Joltik and Deoxys

Jérémy Jean  Ivica Nikolić  Thomas Peyrin

Nanyang Technological University, Singapore

DIAC 2014 – August 23, 2014

http://www1.spms.ntu.edu.sg/~syllab/Joltik
http://www1.spms.ntu.edu.sg/~syllab/Deoxys
Introduction

- Presentation of Joltik and Deoxys candidates.

- Together with Kiasu, they are different instances of the new TWEAKEY framework that we propose.

- Joltik and Deoxys share the same structure inside this framework.

- They use tweakable block ciphers (as Kiasu).

- Joltik: lightweight and hardware-oriented.

- Deoxys: fast and software-oriented (AES-NI).
Tweakable block ciphers for AEAD

Previous work on TBC:

▶ Several known methods for TBC, e.g.: LRW, XEX.
▶ Drawback: birthday-bound security.

(new) The **TWEAKEY framework:** *to appear at ASIACRYPT 2014*

▶ Unified approach to handle keys and tweaks.
▶ Standalone primitive to achieve a TBC.
▶ Tweak and key processed (almost) the same way.
▶ Only a framework $\implies$ unsecured instances exist.
▶ **Security reduction:** regular block cipher with new key schedule.
▶ Particular subclass: Superposition-**TWEAKEY (STK).** $\implies$ Precise the tweakey schedule.
**TWEAKEY framework**

TWEAKEY generalizes the class of key-alternating (KA) cipher.

\[
P = s_0 \oplus f_{s_1} \oplus \cdots \oplus f_{s_r} = s_{r+1} = C
\]

- The regular key schedule is replaced by a TWEAKEY schedule.
- An \( n \)-bit key \( n \)-bit tweak TBC have 2\( n \)-bit tweakey and \( g \) compresses 2\( n \) to \( n \) bits.
- Such a primitive would be a TK-2 primitive (TWEAKEY of order 2).
- The same primitive can be seen as a 2\( n \)-bit key cipher with no tweak (or 1.5\( n \)-bit key 0.5\( n \)-bit tweak, etc).
Towards the STK construction (Superposition-TWEAKEY)

\[ P = s_0 \oplus f \rightarrow s_1 \oplus f \rightarrow \cdots \rightarrow s_r \oplus f \rightarrow s_{r+1} = C \]

**Simplifications**

- We would like to process the key and tweak inputs **independently** in the TWEAKEY schedule \( h \) and in the **same way**.

- The subtweakey addition of \( g(tk_i) \) consists in XORing all the \( n \)-bit words of the tweakey state into the internal state.

- This would:
  - reduce the implementation overhead,
  - reduce the area footprint by reusing code,
  - simplify the security analysis.

- **But**: possible interactions between the XOR of \( n \)-bit tweakey words.
The STK construction

**STK Key Schedule (TK-\(p\))**

- We consider \(c\)-bit nibbles in each (say \(p\)) \(n\)-bit tweakey words.
- The \(h\) function is replaced by \(n\) independent applications of a \(h'\) function, which is a nibble-wise substitution.
- To reduce the interaction of the tweakey words at the output of the \(g\) function, each nibble of the \(k\)-th tweakey word is multiplied by a value \(\alpha_k \in GF(2^c)\).
The STK construction

STK Key Schedule (TK-\(p\))

We consider \(c\)-bit nibbles in each (say \(p\)) \(n\)-bit tweakey words.  

The \(h\) function is replaced by \(n\) independent applications of a \(h'\) function, which is a nibble-wise substitution.

To reduce the interaction of the tweakey words at the output of the \(g\) function, each nibble of the \(k\)-th tweakey word is multiplied by a value \(\alpha_k \in GF(2^c)\).
The STK construction

STK Key Schedule (TK-\(p\))

\[
\begin{align*}
\text{tk}_0 &\rightarrow h' &\rightarrow \alpha_1 &\rightarrow \text{XOR} \rightarrow C_0 \\
\text{tk}_0 &\rightarrow h' &\rightarrow \alpha_2 &\rightarrow \text{XOR} \rightarrow C_1 \\
\text{tk}_0 &\rightarrow h' &\rightarrow \alpha_1 &\rightarrow \text{XOR} \rightarrow C_2 \\
\text{tk}_0 &\rightarrow h' &\rightarrow \alpha_2 &\rightarrow \text{XOR} \rightarrow C_{r-1} \\
\end{align*}
\]

\[
\begin{align*}
P = s_0 &\rightarrow f &\rightarrow \oplus \text{ ART} \rightarrow f \\
P = s_0 &\rightarrow f &\rightarrow \oplus \text{ ART} \rightarrow f \\
P = s_0 &\rightarrow f &\rightarrow \oplus \text{ ART} \rightarrow f \\
P = s_0 &\rightarrow f &\rightarrow \oplus \text{ ART} \rightarrow s_r = C
\end{align*}
\]

STK

- We consider \(c\)-bit nibbles in each (say \(p\)) \(n\)-bit tweakey words.
- The \(h\) function is replaced by \(n\) independent applications of a \(h'\) function, which is a nibble-wise substitution.
- To reduce the interaction of the tweakey words at the output of the \(g\) function, each nibble of the \(k\)-th tweakey word is multiplied by a value \(\alpha_k \in GF(2^c)\).
The STK construction: rationale

Design choices:

► Multiplication in $GF(2^c)$ controls the number of cancellations at the output of $g$, when the subtweakeys are XORed to the internal state.

► Rely on a linear code to bound the number of cancellations.

Security analysis:

► Simplified security analysis in STK.

► Easy analysis of the tweakey schedule (hard for AES).

► Possibility to reuse previous works and several existing tools searching for high-probability differential characteristics (easy to introduce limitations of the number of cancellations of differences).

Implementation:

► Very simple transformations: linear and lightweight.

► Multiplications constants chosen as 1, 2, 4, ... for efficiency.
Joltik

Lightweight and hardware-oriented candidate to CAESAR.
Two family of ciphers: Joltik≠ and Joltik=.

Joltik≠ assumes nonce-respecting users:
- Rely on the $\ThetaCB3$ framework.
- Full security.
- Four recommended parameters (see submission).

Joltik= allows nonce-repeating users.
- Rely on the COPA mode.
- Birthday-bound security.
- Four recommended parameters (see submission).

Exactly the same modes as Kiasu (see previous presentation).
- Rely on the Joltik-BC tweakable block cipher.
Instance of the STK construction.

Two members: Joltik-BC-128 and Joltik-BC-192.

- 128 bits for TK-2: \(|key| + |tweak| = 128\) (2 tweakey words).
- 192 bits for TK-3: \(|key| + |tweak| = 192\) (3 tweakey words).

AES-based design.

Involutive MDS matrix in MixColumns \(\Rightarrow\) low decryption overhead.

S-Box from the Piccolo block cipher (compact in hardware).

Joltik-BC-128 has 24 rounds (TK-2).

Joltik-BC-192 has 32 rounds (TK-3).

TWEAKEY schedule:

- \(h'\) is a simple permutation of the 16 nibbles.
- Multiplications factor are: 1, 2 and 4 in \(GF(16)/0x13\).
- Constant additions to break symmetries (from LED cipher).
### Security claims of Joltik (bits of security, $\log_2$)

#### Nonce-respecting user

<table>
<thead>
<tr>
<th></th>
<th>Joltik ≠</th>
<th>Joltik =</th>
</tr>
</thead>
<tbody>
<tr>
<td>Confidentiality for the plaintext</td>
<td>$k$</td>
<td>$n/2$</td>
</tr>
<tr>
<td>Integrity for the plaintext</td>
<td>$n$</td>
<td>$n/2$</td>
</tr>
<tr>
<td>Integrity for the associated data</td>
<td>$n$</td>
<td>$n/2$</td>
</tr>
</tbody>
</table>

#### Nonce-repeating user

<table>
<thead>
<tr>
<th></th>
<th>Joltik ≠</th>
<th>Joltik =</th>
</tr>
</thead>
<tbody>
<tr>
<td>Confidentiality for the plaintext</td>
<td>none</td>
<td>$n/2$</td>
</tr>
<tr>
<td>Integrity for the plaintext</td>
<td>none</td>
<td>$n/2$</td>
</tr>
<tr>
<td>Integrity for the associated data</td>
<td>none</td>
<td>$n/2$</td>
</tr>
</tbody>
</table>
### Conjectured security of Joltik (bits of security, $\log_2$)

#### Nonce-respecting user

<table>
<thead>
<tr>
<th>Feature</th>
<th>Joltik $\neq$</th>
<th>Joltik $=$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Confidentiality for the plaintext</td>
<td>$k$</td>
<td>$n$</td>
</tr>
<tr>
<td>Integrity for the plaintext</td>
<td>$n$</td>
<td>$n$</td>
</tr>
<tr>
<td>Integrity for the associated data</td>
<td>$n$</td>
<td>$n$</td>
</tr>
</tbody>
</table>

#### Nonce-repeating user

<table>
<thead>
<tr>
<th>Feature</th>
<th>Joltik $\neq$</th>
<th>Joltik $=$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Confidentiality for the plaintext</td>
<td>none</td>
<td>$n/2$</td>
</tr>
<tr>
<td>Integrity for the plaintext</td>
<td>none</td>
<td>$n/2$</td>
</tr>
<tr>
<td>Integrity for the associated data</td>
<td>none</td>
<td>$n/2$</td>
</tr>
</tbody>
</table>
Implementations of Joltik≠

Software implementations

- vperm implementation (SSSE3 and avx2): about the same (expected) speed as LED.
- Projection for bitslice: about 9 cpb for 4KB messages.
- Similar numbers for other Joltik≠ parameters.
- Joltik= expected to be 2x slower.

Hardware implementations

- Estimations (see specs): (LED-128: about 1300GE)
  - 1500 GE for Joltik-BC-128 (TBC only),
  - 2000 GE for Joltik-BC-128 (TBC only),
  - 2100 GE for Joltik TK-2,
  - 2600 GE for Joltik TK-3.
- See estimations for Joltik= in the specs.
Deoxys

Fast and software-oriented candidate to CAESAR.
Also two family of ciphers:

- **Deoxys\(\neq\)** for nonce-respecting users,
- **Deoxys\(=\)** for nonce-repeating users.

Same modes as *Joltik* and *Kiasu*.

Two sets of recommended parameters for each mode.

Rely on the *Deoxys-BC* tweakable block cipher.
Deoxys-BC

- Also an instance of the STK construction.
  - 256 bits for TK-2: $|\text{key}| + |\text{tweak}| = 256$ (2 tweakey words).
  - 384 bits for TK-3: $|\text{key}| + |\text{tweak}| = 384$ (3 tweakey words).
- The round function is exactly the AES round function (AES-NI).
- Deoxys-BC-256 has 14 rounds (TK-2).
- Deoxys-BC-384 has 16 rounds (TK-3).
- TWEAKEY schedule:
  - $h'$ is the same permutation as Joltik.
  - Multiplications factor are: 1, 2 and 4 in the AES field.
  - Constant additions to break symmetries ($\text{RCON}$ from AES KS).
Security claims of **Deoxys** *(bits of security, $\log_2$)*

*Same as Joltik.*

### Nonce-respecting user

<table>
<thead>
<tr>
<th></th>
<th>$\text{Deoxys} \neq$</th>
<th>$\text{Deoxys} = $</th>
</tr>
</thead>
<tbody>
<tr>
<td>Confidentiality for the plaintext</td>
<td>$k$</td>
<td>$n/2$</td>
</tr>
<tr>
<td>Integrity for the plaintext</td>
<td>$n$</td>
<td>$n/2$</td>
</tr>
<tr>
<td>Integrity for the associated data</td>
<td>$n$</td>
<td>$n/2$</td>
</tr>
</tbody>
</table>

### Nonce-repeating user

<table>
<thead>
<tr>
<th></th>
<th>$\text{Deoxys} \neq$</th>
<th>$\text{Deoxys} = $</th>
</tr>
</thead>
<tbody>
<tr>
<td>Confidentiality for the plaintext</td>
<td>none</td>
<td>$n/2$</td>
</tr>
<tr>
<td>Integrity for the plaintext</td>
<td>none</td>
<td>$n/2$</td>
</tr>
<tr>
<td>Integrity for the associated data</td>
<td>none</td>
<td>$n/2$</td>
</tr>
</tbody>
</table>
Conjectured security of $\text{Deoxys}$ (bits of security, $\log_2$)

**Same as** $\text{Joltik}$.

### Nonce-respecting user

<table>
<thead>
<tr>
<th></th>
<th>$\text{Deoxys} \neq$</th>
<th>$\text{Deoxys} = $</th>
</tr>
</thead>
<tbody>
<tr>
<td>Confidentiality for the plaintext</td>
<td>$k$</td>
<td>$n$</td>
</tr>
<tr>
<td>Integrity for the plaintext</td>
<td>$n$</td>
<td>$n$</td>
</tr>
<tr>
<td>Integrity for the associated data</td>
<td>$n$</td>
<td>$n$</td>
</tr>
</tbody>
</table>

### Nonce-repeating user

<table>
<thead>
<tr>
<th></th>
<th>$\text{Deoxys} \neq$</th>
<th>$\text{Deoxys} = $</th>
</tr>
</thead>
<tbody>
<tr>
<td>Confidentiality for the plaintext</td>
<td>none</td>
<td>$n/2$</td>
</tr>
<tr>
<td>Integrity for the plaintext</td>
<td>none</td>
<td>$n/2$</td>
</tr>
<tr>
<td>Integrity for the associated data</td>
<td>none</td>
<td>$n/2$</td>
</tr>
</tbody>
</table>
Performances of Deoxys using AES-NI.

### Benchmark of Deoxys≠ with 128-bit key 128-bit tweak (in cpb).

<table>
<thead>
<tr>
<th>Size</th>
<th>Intel Haswell</th>
<th>Intel Sandy Bridge</th>
</tr>
</thead>
<tbody>
<tr>
<td>1KB</td>
<td>2.12</td>
<td>2.37</td>
</tr>
<tr>
<td>2KB</td>
<td>1.74</td>
<td>1.85</td>
</tr>
<tr>
<td>4KB</td>
<td>1.55</td>
<td>1.59</td>
</tr>
<tr>
<td>8KB</td>
<td>1.46</td>
<td>1.43</td>
</tr>
<tr>
<td>64KB</td>
<td>1.38</td>
<td>1.31</td>
</tr>
</tbody>
</table>

### Benchmark of Deoxys= with 128-bit key 128-bit tweak (in cpb).

<table>
<thead>
<tr>
<th>Size</th>
<th>Intel Haswell</th>
<th>Intel Sandy Bridge</th>
</tr>
</thead>
<tbody>
<tr>
<td>1KB</td>
<td>3.75</td>
<td>4.74</td>
</tr>
<tr>
<td>2KB</td>
<td>3.13</td>
<td>3.91</td>
</tr>
<tr>
<td>4KB</td>
<td>2.84</td>
<td>3.44</td>
</tr>
<tr>
<td>8KB</td>
<td>2.69</td>
<td>3.11</td>
</tr>
<tr>
<td>64KB</td>
<td>2.56</td>
<td>2.80</td>
</tr>
</tbody>
</table>

Notes:

- Benchmarks done in the $K\Delta N\Delta$ model.
- Fast non AES-NI implementations coming soon.
- Twice more TBC calls in Deoxys= to achieve nonce-misuse resistance.
Performances of Deoxys using AES-NI.

<table>
<thead>
<tr>
<th>TWEAKEY</th>
<th>Joltik</th>
<th>Deoxys</th>
</tr>
</thead>
</table>
| Source: | http://www1.spms.ntu.edu.sg/~syllab/speed/.

Deoxys in the top 10% of AES-NI implementations on SUPERCOP.
Security analysis

- We have scrutinized the security of the TWEAKEY framework, and devised the STK subclass.
  → Provide bounds on the number of differences introduces by the tweakey schedule.

- This bound can easily be used in existing differential characteristic search tools.

- We conducted a differential analysis, and selected the number of rounds such that:
  - Joltik-BC has 8 rounds of security margin,
  - Deoxys-BC has 4 rounds of security margin.

- Also in the submission documents: analysis against MITM strategy.
Conclusion

► We propose the TWEAKEY framework to design easy-to-analyze tweakable block ciphers (more in an upcoming ASIACRYPT 2014 paper).

► We instantiate this framework to get two TBC:
  ▶ Joltik-BC, which is lightweight and hardware-oriented,
  ▶ Deoxys-BC, which is fast and software-oriented.

► We plug these two ciphers into two different modes to achieve AEAD schemes:
  ▶ one mode similar to OCB3 for nonce-respecting users,
  ▶ one mode similar to COPA to achieve nonce-misuse resistance.
We propose the **TWEAKEY framework** to design easy-to-analyze tweakable block ciphers (more in an upcoming ASIACRYPT 2014 paper).

We instantiate this framework to get two TBC:

- Joltik-BC, which is lightweight and hardware-oriented,
- Deoxys-BC, which is fast and software-oriented.

We plug these two ciphers into **two different modes** to achieve AEAD schemes:

- one mode similar to **OCB3** for nonce-respecting users,
- one mode similar to **COPA** to achieve nonce-misuse resistance.

Thank you!