arm

The QARMA Chakra

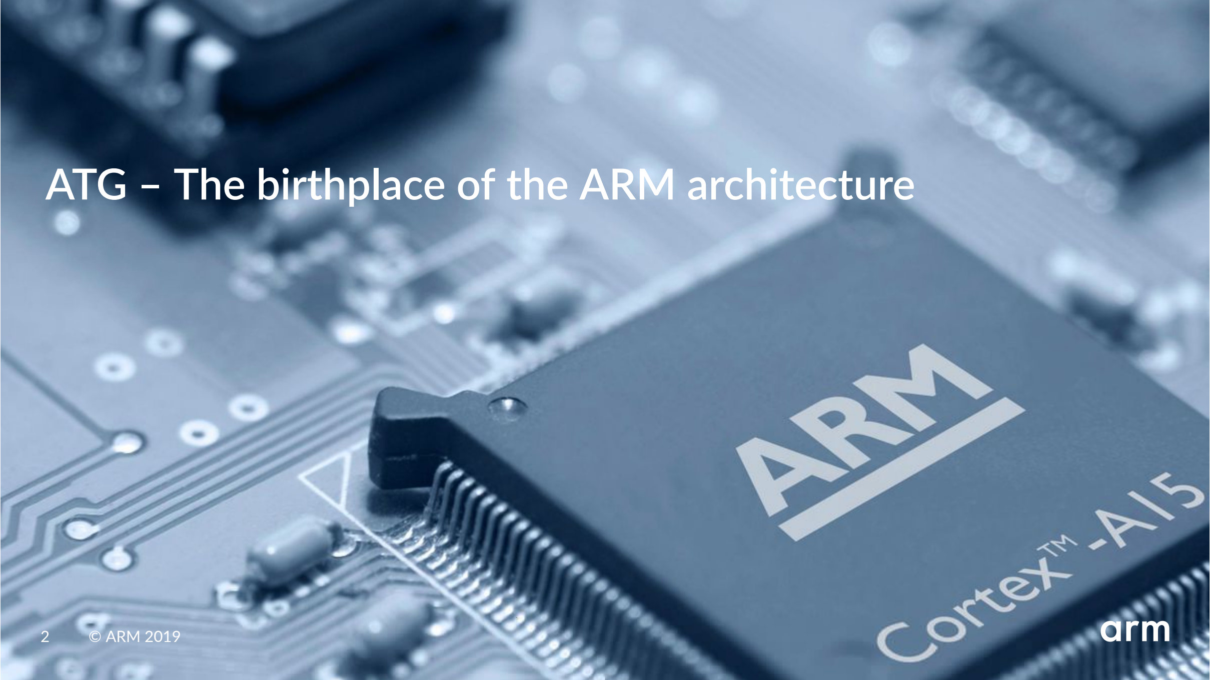
Roberto Avanzi

Architecture and Technology Group

roberto.avanzi@arm.com

Bar Ilan University – 2019 LCD – March 31st, 2019

ATG – The birthplace of the ARM architecture



Acknowledgements

What follows is not only my work.

In fact, quite a few parts of it are joint work.

Some of the folks that suffered greatly while working with me are Subhadeep Banik, Light Darkman, Senyang Huang, Francesco Regazzoni, and Andrey Bogdanov.

Acknowledgements

Thanks also to: Can Acar, Satish Anand, Christof Beierle, Christina Boura, Antonio Cardoso Costa, Mike Campbell, Alexander Dent, Simonetta Diaz, Itai Dinur, Xiaoyang Dong, Maria Eichlseder, Alexandr Gantman, Renwei Ge, Scott McGregor, Richard Grisenthwaite, Yatin Hoskote, Jérémy Jean, Gregor Leander, Jason Parker and the amazing, spectacular REDACTED team he leads at ARM, Rene Peralta, Prakash Ramrakhyani, Andrew Rose, Greg Rose, Brian Rosenberg, Andreas Sandberg, David Schall, Arrigo Triulzi, Meltem Sönmez Turan, and Robert Turner.

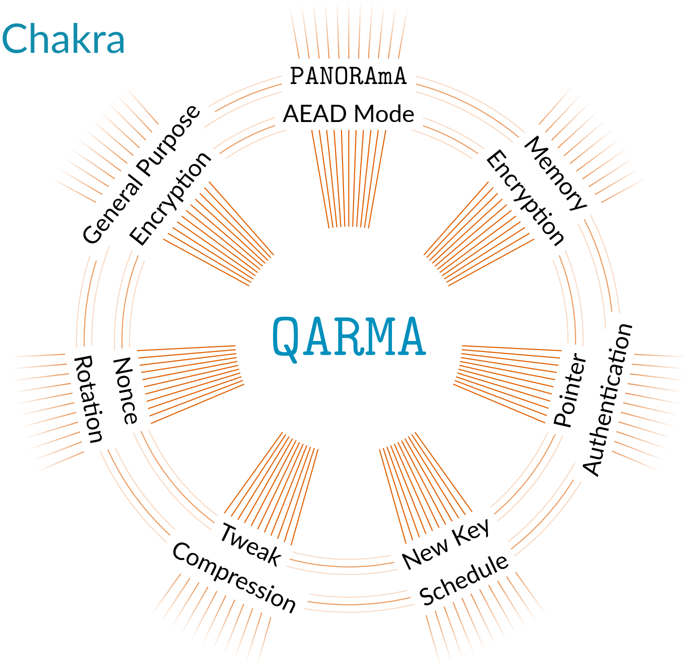
We used tools developed by: Leo Perrin; Oleksandr Kazymyrov, Maksim Storetvedt, and Anna Maria Eilertsen; Stjepan Picek, Lejla Batina, Domagoj Jakobović, Barış Ege, and Marin Golub; Gregor Leander, Brice Minaud, and Sondre Rønjom; and the SageMath and GUROBI developers.

Connections

What is the `QARMA` Chakra

A set of technologies developed *around* `QARMA`, connected with each other and to `QARMA` itself.

The QARMA Chakra



arm

Feature Overview

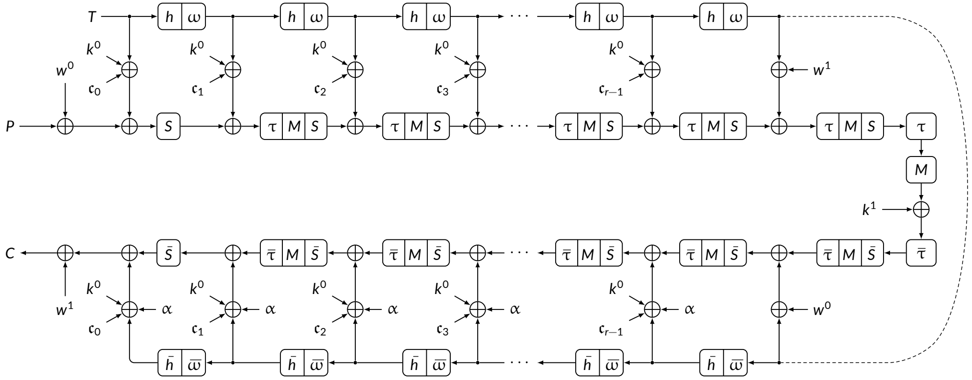
… a cipher partly designed on the slopes of Mount Carmel …

$$\overset{\text{Qualcomm}}{\overset{\uparrow}{\mathtt{Q}}} + \overbrace{\mathtt{A} + \mathtt{R} + \mathtt{M}}^{\text{ARM}} + \overset{\text{Authenticator}}{\overset{\uparrow}{\mathtt{A}}}$$

Roberto M. Avanzi

(and it might badly affect my karma)

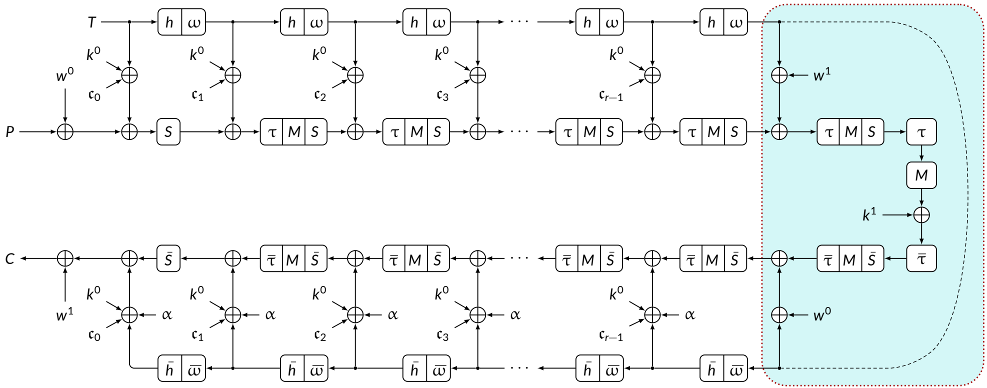
QARMA Encryption



Texts / tweak / state = vectors of sixteen 4/8-bit cells / 4×4 matrices

τ, h = Cell Shuffles; M = Involutory Almost MDS matrix; S = 16 S-Boxes; ω = LSFR $\times$ 7

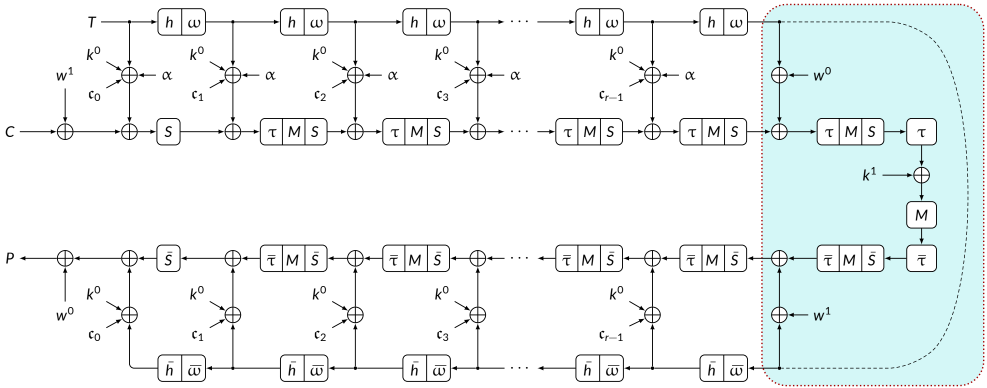
QARMA Encryption



Texts / tweak / state = vectors of sixteen 4/8-bit cells / 4×4 matrices

τ, h = Cell Shuffles; M = Involutory Almost MDS matrix; S = 16 S-Boxes; ω = LSFR $\times$ 7

QARMA Decryption

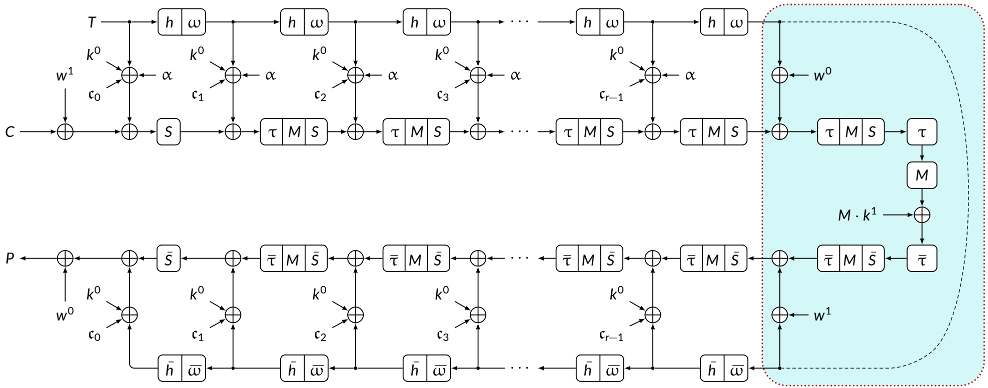


Texts / tweak / state = vectors of sixteen 4/8-bit cells / 4×4 matrices

τ, h = Cell Shuffles; M = Involutory Almost MDS matrix; S = 16 S-Boxes; ω = LSFR $\times$ 7

Decrypt with: $k^0 \mapsto k^0 \oplus \alpha$, swap w^0 and w^1

QARMA Decryption

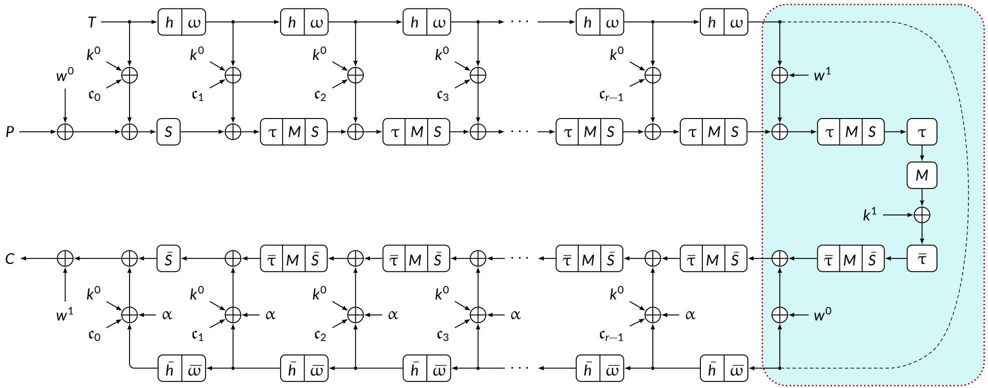


Texts / tweak / state = vectors of sixteen 4/8-bit cells / 4×4 matrices

τ, h = Cell Shuffles; M = Involutory Almost MDS matrix; S = 16 S-Boxes; ω = LSFR $\times$ 7

Decrypt with: $k^0 \mapsto k^0 \oplus \alpha$, swap w^0 and w^1, replace $k^1 \mapsto M \cdot k^1$

QARMA Encryption



Texts / tweak / state = vectors of sixteen 4/8-bit cells / 4×4 matrices

τ, h = Cell Shuffles; M = Involutory Almost MDS matrix; S = 16 S-Boxes; ω = LSFR $\times$ 7

Decrypt with: $k^0 \mapsto k^0 \oplus \alpha$, swap w^0 and w^1, replace $k^1 \mapsto M \cdot k^1$

Features

- QARMA-64 and QARMA-128 are Public Domain!
- Builds on very well understood design methodologies – all aspects explained and verifiable.
- Designed 18 years after the AES, 5 years after PRINCE, learns from Even-Mansour, MIDORI.
- Stealthily already deployed (Pointer Authentication) – ARM partners already know how to implement it!
- Tweakable. Ideal for ΘCB-like modes with BBB security.
- Very short latency (best in class).
- Area and power consumption not minimal, but very small. Good for parallelism.
- In HW, round probably the lightest among those with a full layer of optimal 4-bit S-Boxes with full diffusion and all non-zero component functions of degree 3, and Almost-MDS diffusion.
- Various pipelining and/or unrolling options possible, including very fine grained.
- Code size: even a t-box implementation should be smaller than the AES's.
- Mostly nibble and byte based, so implementation easy on microcontrollers.
- Cool name.
- Hopefully, also secure (so far so good).

Considered attacks

- Linear and differential cryptanalysis (MILP models, following Beierle)
- —, under related tweak model (MILP models, following Beierle)
- Reflection Attacks (follows from structure)
- Generic attacks on Even-Mansour schemes (follows from structure)
- Slide attacks (follows from round heterogeneity)
- Meet-in-the-middle attacks (following `MIDORI`/`MANTIS`)
- Invariant subspace attacks (heuristic arguments)
- Algebraic cryptanalysis (equations and variables counting, degree growth)
- Imp. diff. & zero corr. linear cryptanalysis (following Sun et al. EC '16)
- Higher order differential cryptanalysis (following `MIDORI`/`MANTIS`)
- And more...

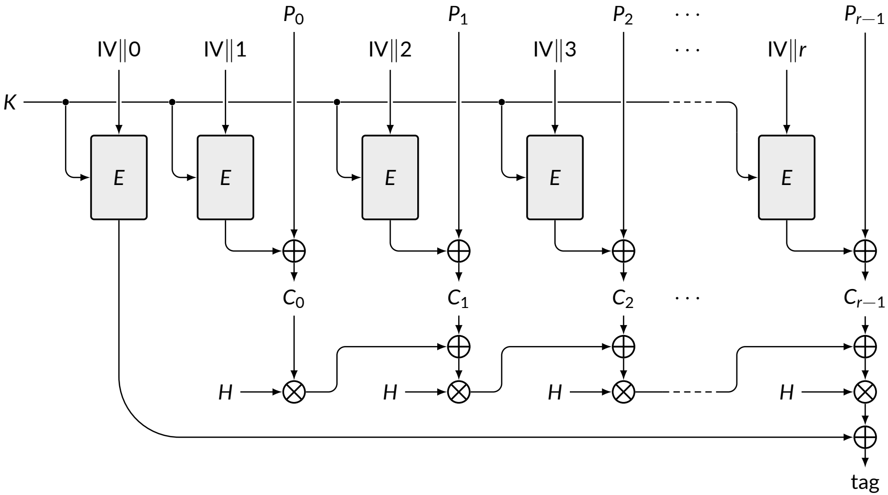
Cryptanalysis of QARMA – Selected Published Results

Cipher	Rounds Attacked	Outer Whitening?	Attack Complexity Time	Data	Memory	Technique	Ref.
64	$4+6$	N	$2^{116}+2^{70.1}$	2^{53} CP	2^{116}	MITM	[ZD16]
64	$4+4$	Y	$2^{33}+2^{90}$	2^{16} CP	2^{90}	MITM	[LJ18]
64	$4+5$	Y	$2^{48}+2^{89}$	2^{16} CP	2^{89}	MITM	[LJ18]
64	$4+6$	Y	2^{72}	2^{61} CP	$2^{78.2}$ bits	trunc. imp. diff.	[YQC18]
64	$4+6$	Y	2^{59}	2^{59} KP	$2^{29.6}$ bits	rel-tweak stat. sat.	[LHW19]
64	$4+7$	Y	$2^{120.4}$	2^{61} CP	2^{116}	trunc. imp. diff.	[YQC18]
64	$3+8$	Y	$2^{64.4}+2^{80}$	2^{61} CP	2^{61}	imp. diff.	[ZDW18]
64	$4+8$	Y	$2^{66.2}$	$2^{48.4}$ CP	$2^{53.70}$	zero corr./Integral	[ADG+19]
128	$4+6$	N	$2^{232}+2^{141.7}$	2^{105} CP	2^{232}	MITM	[ZD16]
128	$5+5$	Y	2^{156}	2^{88} CP	2^{152} bits	MITM	[LJ18]
128*	$4+6$	Y	$2^{237.3}$	2^{122} CP	2^{144}	trunc. imp. diff.	[YQC18]
128*	$4+7$	Y	$2^{241.8}$	2^{122} CP	2^{232}	trunc. imp. diff.	[YQC18]
128	$4+7$	Y	$2^{126.1}$	$2^{126.1}$ KP	2^{71} bits	rel-tweak stat. sat.	[LHW19]

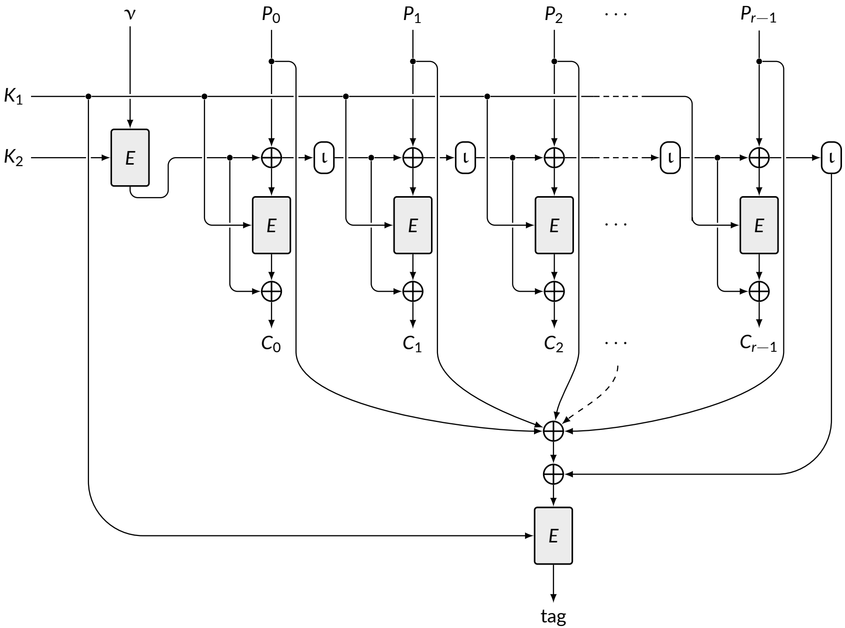
Implementation results (Power at 10 MHz)

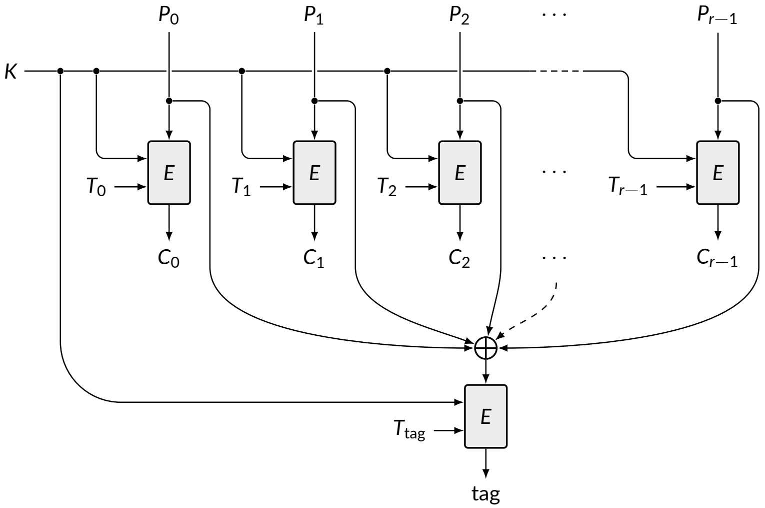
	Block Cipher	Area (GE)	Power (mW)	Energy (nJ)	Delay (ns)
1	MIDORI-128	21647	17.60	1.76	18.80
2	AES-128	51126	66.33	6.63	25.10
	AES-192	58313	87.47	8.75	28.91
	AES-256	71711	133.74	13.37	33.78
3	Deoxys-BC-256	61713	108.83	10.88	34.91
	Deoxys-BC-384	74940	145.59	14.56	40.04
4	QARMA-128_{11}	31242	29.05	2.91	**17.87**
	QARMA-128_{12}	33827	41.59	4.16	**19.35**
	QARMA-128_{13}	36412	48.42	4.84	**20.83**
	QARMA-128_{14}	38998	55.78	5.58	**22.32**

Memory Encryption



P_0
P_1
P_2
$\cdots$
P_{r-1}
IV||0
IV||1
IV||2
IV||3
$\cdots$
IV||r
K
E
E
E
E
E
C_0
C_1
C_2
$\cdots$
C_{r-1}
H
H
H
H
tag


ν
P_0
P_1
P_2
. . .
P_{r-1}
K_1
K_2
E
ι
E
E
E
. . .
E
C_0
C_1
C_2
. . .
C_{r-1}
E
tag


P_0
P_1
P_2
...
P_{r-1}
K
E
E
E
...
E
T_0
T_1
T_2
T_{r-1}
C_0
C_1
C_2
...
C_{r-1}
E
T_{tag}
tag

… and many others.

General Purpose Encryption (Including Data at Rest)

Introducing PANORAmA

PArallelisable

NOnce

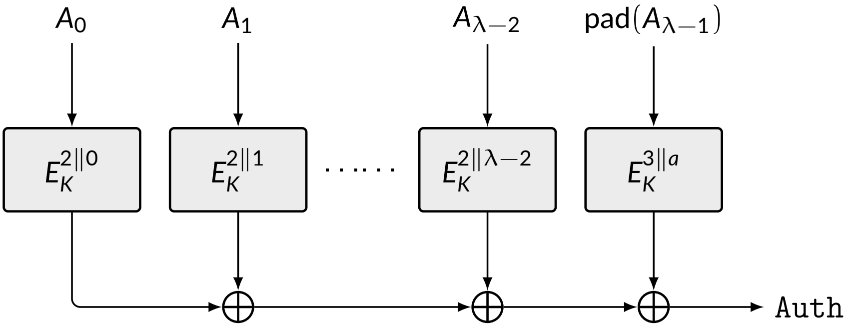
Rotating

Authenticated Encryption

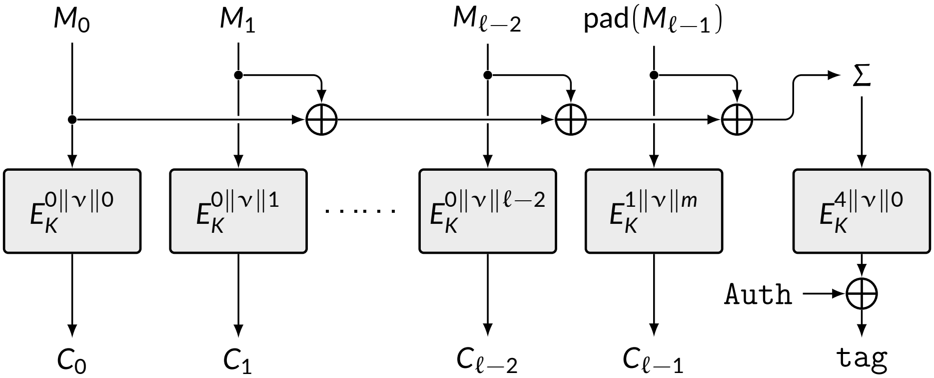
cum

Associated Data

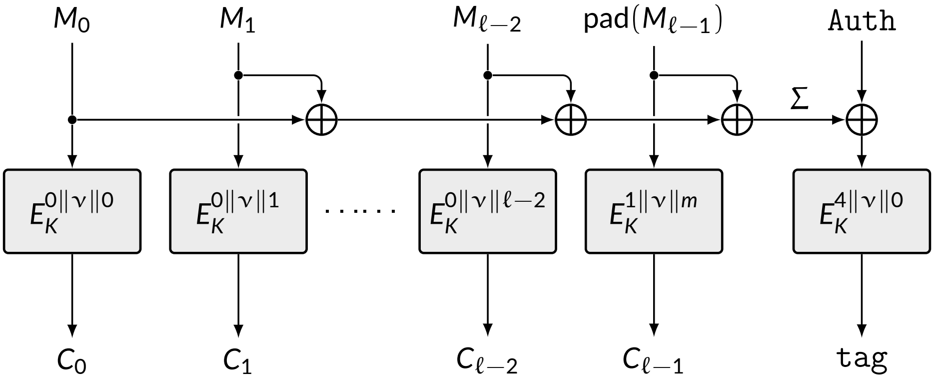
AD Processing



Message Processing



Alternative Message Processing



Usage

- These schemes can be instantiated with any compatible TBC.
- `QARMA` is suitable.

Nonce Rotation

Entropy is like good Real Estate – always at a Premium

- Suppose you have a TBC with only a 128 bit tweak (or even a 64-bit one!).
- Suppose there is a call for standards that asks for 96-bit nonces and processing of at least 2^{46} blocks.
- But in a 128-bit tweak you can only fit, say, 4 bits for tweak domain separation, 96 bits for the nonce, and 28 bits for a block index.
- What can you do?

~~Design a new cipher with larger tweak?~~
~~(Risky and expensive, usually 40++% more latency, see for instance `SKINNY`).~~

If the nonce itself is a counter, increase it every 2^{28} blocks – the primitive should return the new nonce value if updated.

In general, change ("rotate") it every 2^{28} blocks – in an unpredictable way, so that nonce, and thus tweak repetitions cannot be exploited.

Entropy is like good Real Estate – always at a Premium

- Suppose you have a TBC with only a 128 bit tweak (or even a 64-bit one!).
- Suppose there is a call for standards that asks for 96-bit nonces and processing of at least 2^{46} blocks.
- But in a 128-bit tweak you can only fit, say, 4 bits for tweak domain separation, 96 bits for the nonce, and 28 bits for a block index.
- What can you do?

Design a new cipher with larger tweak?
(Risky and expensive, usually 40++% more latency, see for instance `SKINNY`).

If the nonce itself is a counter, increase it every 2^{28} blocks – the primitive should return the new nonce value if updated.

In general, change ("rotate") it every 2^{28} blocks – in an unpredictable way, so that nonce, and thus tweak repetitions cannot be exploited.

Entropy is like good Real Estate – always at a Premium

- Suppose you have a TBC with only a 128 bit tweak (or even a 64-bit one!).
- Suppose there is a call for standards that asks for 96-bit nonces and processing of at least 2^{46} blocks.
- But in a 128-bit tweak you can only fit, say, 4 bits for tweak domain separation, 96 bits for the nonce, and 28 bits for a block index.
- What can you do?

~~Design a new cipher with larger tweak?~~
~~(Risky and expensive, usually 40++% more latency, see for instance `SKINNY`).~~

If the nonce itself is a counter, increase it every 2^{28} blocks – the primitive should return the new nonce value if updated.

In general, change (“rotate”) it every 2^{28} blocks – in an unpredictable way, so that nonce, and thus tweak repetitions cannot be exploited.

Entropy is like good Real Estate – always at a Premium

- Suppose you have a TBC with only a 128 bit tweak (or even a 64-bit one!).
- Suppose there is a call for standards that asks for 96-bit nonces and processing of at least 2^{46} blocks.
- But in a 128-bit tweak you can only fit, say, 4 bits for tweak domain separation, 96 bits for the nonce, and 28 bits for a block index.
- What can you do?

~~Design a new cipher with larger tweak?~~
~~(Risky and expensive, usually 40++% more latency, see for instance SKINNY).~~

If the nonce itself is a counter, increase it every 2^{28} blocks – the primitive should return the new nonce value if updated.

In general, change ("rotate") it every 2^{28} blocks – in an unpredictable way, so that nonce, and thus tweak repetitions cannot be exploited.

Description

1. Start with a nonce ν. Set $N \hookleftarrow \nu$.
2. Encrypt first 2^{28} blocks (a <u>chunk</u>) using tweaks $0000\|N\|i$, i the block index.

   Note that i, once put into the tweak, is truncated to the least significant 28 bits.
3. Set

$$N \hookleftarrow E_K^{1111\|\nu\|(i \gg 28)}(\text{magic number})$$

   truncated to 96 bits.
4. Encrypt now up to 2^{28} blocks (if finished, stop) using tweaks $1000\|N\|i$.
5. More to encrypt? go to step 3.

arm

Analysis

1. Eve asks for encryption of all-zero messages, observes repetitions only after
$$\geqslant (2 \cdot 2^{28} + 1) \cdot \sqrt{2} \cdot 2^{48}$$
expected blocks.
2. This is roughly $2^{77.5}$. Barely distinguishable from a random function, since collisions may happen anyway after $2^{64.5}$ blocks.
3. (Lower bounds only? N is truncated to $96 \gg \frac{1}{2}128$ bits.)
4. NIST requirements only ask to be able to process at least 2^{46} blocks.
5. ???
6. Profit!

Tweak Compression

Entropy is like good Real Estate – always at a Premium

- Suppose you have a TBC with only a 128 bit tweak (or even a 64-bit one!).
- Suppose there is a call for standards that asks for 96-bit nonces and processing of at least 2^{46} blocks.
- But in a 128-bit tweak you can only fit, say, 4 bits for tweak domain separation, 96 bits for the nonce, and 28 bits for a block index.
- What can you do?

~~Design a new cipher with larger tweak?~~
~~(Risky and expensive, usually 40++% more latency, see for instance SKINNY).~~

~~Nonce rotation? Some may not like it, for whatever reason.~~

Use a PRF to compress 2 *n* or 3 *n* bits worth of tweak material onto just *n* bits, and modify the mode to make tweak collisions non-exploitable.

arm

Entropy is like good Real Estate – always at a Premium

- Suppose you have a TBC with only a 128 bit tweak (or even a 64-bit one!).
- Suppose there is a call for standards that asks for 96-bit nonces and processing of at least 2^{46} blocks.
- But in a 128-bit tweak you can only fit, say, 4 bits for tweak domain separation, 96 bits for the nonce, and 28 bits for a block index.
- What can you do?

Design a new cipher with larger tweak?
(Risky and expensive, usually 40++% more latency, see for instance `SKINNY`).

~~Nonce rotation? Some may not like it, for whatever reason.~~

Use a PRF to compress 2 n or 3 n bits worth of tweak material onto just n bits, and modify the mode to make tweak collisions non-exploitable.

arm

Entropy is like good Real Estate – always at a Premium

- Suppose you have a TBC with only a 128 bit tweak (or even a 64-bit one!).
- Suppose there is a call for standards that asks for 96-bit nonces and processing of at least 2^{46} blocks.
- But in a 128-bit tweak you can only fit, say, 4 bits for tweak domain separation, 96 bits for the nonce, and 28 bits for a block index.
- What can you do?

~~Design a new cipher with larger tweak?~~
~~(Risky and expensive, usually 40++% more latency, see for instance `SKINNY`).~~

Nonce rotation? Some may not like it, for whatever reason.

Use a PRF to compress 2 *n* or 3 *n* bits worth of tweak material onto just *n* bits, and modify the mode to make tweak collisions non-exploitable.

Entropy is like good Real Estate – always at a Premium

- Suppose you have a TBC with only a 128 bit tweak (or even a 64-bit one!).
- Suppose there is a call for standards that asks for 96-bit nonces and processing of at least 2^{46} blocks.
- But in a 128-bit tweak you can only fit, say, 4 bits for tweak domain separation, 96 bits for the nonce, and 28 bits for a block index.
- What can you do?

~~Design a new cipher with larger tweak?~~
~~(Risky and expensive, usually 40++% more latency, see for instance `SKINNY`).~~

~~Nonce rotation? Some may not like it, for whatever reason.~~

Use a PRF to compress 2 *n* or 3 *n* bits worth of tweak material onto just *n* bits, and modify the mode to make tweak collisions non-exploitable.

arm

How to construct a suitable PRF?

- Ad hoc? Nooooooo.
- So let us try putting $T = T_0 \| \cdots \| T_{c-1}$ and
$$\text{compressed tweak} = \sum_{i=0}^{c-1} g_i(T_i) \quad .$$
- Multiply in Galois fields by keys and add? $g_i(T_i) = k_i \cdot T_i$ (First idea.) Exploitable. (Hint: collisions give relations between these keys.)
- Multiply in Galois fields by keys, add, then encrypt? Better. But. Still. Exploitable.
- Construct the PRF by adding PRPs? Much. Better.

How to construct a suitable PRF?

- Ad hoc? Noooooooo.
- So let us try putting $T = T_0 \| \cdots \| T_{c-1}$ and

$$\text{compressed tweak} = \sum_{i=0}^{c-1} g_i(T_i) \quad .$$

- Multiply in Galois fields by keys and add? $g_i(T_i) = k_i \cdot T_i$ (First idea.) Exploitable. (Hint: collisions give relations between these keys.)
- Multiply in Galois fields by keys, add, then encrypt? Better. But. Still. Exploitable.
- Construct the PRF by adding PRPs? Much. Better.

arm

How to construct a suitable PRF?

- Ad hoc? Noooooooo.
- So let us try putting $T = T_0 \| \cdots \| T_{c-1}$ and

$$\text{compressed tweak} = \sum_{i=0}^{c-1} g_i(T_i) \quad .$$

- Multiply in Galois fields by keys and add? $g_i(T_i) = k_i \cdot T_i$ (First idea.) Exploitable. (Hint: collisions give relations between these keys.)
- Multiply in Galois fields by keys, add, then encrypt? Better. But. Still. Exploitable.
- Construct the PRF by adding PRPs? Much. Better.

How to construct a suitable PRF?

- Ad hoc? Noooooooo.
- So let us try putting $T = T_0 \| \cdots \| T_{c-1}$ and

$$\text{compressed tweak} = \sum_{i=0}^{c-1} g_i(T_i) \ .$$

- Multiply in Galois fields by keys and add? $g_i(T_i) = k_i \cdot T_i$ (First idea.)
  Exploitable. (Hint: collisions give relations between these keys.)
- Multiply in Galois fields by keys, add, then encrypt?
  Better. But. Still. Exploitable.
- Construct the PRF by adding PRPs? Much. Better.

arm

How to construct a suitable PRF?

- Ad hoc? Nooooooo.
- So let us try putting $T = T_0 \| \cdots \| T_{c-1}$ and

$$\text{compressed tweak} = \sum_{i=0}^{c-1} g_i(T_i) \quad .$$

- Multiply in Galois fields by keys and add? $g_i(T_i) = k_i \cdot T_i$ (First idea.) Exploitable. (Hint: collisions give relations between these keys.)
- Multiply in Galois fields by keys, add, then encrypt? Better. But. Still. Exploitable.
- Construct the PRF by adding PRPs? Much. Better.

How to construct a suitable PRF?

- Ad hoc? Nooooooo.
- So let us try putting $T = T_0 \| \cdots \| T_{c-1}$ and

$$\text{compressed tweak} = \sum_{i=0}^{c-1} g_i(T_i) \quad .$$

- Multiply in Galois fields by keys and add? $g_i(T_i) = k_i \cdot T_i$ (First idea.) Exploitable. (Hint: collisions give relations between these keys.)
- Multiply in Galois fields by keys, add, then encrypt? Better. But. Still. Exploitable.
- Construct the PRF by adding PRPs? Much. Better.

How to construct a suitable PRF?

- Ad hoc? Nooooooo.
- So let us try putting $T = T_0 \| \cdots \| T_{c-1}$ and

$$\text{compressed tweak} = \sum_{i=0}^{c-1} g_i(T_i) \ .$$

- Multiply in Galois fields by keys and add? $g_i(T_i) = k_i \cdot T_i$ (First idea.) Exploitable. (Hint: collisions give relations between these keys.)
- Multiply in Galois fields by keys, add, then encrypt? Better. But. Still. Exploitable.
- Construct the PRF by adding PRPs? Much. Better.

How to construct a suitable PRF?

- Ad hoc? Nooooooo.
- So let us try putting $T = T_0 \| \cdots \| T_{c-1}$ and

$$\text{compressed tweak} = \sum_{i=0}^{c-1} g_i(T_i) \quad .$$

- Multiply in Galois fields by keys and add? $g_i(T_i) = k_i \cdot T_i$ (First idea.) Exploitable. (Hint: collisions give relations between these keys.)
- Multiply in Galois fields by keys, add, then encrypt? Better. But. Still. Exploitable.
- Construct the PRF by adding PRPs? Much. Better.

How to construct a suitable PRF?

- Ad hoc? Nooooooo.
- So let us try putting $T = T_0 \| \cdots \| T_{c-1}$ and

$$\text{compressed tweak} = \sum_{i=0}^{c-1} g_i(T_i) \ .$$

- Multiply in Galois fields by keys and add? $g_i(T_i) = k_i \cdot T_i$ (First idea.) Exploitable. (Hint: collisions give relations between these keys.)
- Multiply in Galois fields by keys, add, then encrypt? Better. But. Still. Exploitable.
- Construct the PRF by adding PRPs? Much. Better.

How to construct a suitable PRF?

- Ad hoc? Noooooooo.
- So let us try putting $T = T_0 \| \cdots \| T_{c-1}$ and

$$\text{compressed tweak} = \sum_{i=0}^{c-1} g_i(T_i) \quad .$$

- Multiply in Galois fields by keys and add? $g_i(T_i) = k_i \cdot T_i$ (First idea.) Exploitable. (Hint: collisions give relations between these keys.)
- Multiply in Galois fields by keys, add, then encrypt? Better. But. Still. Exploitable.
- Construct the PRF by adding PRPs? Much. Better.

arm

How to construct a suitable PRF?

- Ad hoc? Nooooooo.
- So let us try putting $T = T_0 \| \cdots \| T_{c-1}$ and

$$\text{compressed tweak} = \sum_{i=0}^{c-1} g_i(T_i) \quad .$$

- Multiply in Galois fields by keys and add? $g_i(T_i) = k_i \cdot T_i$ (First idea.)
  Exploitable. (Hint: collisions give relations between these keys.)
- Multiply in Galois fields by keys, add, then encrypt?
  Better. But. Still. Exploitable.
- Construct the PRF by adding PRPs? Much. Better.

How to construct a suitable PRF?

- Ad hoc? Nooooooo.
- So let us try putting $T = T_0 \| \cdots \| T_{c-1}$ and

$$\text{compressed tweak} = \sum_{i=0}^{c-1} g_i(T_i) \quad .$$

- Multiply in Galois fields by keys and add? $g_i(T_i) = k_i \cdot T_i$ (First idea.) Exploitable. (Hint: collisions give relations between these keys.)
- Multiply in Galois fields by keys, add, then encrypt? Better. But. Still. Exploitable.
- Construct the PRF by adding PRPs? Much. Better.

How to construct a suitable PRF?

- Ad hoc? Nooooooo.
- So let us try putting $T = T_0 \| \cdots \| T_{c-1}$ and

$$\text{compressed tweak} = \sum_{i=0}^{c-1} g_i(T_i) \quad .$$

- Multiply in Galois fields by keys and add? $g_i(T_i) = k_i \cdot T_i$ (First idea.) Exploitable. (Hint: collisions give relations between these keys.)
- Multiply in Galois fields by keys, add, then encrypt? Better. But. Still. Exploitable.
- Construct the PRF by adding PRPs? Much. Better.

How to construct a suitable PRF?

We could try

$$\text{compressed tweak} = \sum_{i=0}^{c-1} E_K^{\text{tweak}_i}(T_i)$$

followed by

$$C = E_K^{\text{compressed tweak}}(P) \quad .$$

This has two problems.

- Kills the advantages of having: A short tweak input; not too many rounds; usage of TBC based modes of operation.
- Collisions give same encryption function.

How to construct a suitable PRF?

We could try

$$\text{compressed tweak} = \sum_{i=0}^{c-1} E_K^{\text{tweak}_i}(T_i)$$

followed by

$$C = E_K^{\text{compressed tweak}}(P) \quad .$$

This has two problems.

- Kills the advantages of having: A short tweak input; not too many rounds; usage of TBC based modes of operation.
- Collisions give same encryption function.

Description

We could try

$$\text{compressed tweak} = \sum_{i=0}^{c-1} g_i(T_i \oplus K_0) \qquad \text{(I shall call you mini-}T\text{, or } T_{\text{effective}}\text{)}$$

$$g_i = \text{a few rounds of } \texttt{QARMA} \text{ with unique round constants, no tweak, key } K_1 \ .$$

For 2 n-bit long tweaks

$$P \mapsto E_K^{T_{\text{effective}}}(P \oplus o(g_0(T_0 \oplus K_0)))$$
$$C \mapsto D_K^{T_{\text{effective}}}(C) \oplus o(g_0(T_0 \oplus K_0)) \ .$$

For 3 n-bit long tweaks

$$P \mapsto E_K^{T_{\text{effective}}}(P \oplus o(g_0(T_0 \oplus K_0))) \oplus o(g_1(T_1 \oplus K_0))$$
$$C \mapsto D_K^{T_{\text{effective}}}(C \oplus o(g_1(T_1 \oplus K_0))) \oplus o(g_0(T_0 \oplus K_0)) \ .$$

Analysis

- Make sure the number of rounds is sufficient to guarantee full diffusion.
- At least 3.
- Use a bit-wise MILP model to ensure that the number of active S-Boxes is 30-ish, resp. 60-ish for each pair of functions (g_i, g_j) in the case of 64, resp. 128-bit tweak ciphers. This makes collisions not only roughly as rare as random, but expensive to construct (the $\mathtt{QARMA}$ 4-bit S-Box has 2^{-2} lin/diff biases).
- Also: adding the component functions before/after the encryptions (processed with an orthomorphism) will cause collisions to generate non-identical functions (in fact, translates, which are then not detectable).
- ???
- Profit!!!

A new Hope

A new Hope^H

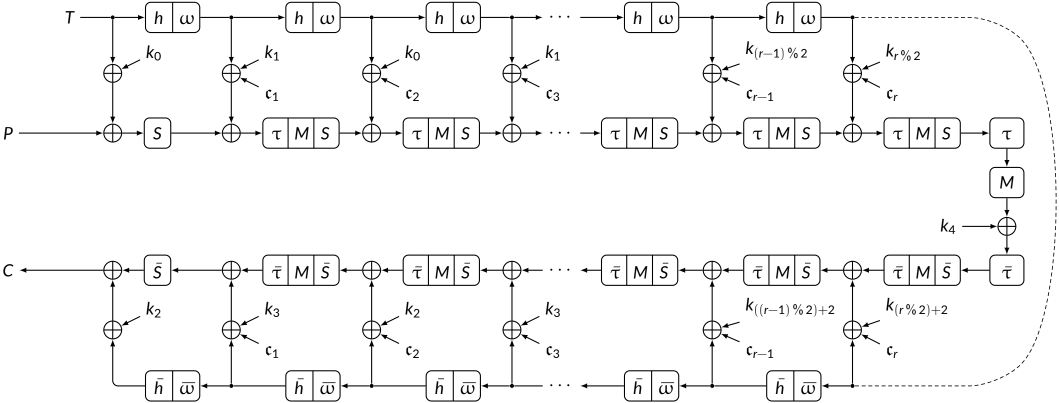
A new Hope^H^H

A new Hope^H^H^H

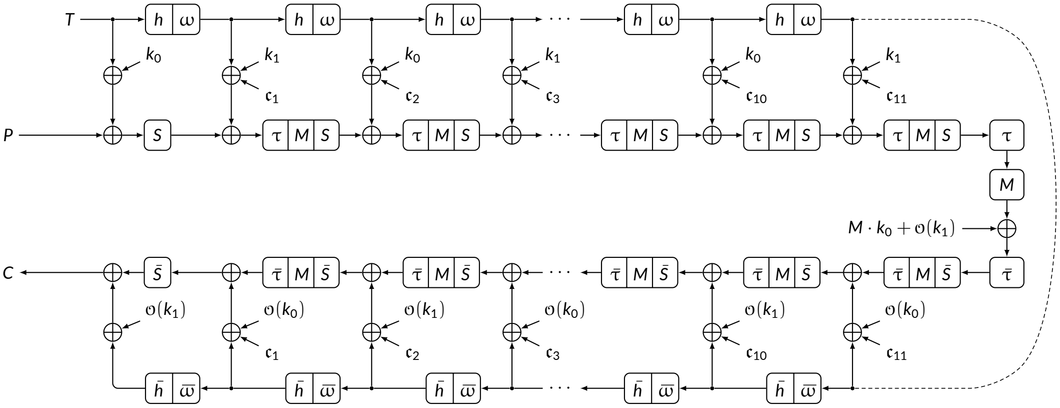
A new Hope^H^H^H^H

A new Hope^H^H^H^H Key Schedule

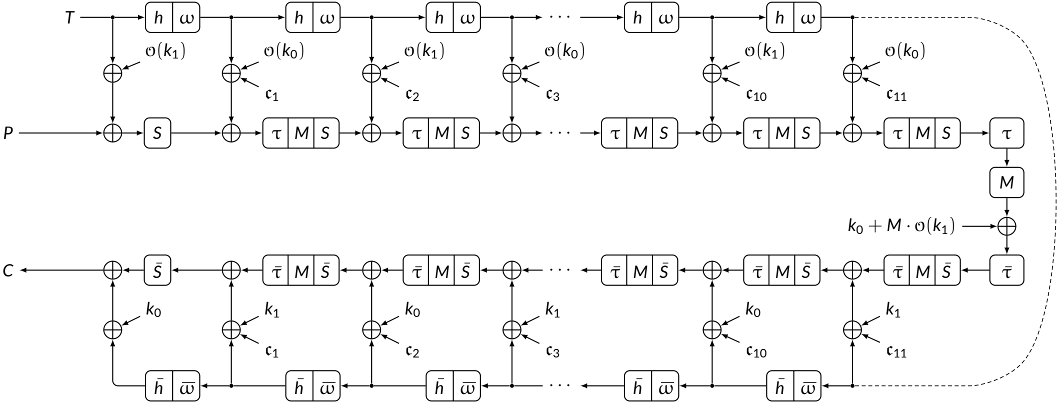
Itai Dinur suggested an Alternating key Construction



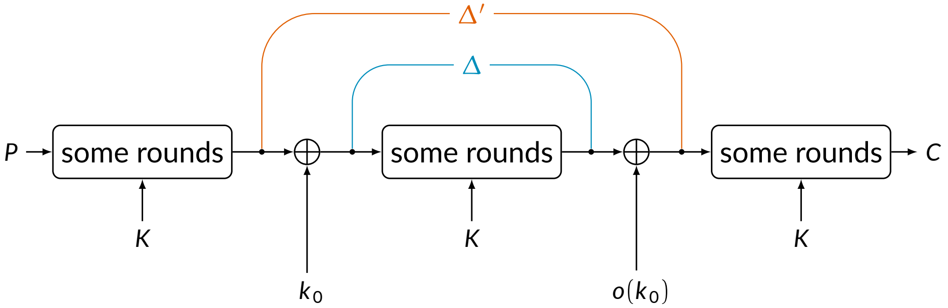
QARMA∗: Encryption



QARMA∗: Decryption



How to use a Characteristic/Distinguisher



The map $x \mapsto o(x)$ is linear, bijective, and also $x \mapsto x + o(x)$ is bijective.
This makes in general Δ' useless unless k_0 is guessed.

More Remarks

Over characteristic 2 algebras, if $o(\cdot)$ is an orthomorphism, then $o^2(\cdot)$ #metoo.

Proof: First, note that $o^2(\cdot)$ is clearly a bijective linear map.
Then $x \mapsto y := x + o(x) \mapsto y + o(y) = x + o(x) + o(x) + o^2(x) = x + o^2(x)$ as a composition of bijections is a bijection. $\square$

Hence, $o^4(\cdot)$, $o^8(\cdot)$, $o^{2^i}(\cdot)$ are orthomorphisms as well.

Idea: (Perchance, to dream?) Use a simple key schedule where we alternate between partial keys, and modify values using functions that, as much as possible, are pairwise "mutually orthomorphic."

(Ay, there's the *rub*. Doing it elegantly. Also $o(\cdot)$ is one XOR, then $o^{2^i}(\cdot)$ is 2^i XORs.)

(The reference is to Shakespeare, not Bochum.)

arm

More Remarks

Over characteristic 2 algebras, if $o(\cdot)$ is an orthomorphism, then $o^2(\cdot)$ #metoo.

Proof: First, note that $o^2(\cdot)$ is clearly a bijective linear map.
Then $x \mapsto y := x + o(x) \mapsto y + o(y) = x + o(x) + o(x) + o^2(x) = x + o^2(x)$ as a composition of bijections is a bijection. $\square$

Hence, $o^4(\cdot)$, $o^8(\cdot)$, $o^{2^i}(\cdot)$ are orthomorphisms as well.

Idea: (Perchance, to dream?) Use a simple key schedule where we alternate between partial keys, and modify values using functions that, as much as possible, are pairwise "mutually orthomorphic."

(Ay, there's the *rub*. Doing it elegantly. Also $o(\cdot)$ is one XOR, then $o^{2^i}(\cdot)$ is 2^i XORs.)

(The reference is to Shakespeare, not Bochum.)

More Remarks

Over characteristic 2 algebras, if $o(\cdot)$ is an orthomorphism, then $o^2(\cdot)$ #metoo.

Proof: First, note that $o^2(\cdot)$ is clearly a bijective linear map.
Then $x \mapsto y := x + o(x) \mapsto y + o(y) = x + o(x) + o(x) + o^2(x) = x + o^2(x)$ as a composition of bijections is a bijection. $\square$

Hence, $o^4(\cdot)$, $o^8(\cdot)$, $o^{2^i}(\cdot)$ are orthomorphisms as well.

Idea: (Perchance, to dream?) Use a simple key schedule where we alternate between partial keys, and modify values using functions that, as much as possible, are pairwise "mutually orthomorphic."

(Ay, there's the *rub*. Doing it elegantly. Also $o(\cdot)$ is one XOR, then $o^{2^i}(\cdot)$ is 2^i XORs.)

(The reference is to Shakespeare, not Bochum.)

More Remarks

Over characteristic 2 algebras, if $o(\cdot)$ is an orthomorphism, then $o^2(\cdot)$ #metoo.

Proof: First, note that $o^2(\cdot)$ is clearly a bijective linear map.
Then $x \mapsto y := x + o(x) \mapsto y + o(y) = x + o(x) + o(x) + o^2(x) = x + o^2(x)$ as a composition of bijections is a bijection. $\square$

Hence, $o^4(\cdot)$, $o^8(\cdot)$, $o^{2^i}(\cdot)$ are orthomorphisms as well.

Idea: (Perchance, to dream?) Use a simple key schedule where we alternate between partial keys, and modify values using functions that, as much as possible, are pairwise "mutually orthomorphic."

(Ay, there's the *rub*. Doing it elegantly. Also $o(\cdot)$ is one XOR, then $o^{2^i}(\cdot)$ is 2^i XORs.)

(The reference is to Shakespeare, not Bochum.)

More Remarks

Over characteristic 2 algebras, if $o(\cdot)$ is an orthomorphism, then $o^2(\cdot)$ #metoo.

Proof: First, note that $o^2(\cdot)$ is clearly a bijective linear map.
Then $x \mapsto y := x + o(x) \mapsto y + o(y) = x + o(x) + o(x) + o^2(x) = x + o^2(x)$ as a composition of bijections is a bijection. $\square$

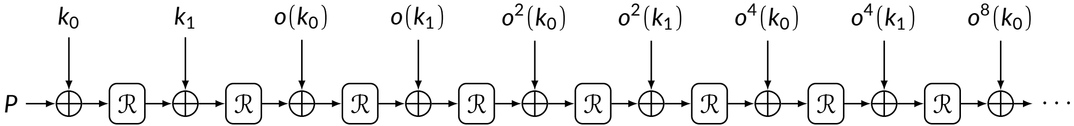
Hence, $o^4(\cdot)$, $o^8(\cdot)$, $o^{2^i}(\cdot)$ are orthomorphisms as well.

Idea: (Perchance, to dream?) Use a simple key schedule where we alternate between partial keys, and modify values using functions that, as much as possible, are pairwise "mutually orthomorphic."

(Ay, there's the *rub*. Doing it elegantly. Also $o(\cdot)$ is one XOR, then $o^{2^i}(\cdot)$ is 2^i XORs.)

(The reference is to Shakespeare, not Bochum.)

Example

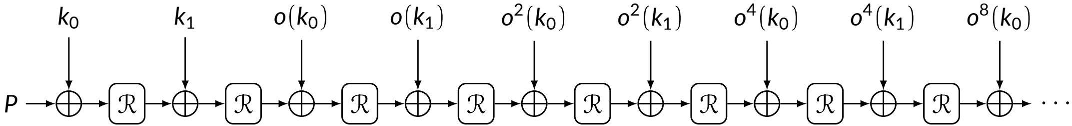


This is a possible starting point. To do: reduce (partial) sliding; and how to properly construt the forward and backward paths of a reflection cipher.

Nearly all of the cryptanalysis done so far would apply almost verbatim or have higher complexities. Invariant Subspace analysis would have to be re-run.

If you find a good way, please do it with us and maybe also Yannick Seurin. Really, be cooperative. Orr's daughter knows how to use an AK47.

Example

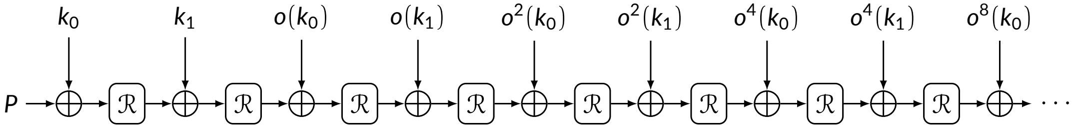


This is a possible starting point. To do: reduce (partial) sliding; and how to properly construt the forward and backward paths of a reflection cipher.

Nearly all of the cryptanalysis done so far would apply almost verbatim or have higher complexities. Invariant Subspace analysis would have to be re-run.

If you find a good way, please do it with us and maybe also Yannick Seurin. Really, be cooperative. Orr's daughter knows how to use an AK47.

Example

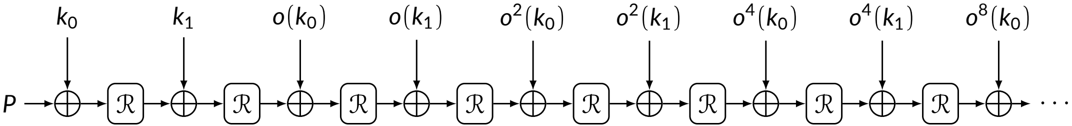


This is a possible starting point. To do: reduce (partial) sliding; and how to properly construt the forward and backward paths of a reflection cipher.

Nearly all of the cryptanalysis done so far would apply almost verbatim or have higher complexities. Invariant Subspace analysis would have to be re-run.

If you find a good way, please do it with us and maybe also Yannick Seurin. Really, be cooperative. Orr's daughter knows how to use an AK47.

Example

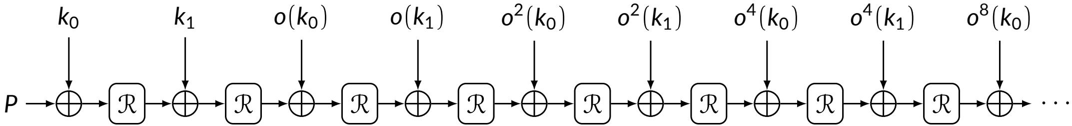


This is a possible starting point. To do: reduce (partial) sliding; and how to properly construt the forward and backward paths of a reflection cipher.

Nearly all of the cryptanalysis done so far would apply almost verbatim or have higher complexities. Invariant Subspace analysis would have to be re-run.

If you find a good way, please do it with us and maybe also Yannick Seurin.
Really, be cooperative. Orr's daughter knows how to use an AK47.

Example

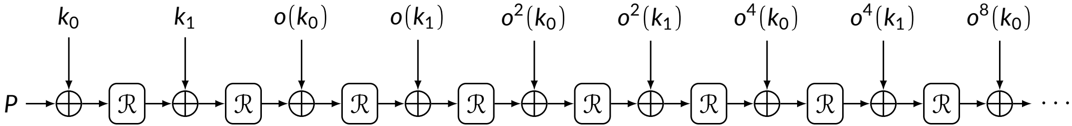


This is a possible starting point. To do: reduce (partial) sliding; and how to properly construt the forward and backward paths of a reflection cipher.

Nearly all of the cryptanalysis done so far would apply almost verbatim or have higher complexities. Invariant Subspace analysis would have to be re-run.

If you find a good way, please do it with us and maybe also Yannick Seurin. Really, be cooperative. Orr's daughter knows how to use an AK47.

Example



This is a possible starting point. To do: reduce (partial) sliding; and how to properly construt the forward and backward paths of a reflection cipher.

Nearly all of the cryptanalysis done so far would apply almost verbatim or have higher complexities. Invariant Subspace analysis would have to be re-run.

If you find a good way, please do it with us and maybe also Yannick Seurin. Really, be cooperative. Orr's daughter knows how to use an AK47.

What about `Qameleon`?

Introducing `Qameleon`

`QARMA` plus

Authentication for

MEmories that

Let

ExfiltratiON

Qameleon

- Qameleon is an AEAD suite of ciphers for various applications.
- In a nutshell, it is PANORAmA instantiated with QARMA, using in some cases Nonce Rotation or Tweak Compression to be able to cover all requirements.
- For those that have read the NIST LWC competition call, they asked for something essentially already proven and well analysed. They stopped short of saying "*just give us ASCON and we shall pretend to have a look at the rest.*" The next thing may just be a known and studied TBC with an OCB-like mode.
- We use QARMA and a subset of ΘCB (plus NR & TC that come with proofs).
- In our opinion these constructions are even better understood than sponges. ISO folks in this room, take note!

Qameleon

- Qameleon is an AEAD suite of ciphers for various applications.
- In a nutshell, it is PANORAmA instantiated with QARMA, using in some cases Nonce Rotation or Tweak Compression to be able to cover all requirements.
- For those that have read the NIST LWC competition call, they asked for something essentially already proven and well analysed. They stopped short of saying "*just give us ASCON and we shall pretend to have a look at the rest.*" The next thing may just be a known and studied TBC with an OCB-like mode.
- We use QARMA and a subset of ΘCB (plus NR & TC that come with proofs).
- In our opinion these constructions are even better understood than sponges. ISO folks in this room, take note!

Qameleon

- Qameleon is an AEAD suite of ciphers for various applications.
- In a nutshell, it is PANORAmA instantiated with QARMA, using in some cases Nonce Rotation or Tweak Compression to be able to cover all requirements.
- For those that have read the NIST LWC competition call, they asked for something essentially already proven and well analysed. They stopped short of saying "*just give us ASCON and we shall pretend to have a look at the rest.*" The next thing may just be a known and studied TBC with an OCB-like mode.
- We use QARMA and a subset of ΘCB (plus NR & TC that come with proofs).
- In our opinion these constructions are even better understood than sponges. ISO folks in this room, take note!

Qameleon

- Qameleon is an AEAD suite of ciphers for various applications.
- In a nutshell, it is PANORAmA instantiated with QARMA, using in some cases Nonce Rotation or Tweak Compression to be able to cover all requirements.
- For those that have read the NIST LWC competition call, they asked for something essentially already proven and well analysed. They stopped short of saying "*just give us ASCON and we shall pretend to have a look at the rest.*" The next thing may just be a known and studied TBC with an OCB-like mode.
- We use QARMA and a subset of ΘCB (plus NR & TC that come with proofs).
- In our opinion these constructions are even better understood than sponges. ISO folks in this room, take note!

Qameleon

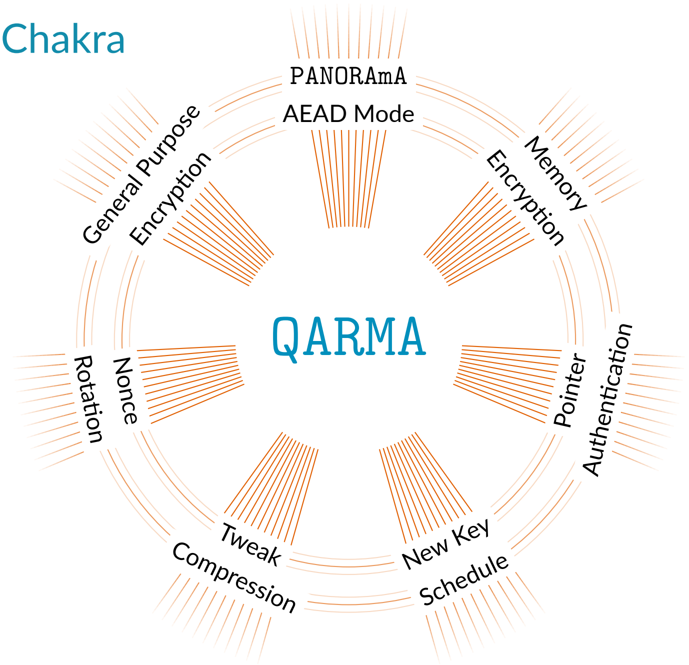
- Qameleon is an AEAD suite of ciphers for various applications.
- In a nutshell, it is PANORAmA instantiated with QARMA, using in some cases Nonce Rotation or Tweak Compression to be able to cover all requirements.
- For those that have read the NIST LWC competition call, they asked for something essentially already proven and well analysed. They stopped short of saying "*just give us ASCON and we shall pretend to have a look at the rest.*" The next thing may just be a known and studied TBC with an OCB-like mode.
- We use QARMA and a subset of ΘCB (plus NR & TC that come with proofs).
- In our opinion these constructions are even better understood than sponges. ISO folks in this room, take note!

arm

Implementation results (Power at 10 MHz)

	Variant	Block Cipher	Optimization	Area (GE)	Power(mW)	Delay(ns)
(A)	qameleon12812896gpv1	QARMA-128$_{11}$	Area	35053	35.0	20.98
			Speed	59678	69.4	9.92
		QARMA-128$_{14}$	Area	42811	53.9	25.45
			Speed	74177	109.8	11.91
(B)	qameleon128128128tcgpv1	QARMA-128$_{11}$	Area	46018	45.8	23.08
			Speed	90138	100.3	10.42
(C)	qameleon12812864gpv1	QARMA-128$_{14}$	Area	42379	52.9	24.26
			Speed	96708	86.7	9.90
(D)	qameleon1286464mev1	QARMA-128$_{11}$	Area	32457	28.6	18.99
			Speed	56873	61.9	9.00
(E)	qameleon6464tcmev1	QARMA-64$_{7}$(TC)	Area	15702	9.5	16.38
			Speed	19408	16.2	7.90
(F)	qameleon6464nnmev1	QARMA-64$_{9}$	Area	13779	9.8	16.38
			Speed	22892	19.6	8.00
	ΘCB	Deoxys-BC-384	Area	77967	112.1	57.68
			Speed	99128	178.7	29.91

The QARMA Chakra



Questions?