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

Joppe W. Bos
Who am I

✓ Finished PhD@laboratory for cryptologic algorithms at EPFL, Lausanne, Switzerland under supervision of Prof. Arjen Lenstra in 2012
  • Computational cryptanalysis: RSA768, ECM, ECDLP112, …

✓ Post-doctoral researcher in the Cryptography Research Group at Microsoft Research, Redmond, USA
  • Fully / Somewhat homomorphic encryption: YASHE
  • Fast curve crypto: Genus 2 cryptography

✓ From 2014, cryptographic researcher at NXP Semiconductors, Leuven
  • Applied cryptography: elliptic curves and **white-box cryptography**
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?
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.
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.

Adversary owns the device running the software. Powerful capabilities

- has full access to the source code
- inspect and alter the memory used
- perform static analysis
- alter intermediate results
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

Embed the secret key in the implementation.
White-Box basic idea

Embed the secret key in the implementation.

Entropy attack by Shamir and van Someren (1999)
- Locate the unusual high entropy of the cryptographic key in a memory dumb using sliding windows for example.

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

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

**Practical WB AES?**
Network of smaller tables: 752 kB
Encoding on intermediate values using ideas by Chow


**Generic idea.**
Transform a cipher into a network of randomized key-instantiated look-up tables
High-level approach

- Use tables rather than individual steps
- Encode tables with random bijections \((g \circ L_i \circ f^{-1})\)
- The usage of a fixed secret key is embedded in these tables

Code generator generates random white-box implementations \(\rightarrow\) lots (10s of thousands) LUTs \(\rightarrow\) performance impact

Figure from: A Tutorial on White-box AES, James A. Muir, Cryptology ePrint Archive, Report 2013/104
White box crypto - practice

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

Remainder of the talk

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
  - High-level of expertise required
  - Need to know which approach is implemented
  - Targets specific s-box or range of LUTs
- Our new attack allows to assess the security of a WB implementation
  - Automatically
  - Without knowledge of the underlying scheme
  - Ignores all (attempts) at code-obfuscation, anti-debugging etc
  - No expertise required
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

- Code
- Stack
- Data

Instruction
- Mem read
- Mem write
- Mem r+w

Time

Memory addresses
Visual crypto identification: code

9x4
Visual crypto identification: code?
Visual crypto identification: code? data!

1+15
Visual crypto identification: code? data?
Visual crypto identification: stack!

1+15
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: $SubBytes(p \oplus \kappa)$
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 *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
Balanced encodings?

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

---

### Correct Key: Ranked #256

- The image contains a table showing a key byte distribution with a significant concentration around the value 256, indicating the correct key. The table is highlighted to emphasize the key byte frequency distribution.

- The rightmost column of the table is marked with a red circle, drawing attention to the key byte 256.

- This concentration around 256 suggests that the key ranked #256 is likely the correct one, as it has a higher frequency of occurrence in the key byte distribution.
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.

| target bit | 0  | 1  | 2  | 3  | 4  | 5  | 6  | 7  | 8  | 9  | 10 | 11 | 12 | 13 | 14 | 15 |
|------------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| 0          | 1  | 256| 255| 256| 255| 256| 253| 1  | 1  | 256| 256| 239| 256| 1  | 1  | 255|
| 1          | 1  | 256| 256| 256| 1  | 255| 256| 1  | 1  | 5  | 1  | 256| 1  | 1  | 1  | 1  |
| 2          | 256| 255| 255| 256| 1  | 256| 226| 256| 256| 256| 256| 1  | 256| 22 | 1  | 256| 256|
| 3          | 256| 255| 251| 1  | 1  | 1  | 254| 1  | 1  | 256| 256| 234| 256| 254| 256| 255| 256|
| 4          | 256| 256| 74 | 256| 256| 255| 256| 255| 254| 256| 256| 256| 1  | 1  | 256| 1  |
| 5          | 1  | 1  | 1  | 1  | 1  | 5  | 256| 253| 1  | 1  | 251| 256| 253| 1  | 1  | 256| 256|
| 6          | 254| 1  | 1  | 256| 254| 256| 248| 256| 252| 256| 1  | 14 | 255| 256| 250| 1  |
| 7          | 1  | 256| 1  | 1  | 252| 256| 253| 256| 256| 255| 256| 1  | 251| 1  | 254| 1  |
| All        | ✓  | ✓  | ✓  | ✓  | ✓  | ✓  | ✓  | ✓  | ✓  | ✓  | ✓  | ✓  | ✓  | ✓  | ✓  | ✓  |

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.

| target bit | 0  | 1  | 2  | 3  | 4  | 5  | 6  | 7  | 8  | 9  | 10 | 11 | 12 | 13 | 14 | 15 |
|------------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| 0          | 256| 256| 1  | 1  | 1  | 256| 256| 256| 254| 1  | 1  | 1  | 255| 256| 256| 1  |
| 1          | 1  | 1  | 253| 1  | 1  | 256| 249| 256| 256| 256| 226| 1  | 1  | 254| 256| 256|
| 2          | 256| 256| 1  | 1  | 255| 256| 256| 256| 251| 1  | 1  | 1  | 254| 256| 1  |
| 3          | 254| 1  | 69 | 1  | 1  | 1  | 1  | 1  | 252| 256| 1  | 1  | 256| 1  | 1  |
| 4          | 254| 1  | 255| 256| 256| 1  | 255| 256| 1  | 1  | 256| 256| 238| 256| 253| 256|
| 5          | 254| 256| 250| 1  | 241| 256| 255| 3  | 1  | 1  | 256| 256| 231| 256| 208| 254|
| 6          | 256| 256| 256| 256| 1  | 256| 256| 1  | 1  | 1  | 256| 256| 1  | 1  | 241| 1  |
| 7          | 63 | 256| 1  | 256| 1  | 255| 231| 256| 255| 1  | 1  | 1  | 1  |
| All        | ✓  | ✓  | ✓  | ✓  | ✓  | ✓  | ✓  | ✓  | ✓  | ✓  | ✓  | ✓  | ✓  | ✓  | ✓  | ✓  |
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, code-obfuscation, integrity checks, platform binding, detect DBI frameworks, etc
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?
References


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

• Plan to release our DCA tools soon!
SECURE CONNECTIONS FOR A SMARTER WORLD