Practical white-box topics
design and attacks – part 1

Joppe W. Bos
White-Box Cryptography and Obfuscation
August 14, 2016, Santa-Barbara, California, USA
What to White-Box?

**Standardized crypto**
- Comply with current standards / protocols required
  - Focus is on AES / DES

**“New” crypto**
- Crypto designed to aid certain WB properties
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 to white-box standardized crypto.
Recall: White box model

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
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 of standard crypto broken)
- Convert algorithms to sequence of LUTs
- Embed the secret key in the LUTs
- Obfuscate the LUTs by using encodings
AES with look-up tables: example, Chow

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

- 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 → 32 bit table → 1024 byte) operations:
  \[ c_j = M_0T_{0,j}(a_{0,j}) \oplus M_1T_{1,j}(a_{1,j}) \oplus M_2T_{2,j}(a_{2,j}) \oplus M_3T_{3,j}(a_{3,j}) \]

- We can now implement a round by only using the following 2 types of lookup tables:

![Diagram of AES round with lookup tables](image_url)
AES (Chow) 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 (Chow) 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
AES (Chow) 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.
Obfuscation, obfuscation, obfuscation

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

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

\[
\begin{array}{c}
A_i \xrightarrow{8} T_{i,j} \xrightarrow{8} MB \cdot M_i \xrightarrow{8x4} \\
4 \quad 4 \quad 4 \quad 4 \quad 4
\end{array}
\]

\[
\begin{array}{c}
MB_i^{-1} \xrightarrow{8x4} \\
4 \quad 4 \quad 4 \quad 4 \quad 4
\end{array}
\]

\[
\begin{array}{c}
A_0 \xrightarrow{8} A_1 \xrightarrow{8} A_2 \xrightarrow{8} A_3 \xrightarrow{8} \\
4 \quad 4 \quad 4 \quad 4 \quad 4
\end{array}
\]

\[
\begin{array}{c}
(f_{0,i}, f_{1,i})A_i^{-1} \cdot c_{i,j}
\end{array}
\]

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: create software traces 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 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: $\text{SubBytes}(p \oplus \kappa)$

![Diagram showing AES algorithm stages]

- Key Expansion
- AddRoundKey
- MixColumns
- AddRoundKey
- ShiftRows
- AddRoundKey
- SubBytes
- ShiftRows
- SubBytes
- R?
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
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 analogy: this is like probing each bus-line individually *without any error*
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.

Countermeasures?

**Academic remedies**

- Cannot rely on random data in the white-box attack model
- Use static random data within the white-box itself?
- DCA might fail when using large encodings → either impractically large tables or simplified schemes → easy to break with algebraic attacks
- Use ideas from threshold implementation?
  - masking scheme based on secret sharing and multi-party computation

**Practical remedy**

- strengthen other measures
  - anti-debug / detect DBI frameworks, code-obfuscation (?), integrity checks, platform binding, etc
Any help to complete our collection of open whitebox challenges and attacks or to improve our tools is highly appreciated!

https://github.com/SideChannelMarvels
Conclusions and future work

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

- Besides traditional (DRM) also other use-cases (HCE) such as 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 (no expertise needed!)
  - Counterpart of the SCA from the crypto HW community

- What if DCA fails, can we do better? What about software FA, CPA, higher-order attacks etc?
  - See the next presentation!
    Riscure was the first show DFA works as well, see our online repo for an implementation

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