1</td>
<td>69</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>252</td>
<td>256</td>
<td>1</td>
<td>256</td>
<td>1</td>
<td>256</td>
<td>256</td>
</tr>
<tr>
<td>4</td>
<td>254</td>
<td>1</td>
<td>255</td>
<td>256</td>
<td>256</td>
<td>1</td>
<td>255</td>
<td>256</td>
<td>256</td>
<td>1</td>
<td>256</td>
<td>256</td>
<td>1</td>
<td>1</td>
<td>256</td>
<td>256</td>
</tr>
<tr>
<td>5</td>
<td>254</td>
<td>256</td>
<td>250</td>
<td>1</td>
<td>241</td>
<td>256</td>
<td>255</td>
<td>3</td>
<td>1</td>
<td>1</td>
<td>256</td>
<td>256</td>
<td>231</td>
<td>256</td>
<td>208</td>
<td>254</td>
</tr>
<tr>
<td>6</td>
<td>256</td>
<td>256</td>
<td>256</td>
<td>256</td>
<td>256</td>
<td>233</td>
<td>256</td>
<td>1</td>
<td>256</td>
<td>1</td>
<td>1</td>
<td>256</td>
<td>256</td>
<td>1</td>
<td>1</td>
<td>241</td>
</tr>
<tr>
<td>7</td>
<td>63</td>
<td>256</td>
<td>1</td>
<td>256</td>
<td>1</td>
<td>255</td>
<td>231</td>
<td>256</td>
<td>255</td>
<td>1</td>
<td>255</td>
<td>256</td>
<td>255</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>All</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
</tbody>
</table>
WB implementations should not leak any side-channel information (by definition of the WB attack model): let’s check!

<table>
<thead>
<tr>
<th>WB implementation</th>
<th>Algorithm</th>
<th>#traces</th>
</tr>
</thead>
<tbody>
<tr>
<td>Wyseur challenge, 2007</td>
<td>DES (Chow+)</td>
<td>65</td>
</tr>
<tr>
<td>Hack.lu challenge, 2009</td>
<td>AES (Chow)</td>
<td>16 (no encodings)</td>
</tr>
<tr>
<td>SSTIC challenge, 2012</td>
<td>DES</td>
<td>16 (no encodings)</td>
</tr>
<tr>
<td>Klinec implementation, 2013</td>
<td>AES (Karroumi, dual ciphers)</td>
<td>2000 → 500</td>
</tr>
</tbody>
</table>
Countermeasures?

Academic remedies

• Cannot rely on random data in the white-box attack model

• Use static random data within the white-box itself?

• Use ideas from threshold implementation?
  • masking scheme based on secret sharing and multi-party computation


Practical remedy

• simply strengthen other measures
  • anti-debug / detect DBI frameworks, code-obfuscation (?), integrity checks, platform binding, etc
All help to make our DCA / DFA or CPA tools more powerful is highly appreciated!
Conclusions and future work

- Software-only solutions are becoming more popular
  - white-box crypto

- Use-cases shifted from DRM to HCE (payment, transit, …)

- Level of security / maturity of many (all?) WB schemes is questionable
  - Open problem to construct asymmetric WB crypto
  - Industry keeps design secret

- DCA is an automated attack which can be carried out without any expertise
  - Counterpart of the SCA from the crypto HW community

- We used DPA, what about FA, CPA, higher-order attacks etc?
  - See our github and the great work by Riscure
    (see the training this afternoon)

Eloi Sanfelix Gonzalez, Cristofaro Mune, Job de Haas: *Unboxing the White-Box: Practical Attacks Against Obfuscated Ciphers*. Black Hat Europe 2015.