
White-Box Cryptography
Don't Forget About Grey Box Attacks

Joppe W. Bos
Real World Crypto 2017
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

Hardware implementations tend to leak key-correlated information
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.

Hardware implementations tend to leak key-correlated information.

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!**
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 deployed protocols use (and store!) AES / DES keys → need for secure white-box cryptography.
Why not use “normal” crypto software code?

- 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
Why not use “normal” crypto software code?

- **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
White-Box in Practice

**White-Box theoretically Impossible?**

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

**In practice**

Network of smaller tables:
≈ 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

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 the corresponding 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 the corresponding 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!

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

**Naive approach**: Port the white-box to a smartcard and measure power consumption.
Differential Computation Analysis

**Naive 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

**Naive 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*

Image source: Brightsight
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
  • Relies heavily on white-box crypto
  • Traditional (DRM) and new use-cases HCE (payment, transit, …)

• DCA is an automated attack → no expertise needed
  • Counterpart of the DPA from the crypto HW community

• Level of security / maturity of many (all?) WB schemes is questionable
  • Open problem to construct asymmetric WB crypto
  • Industry keeps design secret
  • Need for way to measure the real security of such software solutions

• We will probably see more advanced countermeasures and attacks soon

What is the real security level of the deployed HCE solutions?