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

Joppe W. Bos, Charles Hubain, Wil Michiels, and Philippe Teuwen
CHES 2016
August 18, 2016, Santa-Barbara, California, USA
Who is the attacker? External adversary, user, virus?
Where should we assume the attacker to be? What is realistic?

Endpoints are trusted parties
Attacker “observes” data being transferred
Who is the attacker? External adversary, user, virus?
Where should we assume the attacker to be? What is realistic?

Endpoints are trusted parties
Attacker “observes” data being transferred

This is why you attend this conference!
Who is the attacker? External adversary, user, virus?
Where should we assume the attacker to be? What is realistic?

Endpoints are trusted parties
Attacker “observes” data being transferred

This is why you attend this conference!

Adversary owns the device running the software.
Where is this used in practice?

Original use-case for white-box crypto is *digital right management*.

For example: streaming content, protecting DVD’s etc.
Where is this used in practice?

Original use-case for white-box crypto is *digital right management*.

For example: streaming content, protecting DVD’s etc

**Recent trend**

Use *Host Card Emulation (HCE)* to communicate using *Near Field Communication (NFC)*

→ Replace the secure element with software.

Protection of the cryptographic key? How? **White-box implementation!**

*Source: Business Insider*
Huge demand for practical + secure white-box

- 2014: VISA + Mastercard support HCE
- [Berg Insight]: 86% of the Point of Sale devices in North America and 78% in Europe will support NFC by 2017.
- [IHS research]: By 2018, 2/3 of all shipped phones will support NFC.
- → the protocols used need to use (and store!) AES / DES keys → need for secure white-box cryptography.
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 of standard crypto broken)
- Convert algorithms to sequence of LUTs
- Embed the secret key in the LUTs
- Obfuscate the LUTs by using encodings
Obfuscating the LUTs

\[(f_{0,i}, f_{1,i}) A_i^{-1} \cdot a_{i,j}\]

\[\begin{array}{c}
  A_i \\
  T_{i,j} \\
  MB \cdot M_i \\
  MB_i^{-1} \\
  A_0 \\
  A_1 \\
  A_2 \\
  A_3
\end{array}\]

\[(f_{0,i}, f_{1,i}) A_i^{-1} \cdot d_{i,j}\]


Size of 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
Effort and expertise required

**Previous effort**
Previous WB attacks were **WB specific** which means knowing
- the **encodings**
- which *cipher operations* are implemented by
- which (network of) **lookup tables**

**Attack**
1. time-consuming **reverse-engineering** of the code
2. identify which WB scheme is used + target the correct LUTs
3. apply an algebraic attack
Effort and expertise required

Previous effort
Previous WB attacks were WB specific which means knowing
- the encodings
- which cipher operations are implemented by
- which (network of) lookup tables

Attack
1. time-consuming reverse-engineering of the code
2. identify which WB scheme is used + target the correct LUTs
3. apply an algebraic attack

Our approach
Assess the security of a WB implementation
- Automatically and very simply (see CHES challenge)
- Without knowledge of any implementation choices
  → only the algorithm itself
- Ignores all (attempts) at code-obfuscation
Tracing binaries

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

**Idea:** collect information using using *dynamic binary instrumentation* tools (→ visual representation → use traces to find correlation)

- 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)
Trace visualization

Based on Ptra, an unreleased Quarkslab tool presented at SSTIC 2014
Visual crypto identification: code
Visual crypto identification: code?
Visual crypto identification: code? data!
Visual crypto identification: code? data?
Visual crypto identification: stack!
Naïve approach: Port the white-box to a smartcard and measure power consumption.
Differential Computation Analysis

**Naïve approach**: Port the white-box to a smartcard and measure power consumption

**Better approach**: each bit is equally important

→ Serialize bytes in a succession of bits
Differential Computation Analysis

**Naïve approach**: Port the white-box to a smartcard and measure power consumption

**Better approach**: each bit is equally important

→ Serialize bytes in a succession of bits

**Visual challenge**: try to identify the rounds

(Hint: auto-correlation can reveal them!)
DCA: DPA on software traces

HW analogy: 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>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>

Intuition why this works:
Encodings do not sufficiently hide correlations when the correct key is used.

A lot of potential for follow-up work!

Use the extended research results from the grey box world

**Countermeasures**
- Use random masks / delays → white-box model allows to disable entropy source
- Use static random data within the white-box itself?
- Use ideas from threshold implementation? [TI]
- Better DBI framework detection mechanisms
- DCA might fail when using large encodings → larger LUTs → algebraic attacks still work


**Other attacks**
Riscure has proven software fault attacks (DFA) work too [RISCURE].
Once there are countermeasures against DCA and DFA, can we use any of the other known advanced SCA in this setting?

Any help to complete our collection of open whitebox challenges and attacks or to improve our tools is highly appreciated!
Conclusions

- Software-only solutions are becoming more popular
  - white-box crypto
- Traditional (DRM) and new use-cases 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 DPA from the crypto HW community
- This hopefully sparkles more interest in both cryptographic and cryptanalytic white-box research!