White-Box and Asymmetrically Hard Crypto Design

Alex Biryukov
University of Luxembourg

18-May-2019

slides from Whibox’19 workshop
Plan of the talk

• The ASASA story
• Resource Hardness Framework
• Other ideas
Structural cryptanalysis of SASAS*

- Scheme with unknown keyed S-boxes and Affine mappings
- For 128-bit block, 8-bit S-boxes, secret key-size is $2^{17}$ bits

*Biryukov, Shamir, Structural Cryptanalysis of SASAS, Eurocrypt’2001
Structural cryptanalysis of SASAS*

- For 128-bit block, 8-bit S-boxes, secret key-size is $2^{17}$ bits
- **Multiset attack** complexity is $2^{16}$ chosen texts and $2^{28}$ time

*Biryukov, Shamir, Structural Cryptanalysis of SASAS, Eurocrypt’2001*
Structural cryptanalysis of SASAS

• What this has to do with WBC?
Structural cryptanalysis of SASAS

• Many early obfuscations were broken because SASAS and shorter ciphers are structurally very weak (and simple ASA was used in many WBC schemes)

• Strong diffusion in ciphers prevents from building tables with more rounds since lookup tables explode
The ASASA attempt*

*One scheme we couldn’t break in 2001 was ASASA (with bijective S-boxes)
* (ASASA with non-bij. S-boxes was proposed as PK scheme by PatarinGoubin’97 and broken by Ding-Feng’99, Biham’00)

*Biryukov, Bouillaguet,Khovratovich, Cryptographic Schemes based on ASASA.., AC’2014
The ASASA attempt*

- Defined strong and weak white box crypto in [BBK’14] a la [Wyseur’09] (Strong WBC=PK, i.e. no ability to decrypt, was the main goal of the paper, also now called one-wayness (OW))
- Built strong and weak WBC from ASASA
- Strong WBC was based on multivariate crypto, expanding S-boxes+noise

*Fig. 2. Small perturbations to defeat decomposition attacks as injection of sparse high-degree polynomials

*Biryukov, Bouillaguet,Khovratovich, Cryptographic Schemes based on ASASA.., AC’2014
The ASASA attempt*

- Built strong and weak WBC from ASASA
- Strong WBC was based on multivariate crypto, expanding S-boxes+noise
- Strong and some weak WBC broken in 3 nice cryptanalytic papers [GPT’15, DDKL’15, MDFK’15]

*Biryukov, Bouillaguet, Khovratovich, Cryptographic Schemes based on ASASA.., AC’2014
The ASASA attempt

A few more details on our weak WBC scheme

- SPN, recursive approach, assuming ASASA or ASASASA mini-ciphers are secure against decomposition
The ASASA attempt

- ASASASA instances still unbroken
- Overall approach is valid, just needs more rounds $r$, description size grows linearly with $r$. 
The ASASA attempt

- ASASA instances still unbroken
- Overall approach is valid, just needs more rounds.
- Motivated more research on weak-WBC and nice constructions SPACE [BI15], PuppyCipher [FKKM16], SPNBox [BIT16]
Weak white-box

• "We note that a white-box implementation can be useful as it forces the user to use the software at hand“, -Marc Joye’08
Weak white-box

- Incompressibility ≈ Space-hardness ≈ Code-hardness
- Generalize: Resource $R$-hardness

Force to use implementation with special properties:
- *Inefficient* in resource $R$
- Password-protected (access control)
- Tagged/watermarked (tracing)
Resource Hardness Framework*

Efficiency metrics for crypto algorithms:
• Speed (Time complexity, parallel or sequential)
• Code-size (ROM)
• Memory complexity (RAM)
Sometimes *inefficiency* of algorithms in these metrics is required

*Biryukov, Perrin, “Symmetrically and Asymmetrically Hard Cryptography, Asiacrypt’17*
Resource Hardness Framework

Sometimes inefficiency of crypto algorithms in these metrics is required (*several research areas that do not always talk to each other*)

- Weak whitebox-crypto (code size hardness)
- Password hashing (memory hardness)
- Key derivation functions (KDF) (time hardness)
- Big key encryption (code size hardness)
- Time-lock puzzles, PoSW, VDFs (sequential time hardness)
- Proof-of-X (all kinds of hardness)
Resource Hardness Framework

Symmetric vs Asymmetric Resource hardness:

• Symmetric – computation is $R$ hard for all the users

• Asymmetric – computation is easy for “privileged” users knowing the secret $K$
### Resource Hardness Framework

<table>
<thead>
<tr>
<th>Applications</th>
<th>Time</th>
<th>Memory</th>
<th>Code size</th>
</tr>
</thead>
<tbody>
<tr>
<td>Symmetrically hard functions</td>
<td>PBKDF2 [Kal00]</td>
<td>Argon2 [BDK16], Balloon [BCGS16]</td>
<td>XKEY2 [BKR16], WHALE (Sec. 5.2)</td>
</tr>
<tr>
<td>Asymmetrically hard functions</td>
<td>RSA-lock [RSW96], SKIPPER (Sec. 5.1)</td>
<td>DIODON (Sec. 2.4.3)</td>
<td>White-box block ciphers [BBK14, BI15, FKKM16]</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>[BIT16]</td>
</tr>
</tbody>
</table>

Table 1: Six types of hardness and their applications.
Resource Hardness Framework

**Definition 2** (R-hardness). We say that a function $f : \mathcal{X} \to \mathcal{Y}$ is R-hard against $2^p$-adversaries for some tuple $R = (\rho, u, \epsilon(p))$ with $\rho \in \{\text{Time, Code, RAM}\}$ if evaluating the function $f$ using less than $u$ units of the resource $\rho$ and at most $2^p$ units of storage is possible only with probability $\epsilon(p)$. More formally, the probability for a $2^p$-adversary to win the efficient approximation game, which is described below, must be upper-bounded by $\epsilon(p)$.

1. The challenger chooses a function $f$ from a predefined set of functions requiring more than $u$ units of $\rho$ to be evaluated.

2. The challenger sends $f$ to the adversary.

3. The adversary computes an approximation $f'$ of $f$ which, unlike $f$, can be computed using less than $u$ units of the resource $\rho$.

4. The challenger picks an input $x$ of $\mathcal{X}$ uniformly at random and sends it to the adversary.

5. The adversary wins if $f'(x) = f(x)$.

*Generalized from definition of incompressibility from [FKKM16]*
Resource Hardness Framework

Figure 2: The game corresponding to the definition of \((\rho, u, \epsilon(p))\)-hardness against \(2^p\)-adversaries.
Resource Hardness Framework

- How to achieve required $R$-hardness?
- The framework allows us to construct primitives with any hardness type: the idea of *plugs* with specific hardness type
Plugs: Time-Hardness

Symmetric:
• **IterHash** \((t,n)\) — iterates \(t\)-bit hash \(n\) times \((n < 2^{t/2}\) to avoid cycles\)

Asymmetric
• **RSAlock**\((t,n)\) (time-lock) \(n\) squarings mod \(N\), \(N=\text{pq} \approx 2^t\)

\[
\text{RSAlock}_n^t(x) = x^{2^n} \mod N
\]

Secret owner first computes \(e=2^n \mod (p-1)(q-1)\)
Then he computes \(x^e \mod N\) (or CRT)
Plugs: Code-Hardness

Symmetric:
• BigLUT \((t,v)\) – a table with \(2^t\) random \(v\)-bit entries

Asymmetric
• \(\text{BcCounter}(t,v) = E_k(0^{v-t} \| x)\), \(E_k\) is a \(v\)-bit block cipher with secret key \(k\), \(|k| \geq v\)
  \(\)Secret owner knows \(k\)

Hardness for the common user:
\[(\text{Code}, 2^p, 2^{p-t})\)-hard
Plugs: Code-Hardness

Symmetric:
• BigLUT \((t,v)\) – a table with \(2^t\) random \(v\)-bit entries

Asymmetric
• \(\text{BcCounter}(t,v) = E_k(0^{v-t} \| x)\), \(E_k\) is a \(v\)-bit block cipher with secret key \(k\), \(|k| \geq v\), \(|x| = t\), \(t < v\)
  Secret owner knows \(k\)

Improvement for small \(t\): (parallel application of \(l\) tables \(|x| = v\))

\[
f(x_0 || \ldots || x_{\ell-1}) = \bigoplus_{i=0}^{\ell-1} E_k(\text{byte}(i) || 0^{n-t-8} \| x_i)
\]

Hardness for the common user:
\[
(Code, 2^p, \max(2^{p-v}, (2^{p-t} / \ell)^\ell))\text{-hard}.
\]
Plugs: Memory-Hardness

Symmetric:
- \texttt{Argon2(t,M)} with input size $t$ and memory size $M$ (memory hard password hashing function)

Asymmetric:
- \texttt{Diodon} (more details later)
Our collection of $R$-hard plugs

<table>
<thead>
<tr>
<th>Hardness</th>
<th>Symmetric</th>
<th>Asymmetric</th>
</tr>
</thead>
<tbody>
<tr>
<td>Time</td>
<td>$\text{IterHash}^t_\eta$ (Time, $\eta$, $2^{p-t}$)</td>
<td>$\text{RSAlock}^t_\eta$ (Time, $\eta$, $2^{p-t}$)</td>
</tr>
<tr>
<td>Memory</td>
<td>$\text{Argon2}$ (RAM, $M/5$, $2^{p-t}$)</td>
<td>$\text{DIODON}$ (RAM, $M/10$, $2^{p-t}$)</td>
</tr>
<tr>
<td>Code</td>
<td>$\text{BigLUT}^t_v$ (Code, $2^p$, $2^{p-t}$)</td>
<td>$\text{BcCounter}^t_v$ (Code, $2^p$, $2^{p-t}$)</td>
</tr>
</tbody>
</table>

Table 2: Possible plugs, i.e. sub-components for our constructions which we assume to be $R$-hard against $2^p$-adversaries.
Modes of Plug Usage

The plugs can be used in different modes

- Plug-then-randomize (PTR)
- Hard block cipher mode (HBC)
- Hard sponge mode (HSp)
Mode: Plug-then-Randomize

Figure 3: Evaluating the plugged function $(F \cdot P)$

Here $F$ is a random (permutation) oracle

Iterate to increase hardness:

$(\rho, u, \max(\epsilon(p)^r, 2^{p-n}))$-hard against $2^p$-adversaries
Mode: Hard block cipher

Figure 4: The HBC block cipher mode.

- Given related-key-secure $n$-bit block cipher $E_k$, $k \geq n$

$$(\rho, u, \max(\epsilon(p)^r, 2^{p-n}))$$-hard against $2^p$-adversaries
Example: Time-hard block cipher \textit{Skipper}

\begin{algorithm}
\textbf{Algorithm 5} \textsc{Skipper} encryption
\begin{itemize}
\item \textbf{Inputs}: \(n\)-bit plaintext \(x\); \(k\)-bit key \(k\); RSA modulus \(N\)
\item \textbf{Output}: \(n\)-bit ciphertext \(y\)
\end{itemize}
\begin{verbatim}
\begin{align*}
y & \leftarrow \text{AES}_k(x) \\
\text{for all } i \in \{1, 2\} \text{ do} \\
\quad & y_1 \ || \ y_2 \leftarrow y, \text{ where } |y_1| = 88 \text{ and } |y_2| = 40 \\
\quad & y_2 \leftarrow y_2 \oplus T_{40}(y_1^{2^n} \mod N) \\
\quad & y \leftarrow \text{AES}_{k \oplus i}(y_1 \ || \ y_2) \\
\text{end for} \\
\text{return } y
\end{align*}
\end{verbatim}
\end{algorithm}

- The plug is: \((\text{Time}, \eta, 2^{-40})\)-hard \textit{Skipper} is:
  \((\text{Time}, \eta, \max (2^{48-128}, (2^{-40})^2))\)-hard
Hard Sponge Mode (HSp)

- Sponges can be used to construct hash functions, stream ciphers, MACs and AE

Figure 5: A sponge-based hash function.
Hard Sponge Mode (HSp)

- Iteratively use Plug-then-Randomize mode

In the paper: Code-hard hash function based on Keccak which we called Whale.
Example: Memory-Hard function \textit{Diodon}

\begin{algorithm}
\caption{DIODON Asymmetrically memory-hard function}
\begin{algorithmic}
\Input \text{t}-bit block $x$; RSA modulus $N$ of $n_p$ bits; $M, L$;
\Output \text{u}-bit output $y$
\State $V_0 = x$
\ForAll {$i \in \{1, \ldots, M - 1\}$}
\State $V_i = V_{i-1}^{2^n} \mod N$
\EndFor
\State $S = V_{M-1}$
\ForAll {$i \in \{0, \ldots, L - 1\}$}
\State $j = S \mod M$
\State $S = H(S, V_j)$
\EndFor
\Return $T_u(S)$
\end{algorithmic}
\end{algorithm}
Example: Memory-Hard function \textit{Diodon}

Algorithm 2  \textsc{Diodon} for privileged users
\textit{Inputs:} $t$-bit block $x$; RSA factors $q, q'$; $\eta$; $M, T$
\textit{Output:} $u$-bit output $y$

\begin{align*}
e &= 2^{(M-1)\times\eta} \mod (q - 1)(q' - 1) \\
S &= x^e \mod (qq') \\
&\text{for all } i \in \{0, \ldots, L - 1\} \text{ do} \\
j &= S \mod M \\
e_j &= 2^{j\times\eta} \mod (q - 1)(q' - 1) \\
S &= H\left(S, (x^{e_j} \mod (qq'))\right) \\
&\text{end for} \\
&\text{return } T_u(S)
\end{align*}
### Resource hardness Framework

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Conservative</th>
<th>Fast</th>
</tr>
</thead>
<tbody>
<tr>
<td>$t$</td>
<td>128</td>
<td>128</td>
</tr>
<tr>
<td>$u$</td>
<td>128</td>
<td>128</td>
</tr>
<tr>
<td>$n_p$</td>
<td>2048</td>
<td>1024</td>
</tr>
<tr>
<td>$\eta$</td>
<td>2048</td>
<td>1</td>
</tr>
<tr>
<td>$M$</td>
<td>4,000</td>
<td>8,000,000</td>
</tr>
<tr>
<td>$L$</td>
<td>4,000</td>
<td>20,000</td>
</tr>
</tbody>
</table>

- $n_p$ – bits in RSA modulus; $t,u$ – input/output sizes; $M,L$– upper/lower chain length
- RAM (basic user) 1 Mb 1 Gb
- Time (basic user) 10.00 s 9.87 s
- Time (privileged) 13.49 s 10.65 s
## Resource hardness Framework

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Conservative</th>
<th>Fast</th>
</tr>
</thead>
<tbody>
<tr>
<td>( t )</td>
<td>128</td>
<td>128</td>
</tr>
<tr>
<td>( u )</td>
<td>128</td>
<td>128</td>
</tr>
<tr>
<td>( n_p )</td>
<td>2048</td>
<td>1024</td>
</tr>
<tr>
<td>( \eta )</td>
<td>2048</td>
<td>1</td>
</tr>
<tr>
<td>( M )</td>
<td>4,000</td>
<td>8,000,000</td>
</tr>
<tr>
<td>( L )</td>
<td>4,000</td>
<td>20,000</td>
</tr>
<tr>
<td>RAM (basic user)</td>
<td>1 Mb</td>
<td>1 Gb</td>
</tr>
<tr>
<td>Time (basic user)</td>
<td>10.00 s</td>
<td>9.87 s</td>
</tr>
<tr>
<td>Time (privileged)</td>
<td>13.49 s</td>
<td>10.65 s</td>
</tr>
</tbody>
</table>

### Open problem: Diodon is based on scrypt which has lousy linear TM-tradeoff. Also slow due to RSA. Improve?
Few other things
R-hardness and code obfuscation

Using obfuscation idea from [BK’16*]:

• Compiler that runs some resource hard function $F(pwd,x)$
• Computes $R$-hard bits $F(pwd,x) = b_i$ and then makes code transformations:

* Biryukov, Khovratovich, Egalitarian Computing, Usenix’16
**R-hardness and code obfuscation**

Using obfuscation idea from [BK’16]:

- Compiler that runs some resource hard function \( F(pwd, x) \)
- Computes \( R \)-hard bits \( F(pwd, x) = b_i \) and then makes code transformations:

\[
\begin{align*}
\text{for } b_i &= 0 \\
\text{if } x \oplus b_i &\quad \text{then } A \\
\text{else } B \\
\text{for } b_i &= 1 \\
\text{if } x \oplus b_i &\quad \text{then } B \\
\text{else } A
\end{align*}
\]

- The user will have to run \( R \)-hard function \( F(pwd, x) \) at least once
**R-hardness and code obfuscation**

Using obfuscation idea from [BK’16]:

- Compiler that runs some resource hard function \( F(pwd,x) \)
- Computes \( R \)-hard bits \( F(pwd,x) = b_i \) and then makes code transformations:

  - This could work well for previously unseen code.
R-hardness and code obfuscation

Using obfuscation idea from [BK’16]:

• Compiler that runs some resource hard function $F(pwd, x)$
• Computes $R$-hard bits $F(pwd, x) = b_i$ and then makes code transformation:

Would this approach work to make Incompressible, password protected INC-AES?
R-hardness and code obfuscation

• Not really. Unless we already have $K$-unextractable/unbreakable UBK-AES.
• However it shows hope that at least in some cases UBK $\Rightarrow$ INC
Related topics

Related research topics

• Code Obfuscation (for structure hiding)
• Cross-pollination with GreyBox crypto (for value hiding)
• IO
• Malicious crypto – adversarial crypto design
• PK crypto based on new ideas
Open problems

• Can we design a WBC-friendly cipher?
• Would Even-Mansour cipher be a good candidate?
• Design Diodon-like asymmetric memory hard functions with non-linear TM tradeoffs and faster operations
• INC-PWD-AES?
End

(and we are hiring postdocs on WBC and other topics)

cryptolux.org