Tweakable Block Cipher Based Cryptography

Thomas Peyrin

NTU - Singapore

ICISC 2020

Virtual - November 02, 2020
Collaborators

Based on works in collaboration with:

B. Cogliati
T. Iwata
J. Jean
M. Khairallah
S. Kölbl
G. Leander
K. Minematsu

A. Moradi
I. Nikolic
Y. Sasaki
P. Sasdrich
Y. Seurin
S.M. Sim
H. Wang
# Introduction

# Tweakable Block Ciphers Designs

- Block Cipher-Based TBC
- Ad-hoc TBC Constructions

# Tweakable Block Ciphers for AE

# Conclusion
1 Introduction

2 Tweakable Block Ciphers Designs
   ▶ Block Cipher-Based TBC
   ▶ Ad-hoc TBC Constructions

3 Tweakable Block Ciphers for AE

4 Conclusion
A **block cipher** (BC) is a family of permutations parametrized by a secret key $K$.

![Diagram of block cipher](image)

A **tweakable block cipher** (TBC) is a family of permutations parametrized by a secret key $K$ and a public **tweak value** $T$.

![Diagram of tweakable block cipher](image)

We denote:

- $P$ the $n$-bit plaintext
- $C$ the $n$-bit ciphertext
- $K$ the $k$-bit key
- $T$ the $t$-bit tweak
(Tweakable) Block Ciphers

A **block cipher** (BC) is a family of permutations parametrized by a secret key $K$

A **tweakable block cipher** (TBC) is a family of permutations parametrized by a secret key $K$ and a public **tweak value** $T$

A **permutation** on $b = c + r$ bits, where $c$ is the capacity and $r$ is the rate (sponge framework \[BDPV-07\])
Some history: first tweakable block ciphers

Hasty Pudding Cipher from Schroeppel [Schroeppel-99]

- AES competition candidate
- introduces a 512-bit "spice" as a "secondary key, maybe completely or partially concealed, or completely open" and notes that "the spice can be changed very cheaply for each block encrypted". It is "expected to be changed often, perhaps for every encrypted block (allows the primary key to have a long lifetime)"
- spice material is added to the cipher internal state every round
- no claim against "chosen spice attack"
TBC History: Mercy

Some history: first tweakable block ciphers

Mercy cipher from Crowley [Cro-FSE00]

- includes a 128-bit randomizer or “spice”
  (for disk sector encryption: sector number would be used as a tweak)
- “The spice goes through a spice-scheduling procedure, analogous with key scheduling
  [...] this forms six 128-bit round spices”
- claims about TBC security for encryption only
- broken [Flu-FSE01]
Liskov et al. [LRW-C02] introduce first formalisation of TBC:

- “we expect tweaks to be changed frequently, so a tweakable block cipher should have the property that changing the tweak should be efficient. [... ] And, for any tweakable block cipher, changing the tweak should be less costly than changing the key.”.

- “even if an adversary has control of the tweak input, we want the tweakable block cipher to remain secure”

- introduces the two first BC-based generic TBC constructions LRW1 and LRW2

- introduces new TBC-based modes, notably the hash function TCH (broken for certain instantiations [BCS-EC05]) and the AE mode TAE
Applications of TBCs

Some applications:

- many BC operating modes can be seen as TBC modes (using XEX construction). Ex: PMAC, OCB [Rog-AC04]
- XTS disk encryption mode = XEX + Ciphertext Stealing

Is that all?

No, TBCs are very interesting primitives to provide efficient, highly secure, simple (to understand and to prove) operating modes, for most classical symmetric-key security notions.

Standardization effort:

- Deoxys and SKINNY Committee Draft stage at ISO (18033-7)
1. Introduction

2. Tweakable Block Ciphers Designs
   - Block Cipher-Based TBC
   - Ad-hoc TBC Constructions

3. Tweakable Block Ciphers for AE

4. Conclusion
Outline

1. Introduction

2. Tweakable Block Ciphers Designs
   - Block Cipher-Based TBC
   - Ad-hoc TBC Constructions

3. Tweakable Block Ciphers for AE

4. Conclusion
Building a TBC from a BC

**A first (bad) idea**

Masking input/output with a tweak (DESX-like):

\[ \tilde{E}_K(T, P) = E_K(P \oplus T) \oplus T \]

→ results in an undesirable property

\[ \tilde{E}_K(T, P) \oplus \tilde{E}_K(T \oplus \delta, P \oplus \delta) = \delta \]

**A second (bad) idea**

XORing a tweak into the key input:

\[ \tilde{E}_K(T, P) = E_{K\oplus T}(P) \]

→ results in an undesirable property

\[ \tilde{E}_K(T, P) = \tilde{E}_{K\oplus \delta}(T \oplus \delta, P) \]
BC-based TBC : LRW1 and LRW2

First BC-based constructions [LRW-C02], up to birthday bound, changing tweak hopefully cheaper than key : LRW1 and LRW2

LRW1

\[ \tilde{E}_K(T, P) = E_K(T \oplus E_K(P)) \]

CBC-MAC

LRW2

\[ \tilde{E}_{K,h}(T, P) = E_K(P \oplus h(T)) \oplus h(T) \]

h is \(\oplus\)-universal - part of the secret key
Block-cipher based TBC : \texttt{XE} and \texttt{XEX}

**XOR Encrypt (\texttt{XE}) - XOR Encrypt XOR (\texttt{XEX})** [Rog-AC04]

**Idea:** mask input/(output) with a key and tweak-dependant value, s.t. it is efficient if sequential tweaks $T = T' || i || j$ are used:

\[
\tilde{E}_K(T, P) = E_K(P \oplus \Delta) \oplus \Delta
\]

with $\Delta = 2^i \cdot 3^j \cdot L$ and $L = E_K(T')$

PRP/SPRP up to **birthday bound** only:

- collision on $P \oplus \Delta \rightarrow P \oplus P' = C \oplus C'$
- recover the secret $L$, generate forgeries

**Used in:**
- \texttt{XTS} disk encryption mode
- \texttt{PMAC}, \texttt{OCB}, about a third of all CAESAR candidates, ...
## More generic TBC constructions and advances

- more on XEX [CS-INS06] [Min-SAC06] [CS-IT08] [GJM+-EC16]
- birthday-bound TBC from a permutation:
  - TEM [STA+-14] [CLS-C15] [CS-AC15] (XEX with a permutation)
  - MEM [GJM+-EC16] (TEM with more efficient masking)
  - XPX [Men-C16] (improved RK security guarantees)
- beyond birthday-bound TBC constructions from BC
  - [Min-FSE09] [LST-C12] [LS-FSE13] [Men-FSE15] [WGZ+-AC16]
  - [JLM+-LC17] [LL-AC18]
- XTX to extend tweak size [MI-IMA15]
- adding tweak in Luby-Rackoff ciphers [GHL+-AC07]
- building a larger BC out of a TBC (for BBB security)
  - [CDMS-TCC10] [Min-FSE09] [MI-IMA11] [Min-DCC15] [NI-FSE20]

Very active field, many improvements every year ...
Outline

1. Introduction

2. Tweakable Block Ciphers Designs
   - Block Cipher-Based TBC
   - Ad-hoc TBC Constructions

3. Tweakable Block Ciphers for AE

4. Conclusion
Why using ad-hoc TBC constructions?

to get beyond birthday-bound security with improved efficiency!

Theoretical / ad-hoc constructions are not opposed!

We can see a lot of inspiration from ad-hoc TBCs to BC or permutation based ones and vice-versa. A lot of interplay!
How to build an ad-hoc TBC?
The tweak schedule paradox: Tweak + Key = Tweakey

From [LRW-C02]:

- “we expect tweaks to be changed frequently, so a tweakable block cipher should have the property that changing the tweak should be efficient. [...] And, for any tweakable block cipher, changing the tweak should be less costly than changing the key.”
- “even if an adversary has control of the tweak input, we want the tweakable block cipher to remain secure”

Ad-hoc TBC designer’s perspective paradox:

- tweak schedule to be more efficient than the key schedule
- security requirements on the tweak seem somehow stronger than on the key: the attacker can fully control the former (even though tweak-recovery attacks are irrelevant)
The tweak schedule paradox: Tweak + Key = Tweakey

From [LRW-C02]:

- "we expect tweaks to be changed frequently, so a tweakable block cipher should have the property that changing the tweak should be efficient. [...] And, for any tweakable block cipher, changing the tweak should be less costly than changing the key."

- "even if an adversary has control of the tweak input, we want the tweakable block cipher to remain secure"

From a designer’s perspective, key and tweak should be considered as almost the same [JNP-AC14]:

Tweak + Key = Tweakey
The **TWEAKEY framework**

The **TWEAKEY framework rationale** [JNP-AC14]:
tweak and key should be treated the same way → **tweakey**

TWEAKEY generalizes the class of **key-alternating ciphers**
A bad idea:
XOR 128-bit tweak value $T$ to the internal state every round

$T = \begin{bmatrix}
T_0 & T_1 & T_2 & T_3 \\
T_4 & T_5 & T_6 & T_7 \\
T_8 & T_9 & T_{10} & T_{11} \\
T_{12} & T_{13} & T_{14} & T_{15}
\end{bmatrix}$
**How to not tweak **AES

**A bad idea:**
XOR 128-bit tweak value $T$ to the internal state every round

**Related-tweak diff. paths with only 1 active Sbox per round**
How to tweak AES: KIASU

KIASU [JNP-AC14]

Simply XORing 64-bit tweak $T$ in the two first rows of AES internal state at every round leads to no good related-tweak differential paths.
**How to tweak AES : KIASU**

**KIASU [JNP-AC14]**

Simply XORing 64-bit tweak $T$ in the two first rows of AES internal state at every round leads to no good related-tweak differential paths.
How to tweak AES: KIASU

KIASU [JNP-AC14]

Simply XORing 64-bit tweak $T$ in the two first rows of AES internal state at every round leads to no good related-tweak differential paths.

Interesting research topic:

▷ can an attacker leverage the freedom degrees from $T$?
▷ what about more complex attacks?
▷ so far 8 rounds can be attacked [DEM-ACNS16] [DL-CTRSA17]
Reusing existing long-key block ciphers

Idea: reuse existing long-key block ciphers

- what if we use a long-key block cipher and devote part of his key to be the tweak input? △ related-key attacks!
- Q: is AES-256 with 128-bit key and 128-bit tweak a secure TBC? Basically TAES proposal [BGIM-FSE20]
- A: not in TWEAKEY framework (RK attacks [BK-AC09])!
- TAES assumes single-key scenario only, while AES-256 RK attacks require differences in both K and T
Reusing existing long-key block ciphers

Idea: reuse existing long-key block ciphers

- what if we use a long-key block cipher and devote part of his key to be the tweak input? △ related-key attacks!
- Q: is AES-256 with 128-bit key and 128-bit tweak a secure TBC? Basically TAES proposal [BGIM-FSE20]
- A: not in TWEAKEY framework (RK attacks [BK-AC09])!
- TAES assumes single-key scenario only, while AES-256 RK attacks require differences in both $K$ and $T$

Interesting research topic:

Are there related-key differential paths for AES-256 with only one 128-bit word active, so as to attack TAES in the single key model?
Very short-tweak TBC

Elastic-Tweak construction for SPN ciphers [CDJ+19]
Very short-tweak TBC construction used in ESTATE and LOTUS–AEAD/LOCUS–AEAD of NIST LWC competition

Very short-tweak TBC Constructions

A very short tweak $t \ll n$ (like 4 or 8 bits) can be used:

- to simulate independent keys required by some operating modes: $E_{K_i}(P) \sim E_K(T_i, P)$
- for domain separation (full/partial block)
- not in TBC operating modes

Almost the same efficiency as the underlying BC, easy for designer because small tweak
How to build TBCs with large tweaks $t \gg n$?
How to build TBCs with large tweaks $t \gg n$?

Back to the good old problem of key schedule design
Tweakey scheduling design

Designing a tweakey scheduling is hard:

- many many ciphers got broken in the related-key model
- ... but we have a better understanding of how to build a (twea)key schedule since the SHA-3 competition
- simplicity is an important criterion to make the analysis feasible (lack of security analysis is not allowed)
- recently automated tools (SAT, MILP, CP) are really helpful to analyse diff/linear properties of a cipher

Problem:
When $t$ grows large, the SAT/MILP/CP problem instances becomes too large and the solvers can’t handle them anymore.
Tweakey scheduling design

Designing a tweakey scheduling is hard:

- many many ciphers got broken in the related-key model
- ... but we have a better understanding of how to build a (twea)key schedule since the SHA-3 competition
- simplicity is an important criterion to make the analysis feasible (lack of security analysis is not allowed)
- recently automated tools (SAT, MILP, CP) are really helpful to analyse diff/linear properties of a cipher

Problem:
When $t$ grows large, the SAT/MILP/CP problem instances becomes too large and the solvers can’t handle them anymore.

Solution:
Create a tweakey schedule that makes it easy for the solvers!
We can solve this problem using the Superposition Tweakey (STK) construction [JNP-AC14]:
The search problem for the tweak part is now reduced from a $t$-bit to a $n$-bit problem with a few extra cancellation conditions.
We can solve this problem using the Superposition Tweakey (STK) construction [JNP-AC14]:
The search problem for the tweak part is now reduced from a $t$-bit to a $n$-bit problem with a few extra cancellation conditions.

Now the goal is to find:
- cheap $\alpha_i$ transformations that minimize #cancellations
- best $h'$ to maximize resistance against related-tweakey attacks

Interesting research topic:
- finding the $\alpha_i$ to minimize cancellations when $t$ grows large
- maybe use an error correcting code on the tweak/key cells to generate all the successive subtweakeys?
Deoxys-TBC applies this STK idea to the AES [JNP-AC14]
Comparing Deoxys-TBC and AES

Deoxys-TBC applies this STK idea to the AES [JNP-AC14]

<table>
<thead>
<tr>
<th>Cipher</th>
<th>Model</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
</tr>
</thead>
<tbody>
<tr>
<td>Deoxys-TBC-256</td>
<td>SK</td>
<td>1</td>
<td>5</td>
<td>9</td>
<td>25</td>
<td>26</td>
<td>30</td>
<td>34</td>
<td>50</td>
</tr>
<tr>
<td>(14 rounds)</td>
<td>RTK</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>5</td>
<td>9</td>
<td>12</td>
<td>≥16</td>
<td>≥19</td>
</tr>
<tr>
<td>AES-256</td>
<td>SK</td>
<td>1</td>
<td>5</td>
<td>9</td>
<td>25</td>
<td>26</td>
<td>30</td>
<td>34</td>
<td>50</td>
</tr>
<tr>
<td>(14 rounds)</td>
<td>RTK</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>3</td>
<td>5</td>
<td>5</td>
<td>5</td>
<td>10</td>
</tr>
</tbody>
</table>

Comparison of security claims

Deoxys-TBC-256 provides a better resistance than AES-256 against plain related-key attacks, while being more efficient (no Sbox in key-schedule, just byte permutation and a few LFSRs)
Comparing Deoxys-TBC and AES

Deoxys-TBC applies this STK idea to the AES  [JNP-AC14]

<table>
<thead>
<tr>
<th>Cipher</th>
<th>Model</th>
<th>Rounds</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>1  2  3  4  5  6  7  8</td>
</tr>
<tr>
<td>Deoxys-TBC-256</td>
<td>SK</td>
<td>1  5  9 25 26 30 34 50</td>
</tr>
<tr>
<td>(14 rounds)</td>
<td></td>
<td>≥16  ≥19</td>
</tr>
<tr>
<td></td>
<td>RTK</td>
<td>0  0  1  5  9  12  ≥16  ≥19</td>
</tr>
<tr>
<td>AES-256</td>
<td>SK</td>
<td>1  5  9 25 26 30 34 50</td>
</tr>
<tr>
<td>(14 rounds)</td>
<td></td>
<td>5  5  5  5  5  10</td>
</tr>
</tbody>
</table>

Interesting research topic:

- is it be possible to find a permutation that guarantees even more active Sboxes? Or maybe a different tweakey schedule?
- can an attacker exploit the freedom degrees for more advanced attacks
SKINNY applies this STK idea to lightweight crypto [BJK+-C16]
Many other ad-hoc TBCs

Threelfish [FLS+-08]

KIASU-TBC, Deoxys-TBC and Joltik-TBC [JNP-AC14]

Minalpher [STA+-14]

Scream and iScream [GLS+-14]

Skinny and Mantis [BJK+-C16]

QARMA [Ava-FSE17]

Clyde-128 [BBB+-19]

Lilliput [ABC+-19]

CRAFT [BLM+-FSE19]

T-Twine [SMS+-I19]

Pholkos [BLLS+-eP20]

...
1. Introduction
2. Tweakable Block Ciphers Designs
   - Block Cipher-Based TBC
   - Ad-hoc TBC Constructions
3. Tweakable Block Ciphers for AE
4. Conclusion
Beyond birthday-bound security

Classical BC-based AE modes only provide birthday security

Reason: internal collisions on a $n$-bit value gets you a $q^2/2^n$ term in your security proofs. May lead to birthday complexity attacks. Complex proof.

Ex: OCB3 [KR-FSE11]
Beyond birthday-bound security

TBC-based AE modes can easily provide beyond birthday-bound (BBB) security

(picture from [KR-FSE11])

Use tweak input with nonce and counter to always ensure a new TBC instance is called. Easier to understand, better bounds, simpler proofs. priv. bound is 0.

Ex: ΘCB3 [KR-FSE11]
Romulus-N : a lightweight AE mode

Romulus-N [IKMP-19] :

lightweight BBB nonce-respecting AEAD

trades parallelism for small area
Designing an AE mode: what internal primitive to use?

BC?

Permutation?

TBC?
Designing an *AE mode*: what internal primitive to use?

First we need to get an estimation of what is the rate of a BC/TBC/Permutation
We define rate according to output size only
Is it justified?

On scaling costs

Q: assume a $n$-bit permutation costs $x$ bitwise operations, how many do we need to build a $2n$-bit permutation?

A: at least $\times 2$ and probably a bit more:

- Keccak: about $\times 2.2 \sim 2.32$
- PHOTON: about $\times 2$
- SPONGENT: about $\times 4$
We define **rate** according to **output size** only

Is it justified?

**On scaling costs**

**Q**: assume a *n*-bit TBC with *t*-bit tweakey costs \( \times \) **bitwise operations**, how many do we need to build a *2n*-bit TBC with *t*-bit tweakey?

**A**: at least \( \times2 \) and probably a bit more:

- **SKINNY**: about \( \times2.22 \)
- **GIFT**: about \( \times2.84 \)
- **SIMON and SPECK**: about \( \times3.16 \) and \( \times2.37 \)
We define **rate** according to **output size** only

Is it justified?

**On scaling costs**

**Q:** assume a *n*-bit TBC with *t*-bit tweakey costs $x$ *bitwise operations*, how many do we need to build a *n*-bit TBC with 2*t*-bit tweakey?

**A:** much less than $\times 2$:

- **SKINNY**: about $\times 1.1 \sim 1.2$
- **Deoxys**: about $\times 1.3$
- **AES (key)**: about $\times 1.4$
- **SIMON and SPECK (key)**: about $\times 1.06$
Conclusion:

- Increasing block/permutation size costs a lot!
- Increasing tweakey size doesn’t cost much
- Rate should be defined according to the output size

Try to use an internal primitive with the smallest output size as possible for a given security level!
Use case 1: minimal area

In this scenario, we don’t care if the ciphering process is really slow, we just want to minimize area (typically bit-serial or word-serial implementation)

- We will cipher $m$-bit at a time ($m$ is small)
- We want at least $n$-bit security, with a $n$-bit key
Use case 2: low energy consumption and lightweight

In this scenario, we want a small area and good throughput performances (typically round-based implementation)

Efficiency = state size/rate (the lowest the better, basically estimates the inverse of throughput-to-area ratio)

- We will cipher about \( n \)-bit at a time
- We want at least \( n \)-bit security, with a \( n \)-bit key
Use case 3: fast MAC/encryption

In this scenario, we want good throughput performances (high rate)

- We can cipher more than $n$-bit at a time, if needed
- We want at least $n$-bit security, with a $n$-bit key
## TBC Designs

<table>
<thead>
<tr>
<th>Min State Size (S)</th>
<th>Max Rate (R)</th>
<th>Best efficiency (S/R)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>enc.</td>
<td>auth.</td>
</tr>
<tr>
<td>BC</td>
<td>$4n \rightarrow 3n$</td>
<td>1</td>
</tr>
<tr>
<td>Sponge</td>
<td>$2n$</td>
<td>$1/2 \rightarrow 1$</td>
</tr>
<tr>
<td>TBC</td>
<td>$3n \rightarrow 2n$</td>
<td>1</td>
</tr>
</tbody>
</table>

### Use cases

- **Use case 1**: **min. state** with **sponges** (TBC can also do $2n$)
- **Use case 2**: **best efficiency** with **TBC** (reached at lightest point)
- **Use case 3**: **best rate** with **TBC** (for auth.)

### Comments

- **efficiency of sponge is worse than TBC** in theory because one needs a permutation larger than $n$ (effect reduced with a non-hermetic sponge)
- **TBC**: it seems we can increase the auth rate indefinitely by using a bigger tweak (true in practice ... but only up to a certain level)
## 128-bit security

<table>
<thead>
<tr>
<th>Scheme</th>
<th>State Size (S)</th>
<th>Rate enc. (R)</th>
<th>Rate auth. (S/R)</th>
<th>Efficiency enc.</th>
<th>Efficiency auth.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Romulus-N1</td>
<td>3.5n</td>
<td>1</td>
<td>2</td>
<td>3.5n</td>
<td>1.75n</td>
</tr>
<tr>
<td>Romulus-N3</td>
<td>3n</td>
<td>1</td>
<td>7/4</td>
<td>3n</td>
<td>1.71n</td>
</tr>
<tr>
<td>ΘCB3</td>
<td>4.5n</td>
<td>1</td>
<td>1</td>
<td>4.5n</td>
<td>4.5n</td>
</tr>
<tr>
<td>COFB</td>
<td>4n</td>
<td>1</td>
<td>1</td>
<td>4n</td>
<td>4n</td>
</tr>
<tr>
<td>DUPLEX (r ≪ n)</td>
<td>→ 2n</td>
<td>→ 0</td>
<td>1</td>
<td>→ ∞</td>
<td>→ 2n</td>
</tr>
<tr>
<td>DUPLEX (r = n)</td>
<td>3n</td>
<td>1/3</td>
<td>1</td>
<td>9n</td>
<td>3n</td>
</tr>
<tr>
<td>DUPLEX (r = 2n)</td>
<td>4n</td>
<td>1/2</td>
<td>1</td>
<td>8n</td>
<td>4n</td>
</tr>
<tr>
<td>DUPLEX (r ≫ 2n)</td>
<td>→ ∞</td>
<td>→ 1</td>
<td>1</td>
<td>→ ∞</td>
<td>→ ∞</td>
</tr>
<tr>
<td>BEETLE</td>
<td>2.1n</td>
<td>1/2</td>
<td>1/2</td>
<td>4.2n</td>
<td>4.2n</td>
</tr>
<tr>
<td>ASCON-128</td>
<td>3.5n</td>
<td>1/5</td>
<td>1/5</td>
<td>17.5n</td>
<td>17.5n</td>
</tr>
<tr>
<td>Ascon-128a</td>
<td>3.5n</td>
<td>2/5</td>
<td>2/5</td>
<td>8.75n</td>
<td>8.75n</td>
</tr>
</tbody>
</table>
Flexibility of the TBC

**AE mode design process:**
fix the **output size** of the TBC according to your security need, then play with the tweak size to get the proper rate and state size according to your constraints.

Don’t use a large output size internal primitive if you only want a security of $n$ bits!
Idea:

Since auth. rate increases with the size of tweak, why not try constructions with **huge tweaks** for crazy auth. efficiency?

率 will eventually reach a limit, but where?

Deoxys-128/1024 or Skinny-128/1024 variants would theoretically provide 50% ~ 100% speed-up (ongoing work)
Infinitweak

Idea:
Since auth. rate increases with the size of tweak, why not trying constructions with huge tweaks for crazy auth. efficiency?

Interesting research topic:
▶ How can we design such a very large tweak TBC?
▶ What tweakey construction to minimize cancellations?
Outline

1. Introduction

2. Tweakable Block Ciphers Designs
   ▶ Block Cipher-Based TBC
   ▶ Ad-hoc TBC Constructions

3. Tweakable Block Ciphers for AE

4. Conclusion
TBCs are promising primitives

- many more applications:
  - side-channels resistance (modes and implementations)
  - Forkcipher for small messages [ALP+-AC19]
  - easy misuse-resistance/RUP, for example with Romulus-M [IKMP-19]
  - Multi-users security (ongoing work with B. Cogliati): put separately counter, nonce and key in the tweak input of the TBC!
  - Hashing/XOF for example with Naito’s MDPH [N-LC19] construction used for Romulus-H. Ongoing work: blazing fast Deoxys-TBC-based hash function with speed similar to KangarooTwelve [BDP+-ACNS18]
  - backdoor ciphers (MALICIOUS framework [PW-C20]) can use TBC with XOF-based tweak schedule

- many open problems, many interesting research topics for TBCs, both in cryptanalysis and design
Looking for a PhD/postdoc position to work on anything related to cryptography?

Contact me!
Thank you!