Qameleon v. 1.0

A Submission to the
NIST Lightweight Cryptography Standardization Process

This submission is from the following team, listed in alphabetical order:

Roberto Avanzi [1]	Architecture and Technology Group, ARM Germany GmbH
E-mail address	roberto.avanzi@gmail.com, roberto.avanzi@arm.com
Telephone	+49-174-1418294
Postal address	ARM Germany GmbH, Bretonischer Ring 16, 85630 Grasbrunn, Germany
Subhadeep Banik	School of Computer and Communication Sciences, EPFL, Switzerland
E-mail address	subhadeep.banik@epfl.ch
Telephone	+41-21-69-38127
Postal address	EPFL IC IINFCOM LASEC, INF 239, Station 14, 1015 Lausanne, Switzerland
Andrey Bogdanov	DTU COMPUTE, Technical University of Denmark
E-mail address	anbog@dtu.dk
Telephone	+45-45-25-54-72
Postal address	Technical University of Denmark, Richard Petersens Plads Building 322, room 232, 2800 Kgs. Lyngby
Orr Dunkelman	Computer Science Department, University of Haifa, Israel
E-mail address	orrd@cs.haifa.ac.il
Telephone	+972-4-828-8447 (preferred), +972-54-5291912 (secondary)
Postal address	Computer Science Department, University of Haifa, Haifa 31905, Israel
Senyang Huang	Computer Science Department, University of Haifa, Israel
E-mail address	xiaohuangbuct@gmail.com
Telephone	+972-58-4888178
Postal address	Computer Science Department, University of Haifa, Haifa 31905, Israel
Francesco Regazzoni [2]	ALaRI, Università della Svizzera italiana, Switzerland
E-mail address	regazzoni@alari.ch
Telephone	+41-58-666-4307
Postal address	ALaRI, Università della Svizzera italiana Via G. Buffi 13, 6904 Lugano, Switzerland

[1] Project lead, corresponding submitter [2] Backup corresponding submitter

Munich, March 29, 2019

(This page intentionally left blank.)

Qameleon v. 1.0

A Submission to the
NIST Lightweight Cryptography Standardization Process

This submission is from the following team, listed in alphabetical order:

Roberto Avanzi [1]	Architecture and Technology Group, ARM Germany GmbH
E-mail address	roberto.avanzi@gmail.com, roberto.avanzi@arm.com
Subhadeep Banik	School of Computer and Communication Sciences, EPFL, Switzerland
E-mail address	subhadeep.banik@epfl.ch
Andrey Bogdanov	DTU COMPUTE, Technical University of Denmark
E-mail address	anbog@dtu.dk
Orr Dunkelman	Computer Science Department, University of Haifa, Israel
E-mail address	orrd@cs.haifa.ac.il
Senyang Huang	Computer Science Department, University of Haifa, Israel
E-mail address	xiaohuangbuct@gmail.com
Francesco Regazzoni [2]	ALaRI, Università della Svizzera italiana, Switzerland
E-mail address	regazzoni@alari.ch

[1] Project lead, corresponding submitter

[2] Backup corresponding submitter

Munich, March 29, 2019

Introduction

The present document proposes Qameleon, a new *Authenticated Encryption with Associated Data (AEAD)* design based on well-understood technologies. Parameters sets and variants are suggested that can use different key and tweak sizes, and for each of them specific security levels are claimed.

Qameleon targets low-latency scenarios, such as memory encryption. This suggests that the scheme is "perfectly" parallelisable, i.e., as parallelisable as possible on a single task. In particular, we support decryption of any block (while authentication is still taking place). Moreover, as our main focus is memory encryption, we mostly target scenarios in which the nonce are not repeated.

Qameleon is a clean design composed of a mode of operation called PANORAmA, which can be used together with any compatible *Tweakable Block Cipher (TBC)*, and the TBC QARMA [Ava17]. PANORAmA is a subset of the tweaked *Offset CodeBook mode (OCB)* mode ΘCB, with an additional provision for extending the length of messages. Some simplifications, such as the direct encryption of fractional blocks, makes implementations less error-prone (and may offer reduced IP restrictions). It can also be seen as a special instantiation of the *Authenticated Encryption Mode (AEM)* [Rog04].

This choice of design has essentially no set-up time, is highly parallelisable, and can be effectively pipelined, in order to keep up also with extreme bandwidth requirements.

The block cipher QARMA that is used with this mode of operation has been actually designed for such uses. Indeed, while the development of the 64-bit version of QARMA was motivated by the requirements of Pointer Authentication for hardware assisted prevention of software exploitation [AKT14, ARM16, QPS17], the general design was intended to meet the needs of additional specific use cases such as memory encryption and the construction of keyed hash functions. At the same time the cipher has a conservative design and excellent performance.

Qameleon is performant enough to be used as a general purpose cipher, but it is especially optimised for memory encryption, and this is reflected in its name: *QARMA plus Authentication for MEmories that Let ExfiltratiON*. We recall that QARMA itself is an acronym that means *Qualcomm and ARM Authenticator* because of the first intended application of its 64-bit version.

For *Random Access Memory (RAM)* encryption a pure *Authenticated Encryption (AE)* mode would suffice, but since we encompass also export of pages or areas of memory from secure process domains (a.k.a. "enclaves," "realms," or "Secure Partitions") to insecure mass storage, the definition of a general-purpose mode with associated data is desirable. A few variants are included, addressing various use cases. The AEAD variants of Qameleon provides full 128-bit or 256-bit security for plaintext confidentiality, whereas integrity and authenticity are limited by the tag size, which can be of 64 or 128 bits.

Qameleon performs very well in hardware, being significantly faster, smaller, and requiring far less energy than the most important alternatives with similar requisites.

Organization of this document In Chapter 1 on page 7 we specify the mode of operation used by the Qameleon family. Chapter 2 on page 13 contains the specifications of tweakable block cipher QARMA. The proposed parameter sets and some variants are the subject matter of Chapter 3 on page 19. Their claimed security is also given there. In Chapter 4 on page 25 we explain the design rationale of our algorithms. Security arguments are presented in Chapter 5 on page 29. Chapter 6 on page 41 is where we report on our implementations. A feature summary, acknowledgements, lyrics for the official Qameleon song, and a bibliography round off the submission.

Contents

List of Algorithms

List of Figures

List of Tables

List of Acronyms

1 Specification of the Mode of Operation PANORAmA

This chapter contains the full specification of the mode of operation used by Qameleon, which is itself called PANORAmA: *PArallelisable NOnce Rotating Authenticated Encryption cum Associated Data*.

PANORAmA is similar to Deoxys-I by Jean, Nikolić, Peyrin, and Seurin [JNPS16], which is itself closely based on ΘBC [KR11], a variant of *Offset CodeBook mode (OCB)* [RBBK01], also known as *Tweakable Authenticated Encryption (TAE)* [LRW02, LRW11]. The differences between PANORAmA and these modes are:

1. The last block of the message, if fractional, is processed by padding and encrypting it, instead of XORing it with an encryption of zero and truncating the output. The differentiation between the cases of a full last block ending with $1\|0^{8\,(t-1)+7}$ and a fractional last block defective of t bytes, but otherwise identical, is done by tweak domain separation based on the message length. The computation of the *Associated Data (AD)* authenticator is similar.
2. The *National Institute of Standards and Technology (NIST)* call [NIS18] asks for the ability to handle messages up to $2^{50}-1$ bytes. On the other hand, we wanted to instantiate the main variants of the cipher using QARMA-128, that has only a 128-bit tweak input. However, for the principal submission variant, the nonce must be at least 96 bit, and the block counter would be too large to fit in the remaining bits. Hence we developed a technique to "rotate" the values in the nonce field in a way that is unpredictable for the attacker – in particular observing any nonce repetition would expectedly require the processing of significantly more than $2^{50}-1$ bytes, and even such an event should not be exploitable. Alternatively, developing an idea sketched in [Ava17, § 2.9], we also develop a way to use longer tweaks while not changing the proven core of the underlying *Tweakable Block Cipher (TBC)*.

With our choices we aim, among other things, at making implementation easier and avoiding potential pitfalls such as those that affected the OCB2 mode, e.g. [IM18, Poe18, Iwa18].

Regarding a format-preserving processing of the fractional last block: we could re-include it at the NIST's bidding (as this is NIST's right). But we do not feel urged to include it at this stage.

It is possible to define a mode of operation similar to Deoxys-II [JNPS16] as well to address the nonce-repeating scenario: We leave this open for discussion and may add this to the submission if the selection committee feels that such a mode of operation is required. Note that Deoxys-II itself is a variant of *Synthetic Counter-in-Tweak (SCT)*, an inverse-free authenticated encryption mode published in [PS16].

1.1 Notation

We denote by $E_K^T(P)$ the ciphering of the n-bit plaintext P with the tweakable block cipher QARMA (by default QARMA-128, but also QARMA-64 could be used) with a $2\,n$-bit key K and a t-bit tweak T. Usually $t = n$ but if tweak compression is used t can be larger (as discussed for $2\,n$- and $3\,n$-bit tweaks in Chapter 3).

The symbol $\|$ denotes the concatenation operation. For any bit string $x \in \{0,1\}^*$ let x denote its length. The symbol ϵ denotes the empty string. The function pad applies the 10^* padding on n bits i.e.

$$\mathrm{pad}(X) = \begin{cases} X & \text{if } |X| = n \ , \\ X\|1\|0^{n-|X|-1} & \text{if } 1 \le |X| < n \ , \\ \epsilon & \text{if } X = \epsilon \ . \end{cases}$$

We note that pad(·) does not offer prefix-free encoding, but the calls to the tweakable block cipher QARMA enjoy domain separation which ensure that for two different messages m_1, m_2 such that $\text{pad}(m_1) = \text{pad}(m_2)$, the sequence of calls to the QARMA differ.

For any number i in an *algorithm*, we denote by $\lfloor i \rfloor_p$ its truncation to p bits, taking the p least significant bits of i. If i is shorter than p bits, then its upper bits are padded with zeros. Similarly, $\lceil i \rceil_p$ is the truncation of i to its p most significant bits, zero filled in the least significant bits if necessary.

1.2 Algorithms

The *Authenticated Encryption with Associated Data (AEAD)* mode PANORAmA is composed of an encryption algorithm and a verification/decryption algorithm. It is for nonce-respecting users.

We describe it in detail the 128-bit version first, and then mention the differences for the 64-bit case.

1.2.1 Encryption and tag generation

The encryption algorithm $\mathscr{E}$ takes as input a variable-length plaintext byte-string M (with $m = |M|/8$), a variable-length associated-data byte-string A (with $a = |A|/8$), a nonce N, and a 256-bit key K (since the key used in the mode of operation and the key actually used in all instances of the underlying TBC are the same, we use the same letter for both of them). The algorithm outputs: an m'-byte ciphertext C (with $m' = 16 \cdot \left\lceil \frac{m}{16} \right\rceil$ for the 128-bit ciphers, replacing 16 by 8 for the 64-bit versions), and a 64 or 128-bit tag, i.e. $(C, \texttt{tag}) = \mathscr{E}_K^M(A, M)$. This algorithm is given in the form of pseudocode as Algorithm 1.1 on page 11.

In the tweak inputs, the value N is encoded on 96 bits, i and λ are encoded on 124 bits, and j, ℓ, m are encoded on 28 bits. Since the actual variables j, ℓ, and m are longer than 28 bits, when inserted in the tweak they are truncated to the 28 least significant bits.

Graphically, AD processing is represented in Figure 1.1 on the next page and message processing in Figure 1.2 on the following page, where the pictures are simplified by showing the algorithms for messages of less than 2^{28} blocks. (In Figure 1.3 on the next page we describe a different way to compute the final tag to encourage its study. It is not part of the submitted variants of Qameleon.)

Note that the NIST requirements ask "*The family shall include one primary member that has [...] a nonce length of at least 96 bits, [...]. The limits on the input sizes [...] for this member shall not be smaller than* $2^{50} - 1$ *bytes.*" Block counters for $2^{50} - 1$ bytes occupy 43 bits and therefore nonce and counter cannot be packed in 128 bits! Our solution is to replace the nonce every 2^{28} blocks as described in the next subsection.

1.2.2 Nonce rotation

We describe how the nonce is rotated – i.e. replaced with a mathematically unpredictable value after a certain amount of text is processed.

Let v be the original nonce passed as a parameter to the algorithm. This nonce is used to encrypt the first 2^{28} blocks, i.e. the first *chunk*: A *chunk* is understood to be each segment of up to 2^{28} texts encrypted with a given nonce v. After that, new nonces are derived from v using the formula $N = E_K^{1111\|v\|0}(j \gg 28)$ where j is the zero-based index of the message block, and since N is computed *before* the counter overflows into the 29-th bit, $j \gg 28$ is the zero-based index of the *chunk* of 2^{28} message blocks, minus 1.

Note that all chunks beside possibly the last one are full chunks.

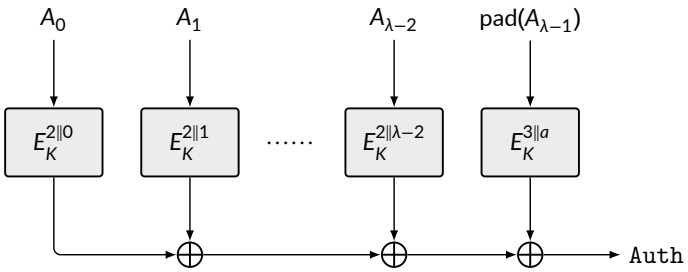

Figure 1.1: Processing of the AD

Here a is the byte length of A and λ the total number of blocks in which A is segmented. These blocks are all non-empty and the last one may be full or fractional. If the last block is fractional, it shall be padded before use.

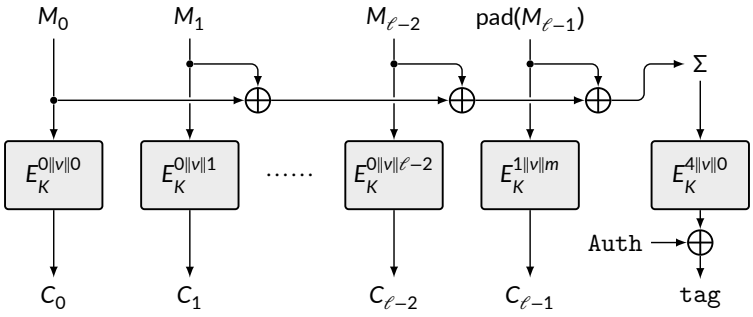

Figure 1.2: Message processing and tag computation

Here m is the byte length of M and ℓ is the number of blocks in which M is segmented: These blocks are all non-empty and the last one may be full or fractional. If the last block is fractional, it shall be padded before use.

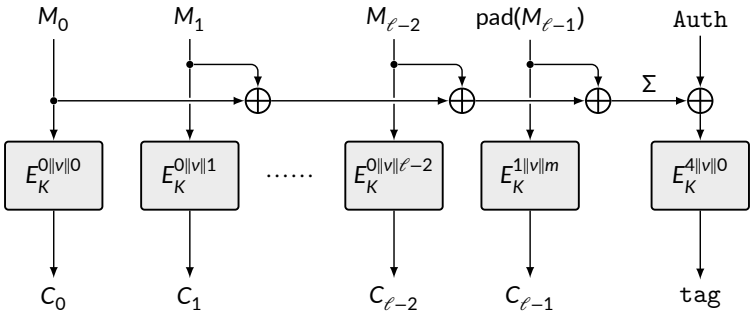

Figure 1.3: Alternative message processing and tag computation

This is a different approach to the computation of the final tag, whereby the AD authenticator `Auth` *is added to the plaintext checksum* Σ before *the latter is encrypted. This is not part of our submission and we share for only for possible study. It inherits all security proofs of OCB/ΘCB and the particular combination of the authenticator with the plaintext checksum seems different from the techniques considered by Rogaway. The downside of this variant is that parallelisation is reduced, and the final tag will have an added block cipher invocation in the critical path. This is not critical in all applications we considered so far.*

Tweak domain separation between the first and the subsequent chunks is guaranteed by flipping the most significant bit of the tweak after the first chunk.

In environments where the nonce is assured to be unique, in place of this solution the nonce can be directly updated, e.g. by using an untamperable counter. While such a scenario is discussed in Chapter 5 on page 29, the nonce rotation is mostly useful for systems which pick the nonce at random (and thus a 96-bit nonce offers good protection against nonce collisions, until about 2^{48} different nonces are used).

1.2.3 Decryption and tag verification

The verification/decryption algorithm $\mathscr{D}$ takes as input a variable-length ciphertext byte-string C (with $m' = |C|/8$), a variable-length AD byte-string A (with $a = |A|/8$), a tag $\mathtt{tag}$, a nonce N, and a $2n$-bit key K. Note that the receiver must know the correct plaintext length m, where $m' = 2\mu \cdot \lceil \frac{m}{2\mu} \rceil$ holds. This algorithm outputs either an error string $\perp$ to signify that the verification failed or an m'-byte string $M = \mathscr{D}_K^N(A, C, \mathtt{tag})$ when the tag $\mathtt{tag}$ is valid. The first m bytes of the m'-byte string M are the original plaintext. This algorithm is given in the form of pseudocode as Algorithm 1.2 on page 12.

1.2.4 Tag length truncation

PANORAmA can output 64- or 128-bit tags. In the following the tag length shall be denoted by t.

In the case where a cipher with 128-bit blocks is used, we need to define how to choose the 64 bits for the tag. We have defined truncation as taking the least significant bits, however for an "orderless" field such as the internal state of a cipher that does not use additions this must be explicitly defined.

Now, in agreement to what we shall define for the internal state of the cipher QARMA in (2.1), the 128-bit registers where $\mathtt{Auth}$ and Σ are accumulated, and finally $\mathtt{tag}$ is computed are divided in 8-bit cells indexed the same way the cipher state is partitioned (since it is a sum of final cipher states), e.g. $\mathtt{tag} = t_0 \| t_1 \| \cdots \| t_{14} \| t_{15}$. Then, the 64-bit tag is defined as $\lfloor \mathtt{tag} \rfloor_{64} = t_0 \| t_1 \| \cdots \| t_6 \| t_7$.

1.2.5 64-bit version

The version of PANORAmA for tweakable block ciphers with 64-bit blocks and tweaks is mostly identical with the 128-bit version just described, the biggest differences being that: The nonce length and the size of the nonce field in the tweak is now 44 bits; the counter field is 16 bits. The masks and shifts are thus adjusted accordingly, as well as the computation of m' from m.

If the tweakable block cipher has 64-bit blocks but it accepts 128-bit tweaks, no change to the nonce length is necessary and the fields are as in the 128-bit version.

Since QARMA-64 has 64-bit nonces, in Section 3.3 on page 22 we explain how to modify the algorithm in order to double or triple the tweak length. A technique for doing so was suggested in [Ava17, Section 2.9], but the method presented here requires less resources and is more secure.

1.2.6 An implementation note

Some variants of the submission append the original plaintext length to the ciphertext, in order to recover it in an unambiguous way. The rationale for this choice is explained in Section 6.1 on page 41.

Algorithm 1.1: The PANORAmA encryption algorithm $\mathscr{E}_K^v(A, M)$

Input: A variable-length plaintext byte-string M, with $m = |M|/8$, a variable-length associated-data byte-string A, a 96-bit nonce v, a 256-bit key K, and the tag length t

Output: A pair $(C, \texttt{tag})$ consisting of: a m'-byte ciphertext C, with $m' = 16 \cdot \left\lceil \frac{m}{16} \right\rceil$; and tag $\texttt{tag}$

// Initialisation

$N \leftarrow v$

$b \leftarrow 0$

// Processing the associated data

$\texttt{Auth} \leftarrow 0$

if $A \neq \epsilon$ **then**

  $a \leftarrow \dfrac{|A|}{8}$ (byte length of A)

  $A_0 \| A_1 \| \cdots \| A_{\lambda-1} \leftarrow A$ where each $|A_i| = 128$ for $0 \leq i < \lambda - 1$ and $0 < |A_{\lambda-1}| \leq 128$

  for $i \leftarrow 0$ **to** $\lambda - 2$ **do**

    $\texttt{Auth} \leftarrow \texttt{Auth} \oplus E_K^{0010 \| \lfloor i \rfloor_{124}}(A_i)$

  $\texttt{Auth} \leftarrow \texttt{Auth} \oplus E_K^{0011 \| \lfloor a \rfloor_{124}}(\text{pad}(A_{\lambda-1}))$

// Processing the message

$\Sigma \leftarrow 0$

if $M \neq \epsilon$ **then**

  $M_0 \| M_1 \| \cdots \| M_{\ell-1} \leftarrow M$ where each $|M_i| = 128$ for $0 \leq i < \ell - 1$ and $0 < |M_{\ell-1}| \leq 128$

  for $j \leftarrow 0$ **to** $\ell - 2$ **do**

    $\Sigma \leftarrow \Sigma \oplus M_j$

    $C_j \leftarrow E_K^{b000 \| N \| \lfloor j \rfloor_{28}}(M_j)$

    if $(j \wedge \texttt{0x0fffffff} = \texttt{0x0fffffff})$ **then**

      $N \leftarrow \left\lceil E_K^{1111 \| v \| 0^{28}}(j \gg 28) \right\rceil_{96}$

      $b \leftarrow 1$

  $\texttt{tmp} \leftarrow \text{pad}(M_{\ell-1})$

  $\Sigma \leftarrow \Sigma \oplus \texttt{tmp}$

  $C_{\ell-1} \leftarrow E_K^{b001 \| N \| \lfloor m \rfloor_{28}}(\texttt{tmp})$

// Tag generation and return

$\texttt{tag} \leftarrow \lfloor E_K^{b100 \| v \| 0}(\Sigma) \oplus \texttt{Auth} \rfloor_t$

return $(C_0 \| \ldots \| C_{\ell-1}, \texttt{tag})$

Algorithm 1.2: The PANORAmA decryption algorithm $\mathscr{D}_K^v(A, M)$

Input: A variable-length byte-string C, a variable-length associated-data byte-string A, a tag tag of length t, a 96-bit nonce v, a 256-bit key K, the plaintext byte length $m = |M|/8$

Output: Either an error string $\perp$ to signify that the verification failed or a m-byte string $M = \mathscr{D}_K^N(A, C, \texttt{tag})$ when the tag tag is valid.

// Initialisation

$N \leftarrow v$

$b \leftarrow 0$

$m' \leftarrow \dfrac{|C|}{8}$ (length in bytes)

if $16 \nmid m'$ **or** $m' \neq 16 \cdot \left\lceil \dfrac{m}{16} \right\rceil$ **then**

  return $(\perp)$

// Processing the associated data

$\texttt{Auth} \leftarrow 0$

if $A \neq \epsilon$ **then**

  $a \leftarrow \dfrac{|A|}{8}$ (byte length of A)

  $A_0 \| A_1 \| \cdots \| A_{\lambda-1} \leftarrow A$ where each $|A_i| = 128$ for $0 \leq i < \lambda - 1$ and $0 < |A_{\lambda-1}| \leq 128$

  for $i \leftarrow 0$ **to** $\lambda - 2$ **do**

    $\texttt{Auth} \leftarrow \texttt{Auth} \oplus E_K^{0010 \| \lfloor i \rfloor_{124}}(A_i)$

  $\texttt{Auth} \leftarrow \texttt{Auth} \oplus E_K^{0011 \| \lfloor a \rfloor_{124}}(\text{pad}(A_{\lambda-1}))$

// Processing the ciphertext

$\Sigma \leftarrow 0$

if $C \neq \epsilon$ **then**

  $C_0 \| C_1 \| \cdots \| C_{\ell-1} \leftarrow C$ where each $|C_i| = 128$

  for $j \leftarrow 0$ **to** $\ell - 2$ **do**

    $M_j \leftarrow D_K^{b000 \| N \| \lfloor j \rfloor_{28}}(C_j)$

    $\Sigma \leftarrow \Sigma \oplus M_j$

    if $(j \wedge \texttt{0x0fffffff} = \texttt{0x0fffffff})$ **then**

      $N \leftarrow \left\lceil E_K^{1111 \| v \| 0^{28}}(j \gg 28) \right\rceil_{96}$

      $b \leftarrow 1$

  $M_{\ell-1} \leftarrow D_K^{b001 \| N \| \lfloor m \rfloor_{28}}(C_{\ell-1})$

  $\Sigma \leftarrow \Sigma \oplus M_{\ell-1}$

  Truncate $M_{\ell-1}$ to m mod 16 bytes

// Tag verification and return

$\texttt{tag}' \leftarrow \lfloor E_K^{b100 \| v \| 0}(\Sigma) \oplus \texttt{Auth} \rfloor_t$

if $tag' \neq tag$ **then**

  return $(\perp)$

else

  return $(M_0 \| \ldots \| M_{\ell-1})$

2 Specification of the Tweakable Block Cipher QARMA

This chapter is mostly an abridged version of [Ava17, Section 2].

2.1 General definitions and notation

The overall scheme of the TBC QARMA is depicted in Figure 2.1 on the following page. There, and throughout this document, a bar over a function – e.g. $\bar{F}$ – denotes its inverse.

QARMA is a three-round Even-Mansour construction where the permutations are parameterized by a core key. The key mixings between rounds are derived from a whitening key. The first and third permutations are functionally the inverse of each other and are further parameterised by a tweak, a concept related to reflection ciphers [BCG+12a]. The central permutation is designed to be easily inverted by means of a simple transformation of its round key.

The cipher is depicted in more detail in Figure 2.2 on the next page.

The keys k_0, k_1, w_0, and w_1 are derived from a master key K via a simple *key specialisation*. The letters P, C and T denote the plaintext, the ciphertext and the tweak, respectively; S represents a layer of sixteen μ-bit S-Boxes, h and τ are permutations, M is an involutory MixColumns-like operation. and ω is a *Linear Feedback Shift Register (LFSR)*.

Ley $n = 16\mu$ with $\mu = 4$ or 8. All n-bit values are represented as arrays of sixteen μ-bit *cells*, which is also viewed as a 4×4 matrix. For instance, the internal state admits representations

$$\mathrm{IS} = s_0 \| s_1 \| \cdots \| s_{14} \| s_{15} = \begin{pmatrix} s_0 & s_1 & s_2 & s_3 \\ s_4 & s_5 & s_6 & s_7 \\ s_8 & s_9 & s_{10} & s_{11} \\ s_{12} & s_{13} & s_{14} & s_{15} \end{pmatrix} , \tag{2.1}$$

so that 4×4 matrices operate column-wise on these values by left multiplication. The bits of the state are stored little endian, for instance, for QARMA-64, bit 63 of the state is the most significant bit and bit 0 is the least significant one. For the storage of the bits in the *cells* we use the same numbering convention used for memory, intended as the ubiquitous little-endian (and used in MANTIS' definition), in other words the zeroth cell contains the most significant bits of the state. The plaintext is given as $P = p_0 \| p_1 \| \cdots \| p_{14} \| p_{15}$, the tweak as $T = t_0 \| t_1 \| \cdots \| t_{14} \| t_{15}$.

Throughout this chapter we use the symbol "+" to denote addition in all algebraic structures. In particular it denotes the exclusive or in the QARMA ciphers, which do not use modular addition. The symbol tk denotes a (round) *tweakey*, i.e. a value derived only from the key, the tweak, and the round constants.

2.2 Key specialisation

The $2n = 32\mu$-bit key K is split as $w_0 \| k_0$ where w_0 and k_0, the *whitening* and *core* keys, are 16μ bits each.

For encryption, put $w_1 = o(w_0)$ and $k_1 = k_0$, where the *orthomorphism* $o(\cdot)$ is defined as

$$o(x) := (x \ggg 1) + (x \gg (16\mu - 1)) ,$$

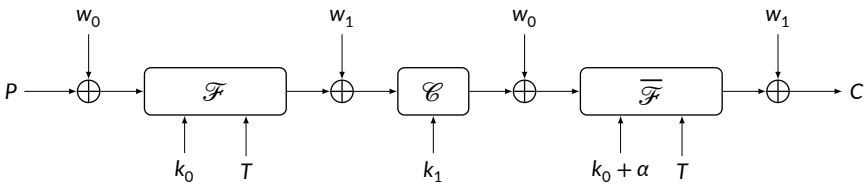

Figure 2.1: The Overall Scheme

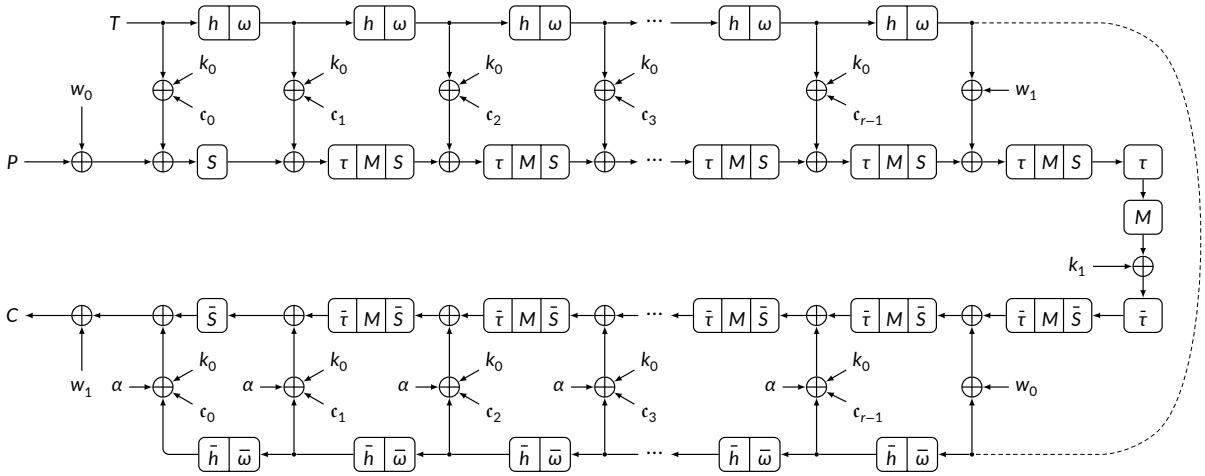

Figure 2.2: The Structure of $\texttt{QARMA}_r$

where $\gg$ denotes register shift to the right, and $\ggg$ circular rotation of the register's bits to the right.

Since the first r rounds of the cipher (ignoring initial whitening) differ from the last r rounds solely by the addition of a non-zero constant α, QARMA possesses a property very similar to PRINCE's α-reflection: The encryption circuit can be used for decryption when $k_0 + \alpha$ is used as the core key, the whitening keys w_0 with w_1 are swapped, and $k_1 = M \cdot k_0$.

2.3 The forward round function

The *Forward Round Function* $\mathscr{R}(\mathsf{IS}; \mathsf{tk})$ is composed by four operations, performed in the following order:

1. `AddRoundTweakey`. The round tweakey tk defined in § 2.10 on page 17 is XORed to IS.
2. `ShuffleCells`. $(\tau(\mathsf{IS}))_i = s_{\tau(i)}$ for $0 \leq i \leq 15$, where τ is the MIDORI cell permutation, i.e.

$$\tau = [\,0, 11, 6, 13, 10, 1, 12, 7, 5, 14, 3, 8, 15, 4, 9, 2\,] \ .$$

3. `MixColumns`. Each column of the cipher internal state array is multiplied by the involutory matrix M defined in Section 2.9 on page 16, i.e. $\mathsf{IS} = M \cdot \mathsf{IS}$.
4. `SubCells`. For the chosen S-Box σ, the S layer acts on the state as follows: $s_i \mapsto \sigma(s_i)$ for $0 \leq i \leq 15$. The S-Boxes are defined in §§ 2.7 and 2.8 on page 16.

A *short* version of the forward round function exists which omits the `ShuffleCells` and `MixColumns` operations, similarly to the AES final round.

After `AddRoundTweakey` the tweak T is updated by the function described next.

2.4 The tweak update function

First, the cells of the tweak are permuted as $h(T) = t_{h(0)} \| \cdots \| t_{h(15)}$, where h is the same permutation

$$h = [\, 6, 5, 14, 15, 0, 1, 2, 3, 7, 12, 13, 4, 8, 9, 10, 11 \,]$$

used in `MANTIS`. Then, the LFSR ω updates the tweak cells with indexes 0, 1, 3, 4, 8, 11, and 13. For $\mu = 4$, ω is a maximal period LFSR that maps cell (b_3, b_2, b_1, b_0) to $(b_0 + b_1, b_3, b_2, b_1)$. For $\mu = 8$, it maps cell $(b_7, b_6, \ldots, b_0)$ to $(b_0 + b_2, b_7, b_6, \ldots, b_1)$, and its cycles on the non-zero values have all length 15 or 30.

2.5 The backward round function

The *Backward Round Function* $\overline{\mathscr{R}}(\mathsf{IS}; \mathsf{tk})$ is the inverse of the forward round function $\mathscr{R}$.

Its *short* form omits `ShuffleCells` and `MixColumns`.

The inverse tweak update using the inverse LFSR $\bar{\omega}$ and the inverse permutation $\bar{h}$ must be applied before `AddRoundTweakey`.

2.6 The central construction and the pseudo-reflector

Two *central rounds* – a forward and a backward one – that use the whitening key instead of the core key, bracket the cipher's *Pseudo-Reflector* $\mathscr{P}(\mathsf{IS}; \mathsf{tk})$, which is essentially just a key addition and a matrix multiplication of the internal state. In more detail, this central construction is defined as follows:

1. A forward round $\mathscr{R}$.
2. The pseudo-reflector $\mathscr{P}(\mathsf{IS}; \mathsf{tk})$ i.e.
   (a) `ShuffleCells`.
   (b) Multiplication of the state by the involutory matrix M defined in Section 2.9 on the next page.
   (c) `AddRoundTweakey`. The round tweakey tk is XORed to the state.
   (d) Inverse `ShuffleCells`.
3. A backward round $\overline{\mathscr{R}}$.

Clearly, if steps (b) and (c) were swapped, then tk would have to be replaced with $M \cdot \mathsf{tk}$ to obtain the same function. Because of this, if tk is the tweakey used during encryption, $M \cdot \mathsf{tk}$ is used instead for decryption.

2.7 The 4-bit S-Box

With respect to [Ava17] here we restrict the choice of the S-Box to the one called there σ_1. Therefore, σ will be used as the 4-Bit S-Box. Its value table is:

$$\sigma_1 := [\, 10, 13, 14, 6, 15, 7, 3, 5, 9, 8, 0, 12, 11, 1, 2, 4 \,] \ .$$

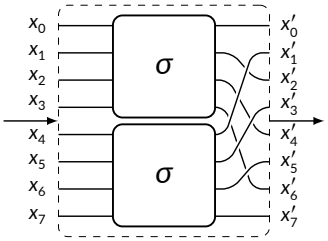

Figure 2.3: The Construction of the 8-bit S-Box Σ of QARMA-128

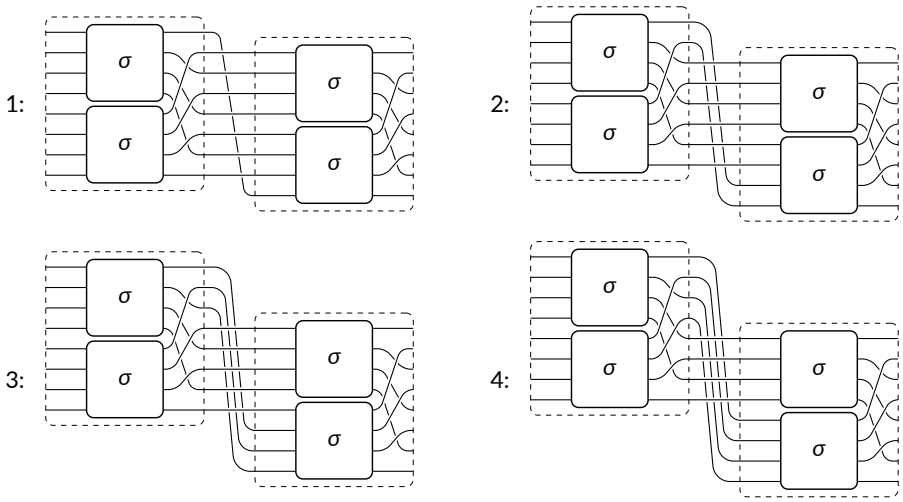

Figure 2.4: Alignment of Output and Input Bits of Consecutive Instances of the 8-bit Composite S-Box

2.8 The 8-bit S-Box

As in MIDORI-128 we construct an 8-bit S-Box Σ by placing two instances of a single 4-bit S-Box in parallel. However, we wire the input and output bits in a single and simpler, but asymmetric way, as shown in Figure 2.3. The S-Box σ is the default S-Box $\sigma = \sigma_1$ described in § 2.7 on the previous page. If we write a 8-bit cell of the state as $(x_7, x_6, x_5, x_4, x_3, x_2, x_1, x_0)$, σ is applied to (x_7, x_6, x_5, x_4) producing the output bits (x'_7, x'_5, x'_3, x'_1), and to (x_3, x_2, x_1, x_0) producing the output bits (x'_6, x'_4, x'_2, x'_0), and the output of the combined 8-bit S-Box is $(x'_7, x'_6, x'_5, x'_4, x'_3, x'_2, x'_1, x'_0)$. Since the construction is not symmetric, the opposite wiring must be implemented for Σ^{-1}.

A *three-round full diffusion property* as in MIDORI-128 (Theorem 1 in [BBI+15]) holds, namely *any input bit nonlinearly affects all 128 bits of the state after 3 full rounds* (i.e. not short rounds).

2.9 The diffusion matrices

QARMA's diffusion layer is composed of a cell permutation and of a matrix multiplication.

The chosen matrices are 4×4 Almost-*Minimum-Distance Separable (MDS)* matrices. Almost-MDS matrices are matrices whose branch number [DR02] is not optimal by 1, i.e. in this case they have a branch number of 4. This allows us to choose matrices whose implementation is much lighter than usual MDS matrices.

Let R_μ be the quotient ring $R_\mu = \mathbb{F}_2[X]/(X^\mu + 1)$, and ρ be the image of X in the ring R_μ. We see that $\rho^\mu = 1$, and thus such that $\{1, \rho, \rho^2, \ldots, \rho^{\mu-1}\}$ is a basis for R_μ as a $\mathbb{F}_2$-algebra. The multiplication by ρ is thus just a simple circular rotation of the bits (to the left), with only signal propagation latency. Matrices over R_μ allow

us to easily include rotations in the diffusion layer.

We can find Almost-MDS matrices over R_μ for $\mu = 4$, resp. $\mu = 8$ (for QARMA-64, resp. QARMA-128). Since R_μ contains zero divisors (for $\mu \geq 2$), care is to be taken when constructing invertible matrices. Theorem 1 of [Ava17] classifies which of the circulants of the form

$$M = \mathrm{circ}(0, \rho^a, \rho^b, \rho^c) = \begin{pmatrix} 0 & \rho^a & \rho^b & \rho^c \\ \rho^c & 0 & \rho^a & \rho^b \\ \rho^b & \rho^c & 0 & \rho^a \\ \rho^a & \rho^b & \rho^c & 0 \end{pmatrix} \tag{2.2}$$

are Almost-MDS, those with an Almost-MDS inverse of the same form, and the involutory ones. According to mathematical and heuristic criteria, in [Ava17] the following involutory matrices are selected:

1. For QARMA-64 ($\mu = 4$) the chosen matrix is:

$$M_{4,2} := \mathrm{circ}(0, \rho, \rho^2, \rho) \ .$$

2. For QARMA-128 ($\mu = 8$) the chosen matrix is:

$$M_{8,2} := \mathrm{circ}(0, \rho, \rho^4, \rho^5) \ .$$

2.10 The encryption and decryption algorithm

The encryption algorithm of $\mathtt{QARMA}_r$ is given in Figure 2.1 on the following page. $\mathtt{QARMA}_r$ has $2r+2$ rounds.

The round constants are derived from the expansion of the constant π. For the 64-bit version of QARMA we replace the first block of sixteen digits of the fractional part with zeros and select the seventh block as the α constant, as shown in Table 2.1 on the next page – as a *hommage* to PRINCE. For the 128-bit cipher, instead, we just take the first block of 128 bits in the fractional part of π as the α constant, set $c_0 = 0$, and then each c_i is a successive block 128 bits of π, as shown in Table 2.2 on the following page.

Note the constant α is always added to the last r backward rounds. This and the pseudo-reflector design prevent perfect symmetry in the data obfuscation path.

Algorithm 2.1: The QARMA Algorithm

Input: A $n = 64$ or 128 bit block P, a $2n$-bit key K, a n-bit tweak T, a flag f that can take the values *encrypting* or *decrypting*

Output: A n-bit block C

// Key specialisation

$w_0 \| k_0 \leftarrow K$

if $(f = \textit{encrypting})$ **then**

  $w_1 \leftarrow o(w_0),\ k_1 \leftarrow k_0$

else

  // $f =$ *decrypting*

  $w_1 \leftarrow w_0,\ w_0 \leftarrow o(w_0),\ k_1 \leftarrow M \cdot k_0,\ k_0 \leftarrow k_0 + \alpha$

// Forward rounds

$\mathsf{IS} \leftarrow P + w_0$

for $i \leftarrow 0$ *to* $r - 1$ **do**

  $\mathsf{IS} \leftarrow \mathscr{R}(\mathsf{IS}, k_0 + T + c_i)$ (short round for $i = 0$)

  $T \leftarrow \omega \circ h(T)$

// Central construction

$\mathsf{IS} \leftarrow \mathscr{R}(\mathsf{IS}, w_1 + T)$

$\mathsf{IS} \leftarrow \mathscr{P}(\mathsf{IS}, k_1)$

$\mathsf{IS} \leftarrow \overline{\mathscr{R}}(\mathsf{IS}, w_0 + T)$

// Backward rounds

for $i \leftarrow r - 1$ *down to* 0 **do**

  $T \leftarrow \bar{h} \circ \bar{\omega}(T)$

  $\mathsf{IS} \leftarrow \overline{\mathscr{R}}(\mathsf{IS}, k_0 + T + c_i + \alpha)$ (short round for $i = 0$)

$C \leftarrow \mathsf{IS} + w_1$

return (C)

Table 2.1: The Round Constants for the 64-bit Ciphers

α = C0AC29B7C97C50DD	c_0 = 0000000000000000	c_1 = 13198A2E03707344
c_2 = A4093822299F31D0	c_3 = 082EFA98EC4E6C89	c_4 = 452821E638D01377
c_5 = BE5466CF34E90C6C	c_6 = 3F84D5B5B5470917	c_7 = 9216D5D98979FB1B
c_8 = D1310BA698DFB5AC	c_9 = 2FFD72DBD01ADFB7	c_{10} = B8E1AFED6A267E96

Table 2.2: The Round Constants for the 128-bit Ciphers

α = 243F6A8885A308D3 13198A2E03707344	c_0 = 0000000000000000 0000000000000000
c_1 = A4093822299F31D0 082EFA98EC4E6C89	c_2 = 452821E638D01377 BE5466CF34E90C6C
c_3 = C0AC29B7C97C50DD 3F84D5B5B5470917	c_4 = 9216D5D98979FB1B D1310BA698DFB5AC
c_5 = 2FFD72DBD01ADFB7 B8E1AFED6A267E96	c_6 = BA7C9045F12C7F99 24A19947B3916CF7
c_7 = 0801F2E2858EFC16 636920D871574E69	c_8 = A458FEA3F4933D7E 0D95748F728EB658
c_9 = 718BCD5882154AEE 7B54A41DC25A59B5	c_{10} = 9C30D5392AF26013 C5D1B023286085F0
c_{11} = CA417918B8DB38EF 8E79DCB0603A180E	c_{12} = 6C9E0E8BB01E8A3E D71577C1BD314B27
c_{13} = 78AF2FDA55605C60 E65525F3AA55AB94	c_{14} = 5748986263E81440 55CA396A2AAB10B6
c_{15} = B4CC5C341141E8CE A15486AF7C72E993	c_{16} = B3EE1411636FBC2A 2BA9C55D741831F6

3 Parameter Sets and Variants, and Security Claims

3.1 Parameter sets and variants

The following parameter sets and variants are part of the submission. The names follow the following convention:

$$\texttt{qameleon}\|\texttt{blocksize}\|\texttt{taglength}\|\texttt{noncelength}\|\texttt{purpose}\|\texttt{version}$$

where the purpose field can be `gp` for "general purpose" or `me` for "memory encryption," and the nonce length can be `nn` if no nonce input is available ("no nonce") and `tc` is added if tweak compression is used.

(A) `qameleon12812896gpv1`

`Qameleon` instantiated with `QARMA-128`$_{14}$ with a 128-bit tag.

This is the primary member of the submission.

(B) `qameleon128128128tcgpv1`

`Qameleon` instantiated with `QARMA-128`$_{11}$ using tweak compression, a 128-bit tag, 128-bit nonces and no nonce rotation.

The tweak is a 256-bit long-tweak and it is compressed to 128 bit as described in Section 3.3, and in particular in Section 3.3.3 on page 24, before being used in the data obfuscation path.

128 bits of the long-tweak field are the nonce and they are processed through $g_0(\cdot)$. The 4 most significant bits of the other half of the long-tweak, which is processed through $g_1(\cdot)$, serve the tweak domain separation, and the remaining 120 bits offer plenty of room for the block counters.

(C) `qameleon12812864gpv1`

`Qameleon` instantiated with `QARMA-128`$_{14}$ with a 128-bit tag, modified with 64-bit nonces and 60-bit counters in the field of the tweak input, and no nonce rotation mechanism.

The ordering for the tweak fields is $\texttt{typefield}_4\|\texttt{counter}_{60}\|\texttt{nonce}_{64}$.

(D) `qameleon1286464mev1`

`Qameleon` instantiated with `QARMA-128`$_{11}$ with a 64-bit tag for *Random Access Memory (RAM)* encryption applications.

This mode is simplified by removing the processing of AD and using fixed length messages corresponding to *Memory Granules (MGs)*, which in turn correspond to one or more last level *Cache Lines (CLs)*. Typical lengths for a MG are 64 or 128 bytes, rarely 256 bytes.

In the `PANORAmA` mode of operations this variant employs tweak fields of 64 bits for the nonce (ideally a counter) and a 60 bit field for a *Physical Address (PA)*; Since the addresses are the bases of the location of 16-byte aligned blocks in the memory, 60 bits suffice to cover even a flat 64-bit address space.

(E) `qameleon6464tcmev1`

`Qameleon` instantiated with `QARMA-64`$_7$, using a 64-bit tag, for memory encryption applications.

This mode is simplified by removing the processing of AD and using fixed length messages corresponding to MGs, as in Parameter Set (D).

In `PANORAmA` this variant employs a long-tweak of 128 bits, with fields of 64 bits for the nonce (ideally a counter), a 4-bit tweak domain separation field, and a 60-bit field for a PA; In this case the

PA field can cover a flat address space of up to 63 bits.

This long-tweak input for QARMA-64 is compressed from 128 to 64 bits as described in Section 3.3, and in particular in § 3.3.3 on page 24, before being used in the data obfuscation path. The nonce is processed through $g_0(\cdot)$. The 4 most significant bits of the other half of the long-tweak, which is processed through $g_1(\cdot)$, serve the tweak domain separation, and the remaining 60 bits contain the physical address.

(F) `qameleon6464nnmev1`

Qameleon-64 with QARMA-64_9 and 64-bit tag, for memory encryption applications in the case the memory is on the same die or on the same package with tampering detection capabilities.

This mode is simplified by removing the processing of AD and using fixed length messages corresponding to memory granules, as in Parameter Sets (D) and (E).

The user may choose to ignore the authentication tag to obtain a pure confidentiality method, or store the tag in memory in order to defend against Rowhammer attacks [KDK+14, SD16] that can circumvent ECC protection [CRGB19].

We shall argue in Chapter 5 on page 29 that these parameters offer sufficient security margins.

3.2 Security claims

3.2.1 Security goals

The authenticated cipher is designed to protect confidentiality of plaintexts (under adaptive chosen-plaintext attacks) and integrity of ciphertexts (under adaptive forgery attempts).

See Section 3.2.3 for quantitative goals for specific parameter sets.

3.2.2 Expected strength in general

Each 128-bit cipher call is assumed to handle at most 2^{50} bytes of plaintext or AD. Each key is assumed to be used to process at most 2^{50} bytes. For 64-bit cipher calls these bounds are lowered to 2^{40}.

The legitimate key holder must not use the same nonce to encrypt two different (plaintext, AD) pairs under the same key. The cipher may lose integrity and confidentiality if this rule is violated. Therefore all security claims below are in the nonce-respecting setting.

Keys are assumed to be chosen independently and uniformly at random.

Qameleon is designed to protect confidentiality of plaintexts (under adaptive chosen-plaintext attacks) and integrity of ciphertexts (under adaptive forgery attempts). Quantitative goals for specific parameter sets are given in the next subsection.

Following [JNPS16], we claim that for any of the proposed variants, given a k-bit key, n-bit block, and t-bit tweak, the security offered against nonce-respecting adversary is: n-bit[1] against confidentiality of the plaintext; and n-bit security for the integrities of both the plaintext and the associated data. With respect to key recovery attacks, following the analysis of Section 5.2:

- Attacks using 2^{50} bytes of data would require more than 2^{224} computations to break the plaintext confidentiality of QARMA-128; and

[1] We note that actually, the correct bound is $\min(n, k)$, but for our purposes, $k > n$, and thus, the bound holds.

- Attacks using 2^{40} bytes of data would require more than 2^{112} computations to break the plaintext confidentiality of QARMA-64.

3.2.3 Expected strength for each parameter set

(A) `qameleon12812896gpv1`

Following the use of the PANORAmA mode of operation: the confidentiality of the plaintext is guaranteed against any nonce respecting adversary that satisfies the amount of data queries allowed. Moreover, any attack on the confidentiality by such an adversary by recovering the key would take more than 2^{224} time.

We note that integrity attacks would also have a success rate of 2^{-128} (as long as the key was not recovered).

These claims are the outcome of [KR11, Lemma 2].

(B) `qameleon128128128tcgpv1`

If the adversary probe multiple tweaks, and no two different tweaks collide, then the same security claims hold as for Parameter Set (A).

As the analysis of Section 5.4 suggests that tweak collisions happen with (close) to the random probability. Thus, before using about 2^{64} different tweaks, we do not expect any security degradation.

To conclude, the security claims for this set are equal to that of Set (A) with probability $O(t^2/2^{128})$. We note that Section 5.4.2 on page 38 discusses the impact of such collisions and shows that they do not impact the security (besides a very limited forgery attack, which requires more data than allowed for a single-key).

(C) `qameleon12812864gpv1`

Same as for Parameter Set (A).

(D) `qameleon1286464mev1`

Confidentiality is guaranteed independently of the number of blocks of data in encryption/decryption queries made by an adversary.

The success probability of single-key attacks against integrity is at most 2^{-256}.

For memory encryption applications, where the nonce is a counter or a split counter set (cf. Section 5.1.2 on page 30) that is not under adversarial control (but can be observed), forgery and integrity violations have a likelihood of 2^{-64}.

(E) `qameleon6464tcmev1`

Confidentiality is guaranteed independently of the number of blocks of data in encryption/decryption queries made by an adversary.

The success probability of single-key attacks against integrity is at most 2^{-128}.

For memory encryption applications, where the nonce is a counter or a split counter set (cf. Section 5.1.2 on page 30) that is not under adversarial control (but can be observed), forgery and integrity violations have a likelihood of 2^{-64}.

(F) `qameleon6464nnmev1`

In the intended scenario integrity and forgery are out of scope.

We claim that this variant offers full security against adversaries that can observe 2^{40} bytes of ciphertexts, which may be the encryptions of chosen plaintexts, where each 8-byte block has been encrypted using a different tweak (the PA of the block). In this case confidentiality is guaranteed,

and any attack will require at least the computational effort of 2^{112} encryptions.

3.3 Long tweak support

In some cases, one may be interested in having a longer tweak than n bits. We offer support for such long tweaks by relying on the existing QARMA-n instance which accepts n-bit tweaks. Hence, our solution is based on first compressing the long tweak T^{ℓ} into an n-bit tweak $T_{\text{effective}}$. In this proposal we discuss $2\,n$-bit tweaks and $3\,n$-bit tweaks, but one can use similar methodology to define support for $t\,n$-bit tweaks for any integer $t > 1$.

We shall treat T^{ℓ} as composed of two (or three) n-bit words, i.e., $T^{\ell} = T_1 \| T_0$ for $2n$-bit tweaks or $T^{\ell} = T_2 \| T_1 \| T_0$. Now, as we compress the tweak into n-bit one, we expect collisions to exist. A few requirements of the tweak compression process are self-explanatory. Most notably, we wish for collisions to exist with their natural probability and that such collisions will not yield meaningful information about the key. In particular, distinguishability should not be affected by tweak compression.

3.3.1 Specification of the tweak compression function

The compression of the long tweak into a shorter one is done by keyed functions.

For $n = 64$ we define $g_0(\cdot)$, $g_1(\cdot)$ and $g_2(\cdot)$ to be 4-round (forward full rounds only) QARMA-64, i.e., $\mathscr{R}^4$ with key K_1 and round constants listed in Table 3.1. For $n = 128$, $g_0(\cdot)$ and $g_1(\cdot)$ are defined as 5-round (forward full rounds only) QARMA-128. In Table 3.2 on the next page we give the corresponding round constants. In § 4.2.4 on page 28 we describe how these constants were chosen.

For $2\,n$-bit tweaks, we compute $T_{\text{effective}}$ as follows:

$$T_{\text{effective}} = g_0(T_0 \oplus K_0) \oplus g_1(T_1 \oplus K_0) \ ,$$

where the $2\,n$-bit key K is considered as two n-bit words, i.e., $K = K_1 \| K_0$.

For $3\,n$-bit tweaks, compute $T_{\text{effective}}$ as follows:

$$T_{\text{effective}} = g_0(T_0 \oplus K_0) \oplus g_1(T_1 \oplus K_0) \oplus g_2(T_2 \oplus K_0) \ .$$

The formal definition of g_i is given in Algorithm 3.1 on the next page.

3.3.2 Usage of the tweak compression function

Not only the tweak compression function is used to provide a longer tweak to the QARMA encryption function, but the summands it returns are added to the plaintexts and ciphertextes as well to further differentiate the encryption functions in case different long tweaks are mapped onto the same short value.

In the case of of $2\,n$-bit long tweaks, a new algorithm QARMA^{ℓ} is defined an as extension of Algorithm 2.1 as follows: encryption is defined as

$$P \mapsto E_K^{T_{\text{effective}}}\left(P \oplus \sigma^2(g_0(T_0 \oplus K_0))\right) \tag{3.1}$$

and accordingly, decryption as

$$C \mapsto D_K^{T_{\text{effective}}}(C) \oplus \sigma^2(g_0(T_0 \oplus K_0)) \ . \tag{3.2}$$

Algorithm 3.1: Tweak Compression Algorithm

Input: A $2n$- or a $3n$-bit tweak T, a $2n$-bit key K

Output: The summands of n-bit effective tweak $T_{\text{effective}}$, given as (γ_1, γ_1) or $(\gamma_0, \gamma_1, \gamma_2)$

$K_1 \| K_0 \leftarrow K$
$s \leftarrow 3$ **or** 4 depending on whether $n = 64$ **or** 128
if $|T| = 2n$ **then**
    $T_1 \| T_0 \leftarrow T$
else
    $T_2 \| T_1 \| T_0 \leftarrow T$

// First branch ($g_0(\cdot)$)
$\gamma_0 \leftarrow T_0 \oplus K_0$
for $i \leftarrow 0$ **to** s **do**
    $\gamma_0 \leftarrow \mathscr{R}(\gamma_0, K_1 + \rho_i^0)$

// Second branch ($g_1(\cdot)$)
$\gamma_1 \leftarrow T_1 \oplus K_0$
for $i \leftarrow 0$ **to** s **do**
    $\gamma_1 \leftarrow \mathscr{R}(\gamma_1, K_1 + \rho_i^1)$

// Third branch ($g_2(\cdot)$)
if $|T| = 3n$ **then**
    $\gamma_2 \leftarrow T_2 \oplus K_0$
    for $i \leftarrow 0$ **to** s **do**
        $\gamma_2 \leftarrow \mathscr{R}(\gamma_2, K_1 + \rho_i^2)$

if $|T| = 2n$ **then**
    return (γ_0, γ_1)
else
    return $(\gamma_0, \gamma_1, \gamma_2)$

Table 3.1: Round constants used in the tweak compression of $2n$-bit and $3n$-bit tweaks for $n = 64$

ρ_0^0 = 0000000000000000	ρ_0^1 = 2B9BA3E381D6DB63	ρ_0^2 = 96E5805273530DCD
ρ_1^0 = 5CA791F11E2F4F7F	ρ_1^1 = EA115044D015CC95	ρ_1^2 = 84876D4BF872E9C7
ρ_2^0 = C9BDB65CF6E990C3	ρ_2^1 = 87E51BB37364EAC6	ρ_2^2 = 6C6255510F188A26
ρ_3^0 = B4001C30BE9DE6F1	ρ_3^1 = ED24D4F5E058BB1C	ρ_3^2 = 5C80839058071EA3

Table 3.2: Round constants used in the tweak compression of $2n$-bit tweaks for $n = 128$

ρ_0^0 = 0000000000000000 0000000000000000	ρ_0^1 = B8D1D517B635D231 A1C53B7F11F69713
ρ_1^0 = 5CA791F11E2F4F7F C9BDB65CF6E990C3	ρ_1^1 = C9000CCDDE4DE79C FB224FF9C51E4E18
ρ_2^0 = B4001C30BE9DE6F1 2B9BA3E381D6DB63	ρ_2^1 = 58B7ADDCE9450428 635BAAF34C104AD0
ρ_3^0 = 96E5805273530DCD E0FE5C2707D8E275	ρ_3^1 = D2EB1B1F7E5C9F00 5376A18E922FC57B
ρ_4^0 = 2A76024E015E6630 5A501202D8CD0F57	ρ_4^1 = FE93002568C93BA5 529F4E65C7740E17

In the case of of 3 n-bit long tweaks, the extension $\texttt{QARMA}^{\ell}$ is as follows: encryption is defined as

$$P \mapsto E_K^{T_{\text{effective}}}\left(P \oplus o^2(g_0(T_0 \oplus K_0))\right) \oplus o^2(g_1(T_1 \oplus K_0)) \tag{3.3}$$

and accordingly, decryption as

$$C \mapsto D_K^{T_{\text{effective}}}\left(C \oplus o^2(g_1(T_1 \oplus K_0))\right) \oplus o^2(g_0(T_0 \oplus K_0)) \ . \tag{3.4}$$

Note that if $o(\cdot)$ is an orthomorphism, then $o^2(\cdot)$ is an orthomorphism as well[2]. We use this simple fact to use a different orthomorphism in the tweak compression function than in the TBC proper.

The security analysis of this construction is given in Section 5.4.

3.3.3 Usage of the tweak compression function in Parameter Sets (B) and (E)

The PA and the tweak differentiation fields should be packed together, thus constituting a 64 bit field or a 128 bit field, that is to be processed by the tweak compression function $g_0(\cdot)$. The nonce (which is itself a counter or a set of counters coming from the integrity structures – not to be confused with the "block index" counter of the other variants), which is constant for an entire MG, is then processed by $g_1(\cdot)$ only once per MG update, providing further power savings.

[2] 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. This proves $o^2(\cdot)$ is an orthomorphism.

4 Design Rationale

NIST's call for AES submissions identified "*the extent to which the algorithm output is indistinguishable from [the output of] a [uniform] random permutation*" as one of the "*most important*" factors in evaluating candidates. This factor has been amply studied for wide-trail ciphers, and QARMA follows the state of the art on the construction of this type of block ciphers. So we can safely presume that QARMA-*n* outputs for distinct inputs are indistinguishable from independent uniform random strings when the number of inputs does not approach $2^{n/2}$ – for a single key and tweak.

For the 128-bit version of the cipher, with the recommended 12-byte nonces (or even for shorter 8-byte nonces) and lengths of up to 2^{50} blocks, the outputs are thus indistinguishable from independent uniform random strings, and in fact the bounds for indistinguishability are higher (this includes the use of nonce rotation, since and rotated nonce collision requires more than $2^{28+48} = 2^{76}$ blocks to be processed – more analysis in given in Section 5.4). The encryption of P into C is thus indistinguishable from a one-time pad, and the tag is indistinguishable from an independent one-time polynomial authenticator of C.

4.1 High level choices

The main application for which QARMA, Qameleon and their variants have been architected is the encryption of memory contents for both live memory (i.e. stored in RAM), where latency is critical and affects overall performance directly, or memory contents at rest. The latter may soon include very low latency *Storage Class Memory (SCM)* and therefore the speed of the ciphering algorithm matters also in that case (note however that exporting to memory may require longer hashes than 64- or 128-bit tags).

In this context the given architecture is usually implemented in a fully unrolled and pipelined way, where, once an *initial latency* corresponding to the design's critical path has passed, the design can produce new ciphertext continuously keeping up with the memory's bandwidth. *What we found is that the resulting performance penalty is only correlated to the initial latency, and this relation is mostly linear.*

Therefore our first priority is the reduction of the critical path for the message encryption part in a hardware implementation, followed by energy consumption, and only then we consider area, SW performance and code size.

The other main priority is to have a design that allows also for great parallelism. For memory encryption in theory there is often no need to have as many cipher blocks as they fit in a cache line, provided that the cipher can keep up with the memory bandwidth in a pipelined implementation, but there are at least two situations where this can be very useful:

(i) When the memory interface is particularly fast (for instance in some *High-Performance Computing (HPC)* or enterprise server applications); and

(ii) When the constraints required by a fine grained pipelined implementation end up increasing the initial latency too much. In this case it may then be advisable to opt for a more monolithic, or less pipelined, and optimised implementation – which can require a lower clocking speed of the circuit. Then we fall in the previous case and this must compensated with parallelism in order to get the required throughput.

Algorithms that produce a pad stream that is then XORed to the plaintext have the advantage of very low latency, but they require either a long initialisation phase and an excellent source of randomness to be

secure. When they are based on stream cipher constructions they are also cumbersome to make them parallelisable.

Therefore we prefer methods that, while having higher latency, place the data obfuscation path though the cryptographic primitive itself, also guaranteeing in-block diffusion, which is often explicitly requested by industrial partners (the same type of requirements led for instance to the design of PRINCE, together with the ability to use it without temporal uniqueness).

For the purpose of architecting a secure solution for memory encryption we have considered various approaches, but since block ciphers are, arguably, still the best understood cryptographic primitives, we have decided to follow the classical approach of a mode of operation together with a self contained block cipher.

We have then reviewed the various suitable modes of operation and found most of the modes based on classical (i.e. non-tweakable) block ciphers be inefficient, as they put two instances of the block cipher in the critical path, at least for the first block in a message. On the other hand, research on *tweakable* block ciphers [LRW02] has been motivated to solve exactly this problem for specific applications, such as disk [Mar10] and memory encryption [HT13].

Therefore we decided to have a "tweaked electronic codebook" mode as our starting point. Under this assumption QARMA was developed and presented in [Ava17], and the associated mode of operation PANORAmA has been "assembled" from established constructions to be used with it.

Despite the original motivations, Qameleon is a general purpose AEAD algorithm and can be used for any other application. Its low initial latency, absence of initialisation phase that thus maintains a high performance also for small inputs, and high parallelism make it suitable for most applications.

4.2 Low level choices

For both QARMA and Qameleon, our main concern was that of taking conservative steps along established design methodologies. So, we never decided to take shortcuts and we often accepted "suboptimal" performance by choosing components that are, or have proven, to be stronger over smaller or more efficient ones. We did not aim at breaking world records in implementation compactness, latency, code size, or power consumption. In fact, it could be claimed that some of the outstanding performance indicators in Chapter 6 on page 41 are achieved *despite* the conservativeness of our choices. Whenever possible, we also have made choices that would make implementations less error prone.

4.2.1 Low level choices for QARMA

(i) Better S-Box and diffusion layer than in MIDORI and MANTIS. The S-Box has been chosen exclusively for its cryptographic properties, and we did not make any attempt at relaxing them in order to balance this later with the choice of diffusion layers. The search process for the S-Box has been sketched in [Ava17], but we decided to describe it thoroughly in the next subsection providing many details missing in the original QARMA paper. The quality of the S-Box has been validated independently by the analysis made by Eichlseder in her Ph.D. Thesis [Eic18].

(ii) Better central construction than the reflector in PRINCE or MANTIS.

(iii) W.r.t. MANTIS an LFSR ω is added to the tweak schedule to further disrupt characteristics.

(iv) The round constants in QARMA have been chosen as "noting up my sleeve" numbers. They are derived from the binary expansion of π. For QARMA-64 we used the same ordering as in PRINCE. For the QARMA-128 we have taken the blocks in the same order as they are found in the expansion of π.

4.2.2 The selection of the QARMA S-Box

With respect to [Ava17] we provide here additional information about the selection of the S-Box σ_1. We reconstructed the selection process, including details which were omitted in the original paper:

1. We generate all fixed-point free involutions on the set [0..15] using Prissette's algorithm [Pri10].
2. For each such involution, we check whether it satisfies certain cryptographic properties.
   In particular we require that:
   (a) The maximal bias of a linear approximation (over $\mathbb{F}_{2^4}$) shall be minimal (1/4) and the number of linear approximations with this bias shall be smallest (30).
   (b) The maximal probability of a differential characteristic shall be minimal (1/4) and the number of differential characteristics with this likelihood shall be smallest (15).
   (c) Each input bit of the S-Box shall influence each output bit non-linearly.
   (d) Algebraic immunity shall be 2.
   (e) Each of the 15 non-zero component functions shall have algebraic degree 3.
   (f) We compute the *Sum of Products (SOP)* and NOT-SOP of each output bit using the Quine-McCluskey algorithm [Qui52, McC56]. We shall consider only those S-Boxes whose output bits can be all expressed as sums of at most four products, each one having at most weight three – for both the SOP and the NOT-SOP. This is done in order to favour those that can be implemented with minimal length critical path.
   (g) Additionally, we minimise both the sum of the degrees and of the weights in the SOP (and of the NOT-SOP), subject to all above constraints. These minima are 10 and 16 respectively.
   (h) At this point we are left with 332 S-Boxes. We further reduce the size of the resulting set by minimising the maximal likelihood for an *arithmetic* differential (i.e. over $\mathbb{Z}/16\mathbb{Z}$) as well.
3. By doing this, we reduced the set to 60 choices. *These 60 S-Boxes are all affine equivalent to each other* – in fact, this is true of all 332 S-Boxes found without restriction (h). We have re-verified this for the present submission using the tool `sboxU` by Perrin [Per19] running under SageMath 8.6 [Sage19].
4. We picked the second S-Box of the resulting list because, synthesising the full QARMA-64 with the first few S-Boxes in the list, it resulted in a minimally smaller area. This was done using the synthesis tools which were available to us at the time when QARMA was first developed. We do no longer have access to those tools, but since all the S-Boxes are affine equivalent to each other, we believe the actual choice is not a concern.

With respect to the classification of optimal 4-bit S-Boxes in linear equivalence classes from [LP07, Table 6], our S-Boxes are linearly equivalent to G_4. These S-Boxes all have boomerang uniformity [CHP+18] equal to 10 which, while not optimal [BC18], is a reasonable value for a 4-bit S-Box.

The small `C` program that generates the 60 S-Boxes accompanies the submission.

4.2.3 Low level choices for the PANORAmA mode of operation

(i) We do not handle the encryption of a final fractional block by adding it to a pad and then truncating the result. This simplifies implementation and removes one critical component that has often, in many a design, led to weaknesses. The resulting increase of ciphertext size, esp. for small message, is in our opinion, a small price to pay.

(ii) We systematically applied tweak separation through a bit field for all components of the algorithm.

(iii) The use of nonce rotation can be viewed as controversial, esp. since there are established alternatives, such as hashing the public value and truncating the result before using it, as in GCM [MV04]. Another alternative, used in Deoxys-II [JNPS16] when a nonce (public value) is longer than 120 bits and up to 128 bits, consists in encrypting the nonce and then truncate the resulting ciphertext to 120 bits to be used as the actual nonce. This is significantly faster than a full hash because it adds only one encryption to the critical path. Similarly, we could have used a round of encryption to compress a 96-bit or longer nonce to a shorter field, but the resulting additional latency increase would have been unacceptable for the intended applications. The chosen approach adds a single encryption only after a significant amount of data has been processed, does not increase initial latency, does not cause accidental nonce collisions at the beginning, and allows the processing of 2^{76} (expected) blocks before a collision may occur, which is way beyond the NIST minimal requirements. Furthermore, those collisions do not seem exploitable to compromise confidentiality or integrity, but only to establish distinguishability from a random output.

In essence, PANORAmA is a subset of the OCB/ΘCB modes, therefore inheriting their security proofs.

4.2.4 Low level choices for the tweak compression

The round constants for the tweak compression algorithm have been selected as follows:

(i) First, we generated a pseudorandom sequence of 64-bit values using the ISAAC-64 random generator [Jen93, Jen96]. We used the parameter RANDSIZL = 8 and initialised the randrsl buffer with the first 2048 characters of the introduction of the NIST all [NIS18], including (single) spacing and punctuation: *"The deployment of small computing devices such as RFID tags, industrial controllers, sensor nodes and [...]."*

(ii) Since we only need to determine the XOR differences of the round constants for $g_1(\cdot)$ and $g_2(\cdot)$ with respect to the constants of $g_0(\cdot)$, and the constants for the first round can be chosen arbitrarily, successive groups of 3 (resp. 8) constants are picked from the resulting stream produced by ISAAC-64.

(iii) A bit-wise *Mixed Integer-Linear Programming (MILP)* program is used to count the active S-Boxes.

(iv) For the 128-bit wide tweak compression, the smaller 4-bit S-Boxes are counted, and not the larger composite 8-Bit S-Boxes, in order to get a more precise estimate for the linear and differential biases.

(v) In the 64-bit case, the constants for $g_1(\cdot)$ come from positions $3 \cdot 213$ to $3 \cdot 213 + 2$ in the pseudorandom stream, and the constants for $g_2(\cdot)$ come from positions $3 \cdot 307$ to $3 \cdot 307 + 2$. These are the lowest indices that gave us the desired number of active S-boxes, which is 31 between $g_0(\cdot)$ and $g_1(\cdot)$ and at least 28 in the other two combinations.

(vi) For the 128-bit case we were lucky with the first 4 constants (the first 8 words in the stream). Between $g_0(\cdot)$ and $g_1(\cdot)$ we get at least 59 active S-Boxes.

(vii) Note that the constants for the first round can be chosen arbitrarily. We set the first one, ρ_0^0, to zero and the other ones, i.e. ρ_0^i for $i > 0$ together with the constants ρ_j^0, $j > 0$ as described next.

(viii) Similarly, the differences of the constants starting from the second round count, not the constants themselves. This means that the constants for $g_0(\cdot)$ can be chosen arbitrarily. Instead of setting them to zero, we have picked other constants from the 314th position in the sequence generated by ISAAC-64 in order to differentiate the rounds, first ρ_j^0, $j > 0$, then ρ_0^i for $i > 0$.

5 Security Analysis

5.1 On the threat models

`Qameleon` targets low-latency scenarios, such as memory encryption. In many of these scenarios, the operating system (or the trusted hardware environment) that decrypts the memory is trusted (as it knows the encryption key). In other words, we target scenarios in which the tweak and the nonce can be observed by the adversary (or have some limited control over it), but not full control over them – for instance the adversary may ask for the decryption of messages with an arbitrarily set nonce and a guessed tag, but not force the use a given nonce for encryption. Hence, one can assume nonce-respecting adversaries.

5.1.1 General usage

For the general usage we have carefully considered the NIST requirements. For simplicity, let us assume an encryption scheme similar to ours, i.e. we have a block-wise mode of operation using a 128-bit tweakable block cipher. The nonce will be the shortest size required of a primary submission member, i.e. 96 bits.

We believe that for general purpose usage, with nonce-respecting users, two cases must be distinguished:

(a) **The user/target device is capable of maintaining an internal state, also across reboots, including those caused by accidental or intentional power disconnections.**

For instance, this is the case if the device has a small internal flash memory, which is on the same die or in the same package as the cryptographic engine, offering tampering detection, and the nonce is incremented and stored before being used.

In this case, the unit can store a counter for the nonces and we argue that there is no need for 96-bit nonces. Even changing nonces one *billion* times a second (assuming the device has enough computational power and communication bandwidth to work under these assumptions), nonces would overflow in more than 2500 billion years! 80-bit nonces would overflow in 38 million years, 72-bit nonces would overflow in 150 millennia, 64-bit nonces would overflow in 580 years.

Because of this, we argue that the NIST requirements are overkill in this case.

Therefore the nonces in this case should be required to be progressive message counters, and there is room for a block index within the message of 60 bits. In this case there is no need for nonce rotation: Algorithm 1.1 on page 11 resp. Algorithm 1.2 on page 12 would have a 60-bit field for the index, a 64-bit field for the nonce, Lines 16–18, resp. 19–21, could be removed.

(b) **The user/target device does not have an internal state that survives reboots, except for the encryption key.**

In this case, nonces have to be generated at random and the only non-repetition guarantee is offered by the randomness. We first note that the NIST call requires the ability of encrypting $2^{50} - 1$ bytes, which in the worst case (from number of involved tweaks) means $2^{50} - 1$ different tweaks. While the birthday paradox suggests repeated nonces (with good probability), we note that their existence is similar to that of a "regular" random encryption process. Moreover, to truly exploit such nonce repetitions, attack using five message blocks (i.e., 80-byte message) are needed: asking for the encryption of (X, Y_i, Y_i, Y_i, Y_i) for a shared block X for all messages (used to detect the nonce collision in the first ciphertext block), and four equal consecutive blocks, allowing to forge a message,tag pair of the form (X, Y_i, Y_i, Y_j, Y_j). As this attack assumes 80-byte messages (that can be reduced to 65-byte

message by picking Y_i to be a padded block of a single byte of information), it allows the use of about 2^{44} different tweaks. One can see that the success rate of this attack is about 2^{-9}, and even then, its impact is very limited.

Considering now PANORAmA with nonce rotation, as we shall see in Section 5.3 on page 36, a first *detectable* collision between first rotated nonces will occur with likelihood about 1/2 after $\approx 2^{71.23}$ bytes. Therefore we do believe using nonce rotated is not a security concern in this context. (More in Section 5.3 on page 36.)

The above considerations have led us to define the parameter sets (A) and (C) on page 19.

5.1.2 Memory encryption

First of all, there are three types of attacks to memory contents:

(i) Violate the confidentiality of memory contents.

(ii) Replay memory contents.

(iii) Forge memory contents.

Furthermore, there are two different contexts:

(a) The memory is external to the *Central Processing Unit (CPU)* die, and it is accessed by the latter via a memory interface and through a memory bus. A snapshot of the memory can be obtained by platform reset attacks [CPGR05] and via cold-boot attacks [Pet07, HSH+08] which exploit memory content retention [And01, p. 281] at low temperatures [LM79, Sko02]. Even worse, the memory interface can be inexpensively and effectively interposed [KSP05, Win09, Pac18].

(b) The memory device is on the same die as the CPU, or in the same package offering tampering detection. We shall call this case, with some abuse of language, the case of internal memory. Continuous memory eavesdropping or even memory manipulation would be therefore extremely difficult, but platform reset attacks attacks and some form of cold-boot attack could still be possible.

Before discussing the security parameters for the case of external, interposable memory, we shall review the schemes that are used to guarantee the integrity of memory contents, as these will determine minimal requirements on the size of nonces.

We assume that we need to guarantee *spatial uniqueness* to a flat 64-bit memory space (i.e. every memory location has its own set of permutations applied to it), therefore requiring 60 bits for the encoding of the base address of a cipher block. The available space for the nonce is thus at most 64 bits.

5.1.2.1 The case of external, interposable memory

We are going to assume that risks (i) and (iii) are addressed by the robustness of the encryption algorithm. It is also usually required that a memory encryption system offers both spatial and temporal uniqueness.

A review of memory integrity mechanisms In what follows a *memory granule* is the smallest unit that is encrypted and integrity protected. It usually coincides with a cache line, but can also be a multiple thereof.

We consider the issue of attackers that attempt to replay older contents onto a memory granule.

In early memory integrity systems such as XOM [MVS00], Merkle Trees [Mer82] have been used to guarantee memory integrity. However, Merkle Trees suffer from two problems: updating the structure is in-

herently not parallelisable, and they require large hashes to guarantee security, because of the birthday paradox – in a real world scenario 128 bits are necessary. This leads to large memory overheads (about 33% for 4-ary trees, for instance with 64-byte memory granules and 16-byte hashes) and significant performance penalties.

An alternative to Merkle Trees in order to gain the ability to parallelise the update of the integrity structure is offered by Counter Trees. The first such structure in the scientific literature was Hall and Jutla's *Parallelisable Authentication Tree (PAT)* [HJ02, HJ05, HJ08] a tree whose nodes are *pairs* (v, h) consisting of a nonce v and a hash h. This has been followed by the *Tamper-Evident Counter (TEC)* Tree [ECL+07], and by the 8-ary counter tree with embedded tags used in intel's SGX [Gue16]. These structures can be updated in parallel, but they do not reduce memory overhead significantly.

A further technique to reduce the memory overhead induced by counters and tags is the use of Split Counters [YEP+06] in the context of Bonsai Trees [RCPS07]. Split Counters use a single major counter for a contiguous range of memory granules, and a minor counter for each memory granule in that range. For instance, in a 512-bit memory granule one could store 64 7-bit minor counters, a 56-bit major counter and reserve 8 bits for metadata. Therefore a memory granule worth of counter information would cover 64 memory granules, resulting in a very high arity of 64 and in a much shallower integrity tree than with monolithic counters. When the memory contents of a memory granule are updated, the corresponding minor counter is increased – when the minor counter overflows, the major counter is increased as well, which means that the integrity information for all the sibling memory granules must be updated as well (and in most schemes, they must be also re-encrypted). The nonce information for the integrity (and confidentiality) algorithms for a given memory granule is the concatenation of the corresponding major and the minor counters, i.e. 63 bits worth of information. By means of this strategy the memory overhead can be pushed to about 15% and less. These structures and some variations, such as Morphable Counters [SNR+18] are the current state of the art.

Note that integrity structures need only be updated when data is actually written to external memory. In a modern system with multiple levels of cache, this means eviction of a cache line from the last level of cache inside the security perimeter of the die containing the computational cores and the encryption engine.

Therefore in what follows we shall assume that the memory contents are protected by including counters in the computation of the ciphertext and tag, that the counters are protected by some form of counter tree, and that counters and tags are stored in memory in a reserved, but not hardened, area of the RAM.

The attacks In order to replay a memory granule an attacker has to observe the repetition of a (possibly split) counter for a given granule and the replace the memory granule together with its integrity tag. This will happen, in the above examples, after at least 2^{63} writes (cache evictions).

The lowest complexity attack, i.e. replay any content to the same location, takes time 2^{63} writes to memory. Note that the attacker needs to evict both the text and the the cache line containing the tag value. Since the whole cache line containing the tag must be evicted even if the tag is only 64 bits, we have 2^{64} evictions.

The time needed for 2^{64} cache evictions is therefore a measure of the complexity of an attack. Very high performance servers can have an aggregated sustained memory bandwidth of up to $\approx$ 230 GB/s using 32 channels (we are going for the most pessimistic estimates even if usually only one channel can be used to access a given memory location). Now, since we are looking at a POWER8, we assume a 128-byte cache line. 2^{64} cache line evictions correspond to 2^{71} bytes of traffic which, at the quoted bandwidth, would take at least $2^{33.15}$ seconds, i.e. 303 years. (Had we considered a recent Xeon with its buffered memory interface and 64-byte cache lines, the results would have been similar.)

This does not take into account other memory traffic (that may share the same channels, such as the other

nodes of the integrity tree), including the traffic for the hashes and the fact that the attacker needs time between cache evictions to "harvest" the data from memory reliably with a high-performance interposer – this will at least halve the bandwidth useable for the attack, bringing the time to over 600 years.

Including the address in the hash computation will force the attacker to use just one memory channel, *de facto* reducing the available bandwidth by a factor of 8 (in the POWER8 case) and thus multiplying the time required for the attack by the same factor (to almost five millennia).

Note that the above complexity estimation does not distinguish between keyed and unkeyed tags/hashes.

Furthermore, if the cache line containing the tags is integrity protected itself, even if by just an additional level of hashing, then either the tag of the targeted line must match as well, or the whole line has to be replaced, increasing significantly the complexity of the attack.

We claim that in this case having 64-bit nonces and 64-bit tags makes the scheme secure. Regarding plaintext confidentiality, we believe that memory contents should not be protected with a significantly weaker algorithm than data at rest. The current trend is to move to 256-bit keys for data at rest, so we recommend the use of 256-bit keys. This leads us to the choice of Qameleon instantiated with QARMA-128, and thus to Parameter Set (D). Only if power consumption or area are critical concerns, the use of QARMA-64 with the tweak compression described in Section 3.3 – or some other lightweight pseudorandom function to compress the tweaks – can be recommended, leading to a justification for Parameter Set (E). The key used in the tweak compression scheme can be the same as the main encryption key or a different one to possibly obtain stronger security.

5.1.2.2 The case of internal memory

In this case, risks (ii) and (iii) are out of scope.

Also, the attacker cannot observe several different encryptions to the same location. In fact, the best attack in this situation, ignoring power and *Electro-Magnetic (EM)* side channels, seems to be variant of the *cold boot attack*: cryogenically freeze the package, decap it while keeping it at low temperature, and extract the RAM module to attach it to a reading device. Against such attacks memory encryption is still advisable, but the attacker will observe, for a single key, a single ciphertext per location, with the physical address entering the tweak input. She may be helped in her cryptanalytic efforts by the use of a limited amount of chosen plaintext (for instance, while passing some chosen input to the device, that will then copy it into the protected memory).

For this purpose, it is unrealistic to obtain more than 2^{40} bytes of ciphertexts (this would be 1 TB of RAM), only a small portion of which will correspond to chosen plaintexts. It is to expected that a lot of errors will be introduced by the process of extracting the memory module as well, further complicating the cryptanalysis

We claim that in this case 128-bit keys offer sufficient protection, and there is no need to have temporal uniqueness of ciphertexts, but only spatial. This leads to Parameter Set (F).

5.1.2.3 Additional security targets

In addition to the discussion above, we do not claim additional security targets. Most notably, in the context of memory encryption, the notion of *Releasing Unverified Plaintext (RUP)* [ABL+14], is irrelevant. Consider a memory-encryption mechanism which is asked to decrypt a cache line. In such attack scenario, the adversary is allowed to choose arbitrary ciphertexts to the decryption oracle, and obtain the corresponding plaintexts (even if the adversary has no legitimate tag for it).

Obviously, an adversary with the capabilities of offering raw data for the decryption (and observing the

plaintext) is quite strong. In the internal memory scenario, this attack requires the ability to control inputs to the decryption process, and obtain the intermediate values[1] (all found inside the cryptographic engine). Such an adversary can easily ask for the decryption of real messages, and thus, break their confidentiality immediately.

The case for external memory is slightly more delicate, but maintains the same settings: the cryptographic engine may decrypt the cache line, pass it on to the CPU, which in turn act upon its contents, possibly waiting for a Flush instruction from the cryptographic engine in case of a failed authentication. This gives rise to issues related to speculative execution (e.g. [KGG+18, LSG+18]) or by adversaries capable of reading the decrypted memory before it is cleared (similarly to the case of internal memory). While the discussion of the latter problem is similar to the case of internal memory, the discussion of the first problem suggests that the device outside the cryptographic engine should be wary of such speculative execution. Given the impact of Spectre and Meltdown and the amount of work done to mitigate them, we believe that this is a legitimate assumption.

5.2 On the security of QARMA

QARMA has not yet withstood the same intense scrutiny as other ciphers mentioned in this document such as, for instance PRINCE, not to speak of the AES.

However, the closely related cipher MANTIS [BJK+16] has been under intense scrutiny and QARMA has been designed to resist the attacks that have been mounted on reduced round versions of MANTIS as well as most attack methodologies that have been mounted on reduced round versions of PRINCE. Its design is based on decades of research into overall structure, lightweight diffusion layer construction, and S-Box selection since the design of the AES, with the purpose to make the cipher lightweight but at the same time building hedges against a variety of attacks.

For instance, while essentially inheriting the round structure of MIDORI [BBI+15] through its MANTIS heritage, it modifies the components – both the S-box layer and the diffusion matrix – to resist the attacks that affect MIDORI-64 [GJN+17, TLS16]. Similarly, these choices, as well as the elimination of the central SuperBox make the attacks mounted so far on reduced-round MANTIS ineffective: Indeed, as discussed by Eichlseder in her Ph.D. Thesis [Eic18], the key-recovery attack on MANTIS-5 described in [DEKM17] does not seem to carry over. In [Eic18, § 3.5.3] the applicability of the techniques used to mount the successful attacks on MANTIS to QARMA is discussed, finding that *"The QARMA design fixes several of the issues that we exploited for the attack on MANTIS-5: The strengthened inner round permits no Superbox property, and the new S-box, MixColumns matrix and tweak schedule do not display the same differential fixed points. This means that neither the simple optimal differential characteristic, nor the clustering effects observed in Section 3.3.3 seem applicable."* One of the weaknesses exploited in MANTIS are the structural properties of its S-box, which are in part shared by QARMA's σ_0 – and this is the reason we dropped that S-Box in this submission. We remind the reader that in Section 4.2.2 on page 27 we described how σ_1 was chosen.

Eichlseder also comments *"While the building blocks appear stronger than those of MANTIS, they also have their downsides: The MixColumns matrix of QARMA permits related-tweak truncated-differential characteristics with fewer active S-boxes than the matrix of MANTIS, with only 30 (instead of 34 for the MANTIS matrix in either the MANTIS or [...]) active S-boxes for 5 rounds, and 48 (instead of 52) for the full 7 rounds."* We are aware of this – indeed we accepted this compromise as part of the original QARMA design – but we also observe

[1] For internal memory devices this means that the adversary was capable to read the internal memory of the cryptographic engine, e.g. by performing a cold boot attack. This allows the adversary to mount only a single round of the attack and to time the freezing of the device to the exact moment in time where the decrypted values are stored, but not yet discarded due to failed decryption.

that these related-tweak truncated-differential characteristics do not seem to be exploitable – not only for the particular use cases of memory encryption, where tweaks are not under adversarial control (and may even be partially encrypted themselves in RAM), or when using long tweaks, but in general. Furthermore, *some* resistance against truncated differential cryptanalysis is provided: using the methods from [SLG+16], in [Ava17] is proved that truncated *impossible* differential cryptanalysis is not feasible for the full ciphers.

Indeed, Eichlseder and Kales [EK18] successfully break the security claims of MANTIS-6, but their methods (as mentioned by Kales at the QA session after his presentation at FSE 2019), applied to QARMA, breaks at most QARMA-4.

We start reviewing the analysis performed (or recycled from other designs) during QARMA's design. Then we shall review third party cryptanalysis and observations. Finally we will review our analysis on the use of the Even-Mansour design and review the corresponding security claims from [Ava17].

5.2.1 Design cryptanalysis

As reported in [Ava17], the cipher QARMA has been designed to withstand several attacks:

(i) Linear and differential cryptanalysis in the single-key, single-tweak model (MILP models, following the techniques described in Beierle's PhD Thesis [Bei18, Section 5.4.5]);

(ii) Differential cryptanalysis under a single-key, related-tweak model (MILP models, following Beierle);

(iii) Reflection Attacks (resistance follows from structure);

(iv) Generic attacks on Even-Mansour schemes (resistance follows from structure, see Section 5.2.3);

(v) Slide attacks (follows from round heterogeneity);

(vi) Meet-in-the-middle attacks (follows from MIDORI's resistance because of round similarity);

(vii) Invariant subspace attacks (new heuristic arguments presented in the paper);

(viii) Algebraic cryptanalysis (by counting equations and variables);

(ix) Impossible (truncated) differential and zero correlation linear cryptanalysis (using the method from Sun et al. [SLG+16]); and

(x) Higher order differential cryptanalysis (follows from MIDORI's resistance because of round similarity).

5.2.2 Disclosed cryptanalysis

Complexity details for all the following result are given in Table 5.1.

Regarding further cryptanalysis, the very first third party result we are aware of is a *Meet-in-the-Middle (MITM)* key recovery attack against 10-round QARMA-64 and 10-round QARMA-128, without outer whitening. The attack does not seem to be extendable further. This seems to confirm the designer's analysis that MITM attacks to 11 or more rounds should not be feasible.

Yang, Qi, and Chen [YQC18] generalise truncated differences and them in an impossible differential attack on 11-round QARMA-64.

Zong, Dong, Wang [ZDW18] derive related-tweakey/key impossible differentials from single-key ones, and presents also a tool for constructing those characteristics. Both QARMA-64 and Joltik-128 are studied. Again, this attack reaches 11-round QARMA-64.

Li and Jin [LJ18] mount meet-in-the-middle attacks to QARMA-64 and QARMA-128 including the outer

Table 5.1: Cryptanalysis of QARMA – Selected Published Results

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

The number of analyzed rounds is written as $x+y$, where x is the number of S-Box layers before the Pseudo-Reflector and y is the number of S-Box layers after it.
A Y under "Short Rounds" means that the first and last rounds of the analysed cipher are short and not full.
A Y or a N under "Outer/Inner Whitening?" denotes whether the additions of the Outer/Inner whitening keys are included in the cryptanalysed cipher or not.
The label 128* means that in [YQC18] a superseded set of matrices is used for QARMA-128, that has been later replaced in the final published version.

whitening key additions. The authors attack 8 and 9 rounds of QARMA-64 and 10 of QARMA-128.

Li, Hu and Wang [LHW19] have constructed Related-Tweak Statistical Saturation attacks, and mounted such an attack to 10-round QARMA-64 with outer whitening key and an 11-round attack on QARMA-128.

Ankele, Dobraunig, Guo, Lambooij, Leander, and Todo [ADG+19] present an attack on 12-round QARMA with four forward and eight backward rounds (counted as S-Box layers). This attack is very important because it suggests that the $\texttt{QARMA-64}_5$ might be breakable.

Some of these attacks break the $T \cdot M, T \cdot D \leq 2^{n-\epsilon}$ barrier for the reduced round versions of the cipher but the remaining security margin is still quite ample, at least 4 rounds for $\texttt{QARMA-64}_7$, 8 rounds for $\texttt{QARMA-64}_9$, and upwards of 13 rounds for QARMA-128.

5.2.3 On the security of the Even-Mansour construction

Recall that the whitening key derivation function $\mathfrak{o}(\cdot)$ is an orthomorphism. If orthomorphisms are used to create a key schedule, the complexity of the attacks usually increases and approaches that of schemes with independent keys or using independent permutations (see [CLL+14] for the two-round case). Of course, QARMA is not composed of three ideal permutations, but we can gain some insight from the consideration of the attacks on Even-Mansour schemes.

An analogue of the cryptanalysis described in [DDKS13] seems unlikely to be applied directly: The attack on single-key three-round designs with an involutory round is the one that seems closer to our design. It can be adapted at once observing that, for each fixed core key, the mapping $x \mapsto \Delta$ assumes only $2^{n/2}$ values which occur $2^{n/2}$ times each (with respect to the notation in [DDKS13], this is in-degree t). This leads to a time/data/memory (T/D/M) tradeoff of $TD = 2^{3n/2}$, where the data consists of known texts and memory

is online storage. The attack has to be repeated for each candidate core key in order to determine the whitening key as well, so we obtain $T = 2^{7n/4}$, $D = 2^{3n/4}$ and $M = D$. Since evaluations of the sub-ciphers can be done for pairs of core keys $(k_0, k_0 + \alpha)$, the memory usage can be halved. For QARMA-64 this turns into an attack with 2^{112} time and 2^{47} blocks of data, i.e. 2^{50} bytes, for QARMA-128 we can get a tradeoff of 2^{224} time and 2^{95} blocks of data, i.e. 2^{99} bytes, or a different tradeoff of, say, 2^{255} operations with 2^{64} data or 2^{272} operations with 2^{47} data, and in this case the lower requisite of 2^{224} operations with 2^{46} 16-byte blocks of data is most likely met. In fact, for this mode we could even claim security up to 2^{56} 16-byte blocks, or 2^{60} bytes, processed.

The single-key two-round attack can be applied, under the assumption that for, a known core key, a certain difference Δ at the sides of the central construction will occur with likelihood $2^{-n/2}$ (but choosing the plaintext does not seem to give control on this event).

In this case the cipher collapses into a single-key two-round Even-Mansour construction, not a single round EM, because $\mathcal{o}(\cdot)$ is an orthomorphism and thus the sum of the two whitening keys w_0 and $w_1 = \mathcal{o}(w_1)$ is a value 1-1 with w_0. For each core key, time complexity is slightly smaller than 2^n, data (known texts) is slightly smaller than $2^{3n/4}$, and memory is around $2^{n/4}$. The time complexity must be multiplied by 2^n to cover all core keys, and the data by $2^{n/2}$ because of the usable proportion, whereas online memory usage stay the same. We do get an attack with T slightly better than brute force, but $TD \sim 2^{14n/4}$. For instance, with $n = 128$ we have $T = 2^{256}$ with $D = 2^{192}$ and with $n = 64$ we have $T = 2^{128}$ with $D = 2^{96}$.

Also in this case the NIST requirements (and recommendations) as well as our increased bounds above would be satisfied.

Similarly, for the attacks in [DDKS15], with the same likelihoods for a known central difference Δ for a certain core key (resp. class of keys), the equations to solve for the whitening key (and possibly the remaining bits of the core key) would still correspond to the whole cipher minus the central construction, so we do not expect it to be significantly easier than attempting to exploit reflection characteristics.

Finally, a three-round, two-key Even-Mansour scheme, according to [DDKS14] is attacked with a time/data tradeoff of $TD = 2^{2n}$ where $M = D$, for *unkeyed* permutations – so the complexity must be increased to take guesses of the core key into account as above. It is an open question whether our scheme, with a second key derived from the first by means of an orthomorphism offers the same security bound.

We consider now the adoption of the PRINCE-like lower estimates for the security of QARMA on the basis of existing analysis of Even-Mansour constructions to have been excessively conservative. We are thus claiming that:

- Attacks using 2^{50} bytes of data would require more than 2^{224} computations to break the plaintext confidentiality of QARMA-128; and
- Attacks using 2^{40} bytes of data would require more than 2^{112} computations to break the plaintext confidentiality of QARMA-64.

5.3 On the security of PANORAmA

For simplicity let us consider here the case of 128-bit blocks and tweaks.

Without the nonce rotation, and assuming counters of unlimited magnitude, we can bound the security using, any security proof for the *Beyond the Birthday Bound (BBB)* security of ΘCB3. For instance one can easily adapt the security proof of the TAE mode of operation by Liskov, Rivest, Wagner [LRW02, LRW11]. In other words, the in the nonce-respecting setting; more precisely, confidentiality is perfectly guaranteed and the forgery probability is $2^{-\tau}$, independently of the number of blocks of data in encryption/decryption

queries made by the adversary, where τ is the tag length.

However, for PANORAmA there is an additional case where distinguishability can go wrong: if nonce rotation is implemented, repeated tweaks could be detected. In the single-key, nonce-respecting setting, we note that this occurs only if there is first a collision between *rotated* nonces, as the original nonces cannot collide before they are all used up, and the separation of tweaks in the highest bit guarantees that only tweaks after the first 2^{28} blocks can collide. Assuming that the tweakable block cipher is ideal, the attacker may, for instance, ask for the encryption of several all-zero plaintexts, and check whether some blocks collide. Another option to exploit a colliding rotated tweak is to apply the attack of Section 5.1.1 on page 29 (scenario B).

The case of a rotated nonce collision happens with probability of about 1/2 after $\sqrt{2 \ln 2} \cdot 2^{48}$ rotated nonces are computed. As each rotated nonce happens after 2^{28} message blocks are processed, a rotated nonce collisions are expected only after about 2^{76} blocks (i.e., 2^{80} bytes). Obviously, this is beyond the data limit for a single key.

5.4 On the security of long tweak compression

The need to support long tweaks without altering the design led us to offer a method to transform a long tweak (in this proposal, of $2n$ or $3n$ bits) into a standard-length tweak of n bits. Moreover, we aim to do so in an way that does not increase the latency by much, while maintaining the security level.

5.4.1 Tweak collisions

Hence, we need a function $f(\cdot)$ that accepts $2n$- or $3n$-bit tweak and produces an n-bit one. This function should satisfy two main goals: It should be hard to find collisions in $f(\cdot)$, as such collisions result in the same effective tweak, which in turn can be easily identified, and it should be hard to exploit random collisions to extract keying material. The reason one should key the function $f(\cdot)$ is to avoid a huge precomputation that affects all the instantiations, and thus, as collisions exist, we should not be able to exploit them for key-recovery.

Let us introduce some notation. Let T be a $2n$- or $3n$-bit value. We split it into 2 resp. 3 equal segments as $T_1||T_0 \leftarrow T$ or $T_2||T_1||T_0 \leftarrow T$. A key K is split as $K_1||K_0 \leftarrow K$.

For the case of $2n$-bit tweaks, one can think of the following conditions:

- $f_k(T_0, \cdot) : \{0,1\}^n \rightarrow \{0,1\}^n$ should be a permutation,
- $f_k(\cdot, T_1) : \{0,1\}^n \rightarrow \{0,1\}^n$ should be a permutation,
- It is computationally infeasible to deduce information about the key K given a collision $f_K(T_0, T_1) = f_K(T'_0, T'_1)$.

The first two conditions ensure that a tweak collision *must* involve a difference in both tweaks. Moreover, these conditions guarantee that in the context of a single process, one cannot find a collision between two different instances of $f_k(\cdot)$. Similarly, for $3n$-bit tweaks, we demand that $f_k(T_0, T_1, \cdot)$, $f_k(T_0, \cdot, T_2)$, and $f_k(\cdot, T_1, T_2)$ are all permutations.

Our proposal for $f_k(\cdot)$ is based on the well established technique of XORing *Pseudo-Random Permutations (PRPs)* to generate a *Pseudo-Random Function (PRF)* [Luc00, Pat08, Pat13, CLP14, MP15]. Namely, we propose to use:

$$f_k(T_0, T_1) = g_0(T_0 \oplus K_0) \oplus g_1(T_1 \oplus K_0)$$

and

$$f_k(T_0, T_1, T_2) = g_0(T_0 \oplus K_0) \oplus g_1(T_1 \oplus K_0) \oplus g_2(T_2 \oplus K_0) \ .$$

It is easy to see that assuming $g_i(\cdot)$ to be PRPs (and they are as we pick them as four, resp. five full rounds of QARMA-64, resp. -128, keyed with K_1 and different round constants for each $g_i(\cdot)$), then the condition on the permutation is immediately satisfied.

A *tweak collision* occurs when

$$f_k(T_0, T_1) = f_k(T'_0, T'_1)$$

for double tweak lengths, and

$$f_k(T_0, T_1, T_2) = f_k(T'_0, T'_1, T'_2)$$

for triple tweak lengths. The fact that the adversary must have (at least) two active tweak words for a tweak collision to occur suggests that in the memory space of a single process there are indeed no such tweak collisions.

Still, we may be interested in what information can be deduced once a tweak collision is detected (and at what computational cost). We note that a tweak collision suggests:

$$g_0(T_0 \oplus K_0) \oplus g_1(T_1 \oplus K_0) = g_0(T'_0 \oplus K_0) \oplus g_1(T'_1 \oplus K_0)$$

(a similar equation holds for the 3 n-bit case). We first note that as we observe only the impact of the same tweak (as we discuss shortly), we do not obtain the actual value of $g_0(T_0 \oplus K_0) \oplus g_1(T_1 \oplus K_0)$. Hence, an adversary needs to guess both K_0 and K_1 to check whether the above condition holds. This of course takes time $O(2^{2n})$, which is equivalent to exhaustive search.

A different approach is to try and apply a differential attack, as we know that the output differences of $g_0(\cdot)$ and $g_1(\cdot)$ are the same. However, $g_0(\cdot)$ and $g_1(\cdot)$ use different sets of constants, making the analysis of the differential properties hard. The constants selected for $g_0(\cdot)$ and $g_1(\cdot)$ were selected to offer a number of active S-boxes which is as large as possible as we could find within the time constraints for this search.

In order to solve this issue we have used a bit-wise MILP-model: for 64-bit functions, the resulting program showed that there are at least 30 active S-boxes for the two branch case ending in a collision, and as each has probability of at most 2^{-2}, we obtain that the probability of a collision is upper bounded by 2^{-60}. For the 3-branch case, we picked the differences between the round constants of $g_0(\cdot)$, $g_1(\cdot)$ and $g_2(\cdot)$ such that every pair $(g_i(\cdot), g_j(\cdot))$ also has a large number of active S-boxes. The MILP program processing this problem suggests 31 active S-boxes for the pair $(g_0(\cdot), g_1(\cdot))$, and at least 28 each for the pairs $(g_0(\cdot), g_2(\cdot))$ and $(g_1(\cdot), g_2(\cdot))$.

For 128-bit functions and tweak doubling, hence the pair $(g_0(\cdot), g_1(\cdot))$, we get 60 active S-Boxes, i.e., a probability of 2^{-120}. For 128-bit tweak doubling, hence with the pair $(g_0(\cdot), g_1(\cdot))$, we get at least 59 active S-Boxes.

Hence, even if the adversary can control the difference between the tweaks (e.g., by starting and terminating the process to obtain a process id that fits the tweak difference), the chances that a tweak collision happens is very close to the random case. To conlcude, the analysis suggests that the "best" approach for generating tweak collisions is by a random chance, which is inevitable (but as we now show of little impact).

5.4.2 Impact of tweak collisions on Qameleon

We now turn our attention to what happens (and how the adversary can exploit) tweak collisions. We first discuss whether the collision can be detected when using Qameleon (and at what cost) and then we discuss whether these detected collisions can be used for attacks (and at what cost).

5.4.2.1 Detecting tweak collisions

Assume that the proposed encryption scheme was without the orthomorphism, i.e., if $P \mapsto E_K^{T_{\text{effective}}}(P)$. Obviously, this is a very simple case to detect tweak collisions, as after $2^{n/2}$ pairs of tweaks we expect such a collision, and asking for the encryption of two plaintexts P_0 and P_1 under all these tweaks, allows for an immediate identification of the colliding tweaks (as the corresponding ciphertexts will be the same).

In the $2\,n$-bit case, the addition of the orthomorphism breaks this property immediately, as for the two colliding tweaks T, T', the corresponding g_0 and g'_0 are different. For the $3\,n$-bit case, at least one of the pairs $(T_0, T'_0), (T_1, T'_1)$ must be different, again breaking the property.

At the same time, we note that for a given (key,tweak) combination, the addition of g_0 (and of g_1) is the same for all plaintexts. Hence, if we consider a $2\,n$-bit tweak collision, if by any chance $P \oplus g_0$ (encrypted with T) is equal to $P' \oplus g'_0$ (encrypted with T'), then their ciphertext will be the same. Actually, whenever the ciphertexts are the same, than necessarily, the plaintexts difference is $g_0 \oplus g'_0$. This gives rise to a simple attack — pick about $2^{n/2}$ pairs (or triplets) of tweaks, and ask for the encryption of about $2^{n/2+2}$ plaintexts under each of them. We then look for colliding ciphertexts between different sets (i.e. encrypted using different tweaks). If all collisions arise from plaintexts with the same difference, then with very good probability the sets correspond to a tweak collision.

In the $3\,n$-bit case, a tweak collision and a difference $g_0 \oplus g'_0$ between the plaintexts ensures a difference $g_1 \oplus g'_1$ in the ciphertexts. Hence, one can easily amend the above attack to work also for the longer tweaks.

We note that this attack requires about 2^n plaintexts (and about the same time). Given that after 2^{50} plaintexts one should re-key `Qameleon`, this attack is "outside" the security model.

A different approach would actually be to use a chosen ciphertext attack scenario. In the $2\,n$-bit case, the adversary picks two ciphertexts C_0 and C_1 and asks for their decryption under all $2^{n/2}$ tweaks. Tweak collisions could be found if the corresponding plaintexts P_0 and P_1 have the same difference with P'_0 and P'_1. This attack requires the ability to decrypt, and thus, we can consider two possible attack vectors:

- The case of releasing unverified plaintexts — in this case the adversary just obtains the corresponding plaintexts corresponding to the ciphertext queries, even if the tag is wrong/non-present. As we stated before, we do not discuss such RUP adversaries.
- The case of an adversary trying to forge tags (i.e., submit ciphertexts with an attempt on the tag). Such adversary needs to submit about 2^t pairs of (ciphertext, tag) to succeed in the decryption of a single ciphertext, where t is the tag length. Hence, for $2^{n/2}$ tweaks, we expect a total of $2^{n/2+t}$ (chosen ciphertexts,tags) tags. A different approach is to wait for 2^n legitimate blocks and apply the attack then. We note that the data complexity of both attacks is larger than the data limit (more than 2^{50} bytes for $n = 128$, and more than 2^{40} bytes for $n = 64$ whenever the tag length satisfies $t > 8$).

The adaptation of these attacks to the case of $3\,n$-bit tweak is immediate, resulting in a similar conclusion.

5.4.2.2 Elevating tweak collisions to attacks

One can use tweak collision for one of two purposes — either as a tool for distinguishing the output of `Qameleon` from that of a random function or for key recovery. As noted in the analysis of tweak collisions, given a tweak, one cannot extract key information about it. Thus, we now discuss distinguishing attacks based on tweak collision.

Once two tweaks collide, they call the underlying block cipher with the same parameters (as nonce-respecting adversaries may use the same nonce for different tweaks). In such a case, the addition of the orthomor-

phisms is mandatory, as otherwise one could immediately distinguish the tweak collision by asking for the encryption of the same 2-block plaintext under all $2^{n/2}$ tweaks. A collision of two ciphertexts happens with probability 1 if there was indeed a tweak collision, whereas for a random function repeating two ciphertext blocks in $2^{n/2}$ samples is unlikely.

The addition of the orthomorphism changes this behavior, as even though the invocation of QARMA which accepts the same key and effective tweak, it is masked from the outside by different values.

5.5 On side-channel resistance

We did not include any specific technology in the design to protect against side-channel attacks.

QARMA (and thus Qameleon) can resists side-channel attacks by adopting the same techniques as any other bricklayer block cipher design. The available literature of techniques to provide this type of protection to QARMA is too large to be meaningfully included here. We observe that the simpler diffusion layer without finite field multiplications, and the use of 4-bit S-Boxes instead of 8-bit S-Boxes, should make this lighter than for other ciphers, such as the AES.

For instance, for the implementation of the S-Boxes the techniques from [BNN+12, BGN+14, BNN+15] could be used. Other techniques include dual-rail logic [TV04, PM05, PM06] or isolation of the power supply [Sha00, GMOP15] and shielding of the circuit.

6 Implementations

6.1 Software

Qameleon is accompanied by a software package containing a C implementation of each of the proposed variants. The purpose of the attached software implementations is to provide a reference for analysis and further implementations of the algorithm. The code is designed to be simple to be understood and portable. For this reason, no optimisations or platform specific functions have been used.

The implementation of each variant is in its own subdirectory of crypto_aead following the submission guidelines and named exactly as in Section 3.1 on page 19. Each variant has been verified using the test vectors indicated in the submission rules, except for the memory encryption variants which use a smaller set of test vectors (the program to generate them is always included). The test vector output files for each variant is provided as well.

Remark *In the variants where the message length is not always a multiple of the block length, i.e. Sets (A), (B), and (C), we append the message length to the ciphertext in a 64-bit field that after the authentication tag as a result of the function crypto_aead_encrypt. The function crypto_aead_decrypt will then use this piece of information to properly decrypt. We do not encrypt this length because it is a public parameter.*

This is done only to meet the submission guidelines and the limitations of SUPERCOP and of, that mandates "The outputs of crypto_aead_encrypt and crypto_aead_decrypt shall be determined entirely by the inputs listed above (except that the parameter nsec is kept for compatibility with SUPERCOP and will not be used) and shall not be affected by any randomness or other hidden inputs." *This seems to imply that the length of the plaintext must be recoverable from the ciphertext. In our case this is in theory possible by trying, for the last block, all possible truncated versions and verifying the padding as well, however this is cumbersome and introduced a negligible likelihood of incorrect decryption. Hence, we prefer to append the length. Of course this makes the output of crypto_aead_encrypt no longer distinguishable from a random string, but this is easily solvable: For instance, as mentioned before, by encrypting the message length as well, but such a choice belongs in our opinion to the domain of protocols, and not of ciphering primitives.*

6.2 Hardware

Qameleon was designed with the objective of being efficient for memory encryption. This necessitates that the underlying block cipher be able to encrypt and decrypt data with minimum delay. PRINCE [BCG+12b, BCG+12a], was the first block cipher designed explicitly for memory encryption. The cipher when unrolled in hardware can encrypt data in hardware with very little signal delay. Also designing a combined encryption/decryption circuit is very efficient in hardware as it only requires small alteration to the master key. PRINCE however has only a variant with a block size of only 64 bits that offers a security of 128 bits. It is also not known how to PRINCE incorporate a tweak in PRINCE. So applying a mode like ΘCB as in the Deoxys CAESAR submission [JNPS16] to PRINCE is not directly possible.

The QARMA-128 tweakable block cipher family [Ava17], offers us the advantages of of a 128-bit block, 256-bit security and a 128-bit tweak and hence can be used as the underlying encryption engine in tweakable modes of operation such as ΘCB. For block ciphers offering 128-bit blocksize QARMA-128$_r$ is fast enough in hardware when compared with other ciphers in literature for $11 \leq r \leq 14$. In Table 6.1 on the next page

	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**

Table 6.1: Implementation results for various block ciphers. (Power reported at 10 MHz)

we report the performance characteristics of some well known block ciphers when implemented using the standard cell library CORE90GPSVT 2.1.a (STM 90nm). The designs were fully unrolled and exclusively optimized for area. Apart from MIDORI-128 [BBI+15] which was designed for energy efficiency, QARMA-128_r finds itself well placed in terms both area and signal delay to compute an encryption operation.

6.2.1 Qameleon: PANORAmA using QARMA-128 (Circuit details)

The circuit for Qameleon, is shown in Figure 6.1 on the following page. This is the PANORAmA mode of operation using QARMA-128 as the underlying block cipher, in other words the ΘBC mode with minor differences related to incomplete block processing and nonce rotation. The 128-bit tweak is generated by combining the Nonce, current value of the counter and a 4 bit nibble that depends on the size of the plaintext and AD. The value of this nibble and all other select signals are generated centrally. Note that the nonce is included in the tweak only during the processing of the message blocks and so it has to filtered to zero when the AD is being processed. Apart from the block cipher core, the circuit has two 128-bit registers Σ and Auth to store the running sum of the plaintext blocks, and to store the output of the result of processing the AD phase.

The tweak arrangement is the most subtle part of the design and is composed thus. We keep the updated value of the nonce in a register, which is refreshed every 2^{28} encryptions of message blocks. The count of the the number of encryptions is natuarally kept in a 46 bit counter which is compared with the 0x0fffffff signal to trigger such an update. When such an update is required the input string consisting of the 110 leading zeros and 18 msbs of the counter (Ctr_{MSB}) are encrypted and updated on the register. Note that the tweak consists of the first 4 leading bits Nib_1, a 96 bit field occupied by the original nonce or the updated value of the nonce and a 28 bit field meant for either the current 28 lsbs of the counter, the 0^{28} signal, or the 28 lsbs of the bytelength of the AD/message (for the last block processing). Each takes a particular value depending on the stage of operation of the mode, and all signals to filter these are generated centrally.

The input to the encryption circuit can be either the plaintext, AD, the $0^{110}||Ctr_{MSB}$ or the contents of the sum (Σ) register. The output of the encryption engine is output of the ciphertext (CT). The last encryption pass produces the tag which is obtained by adding the block cipher output to the Auth register.

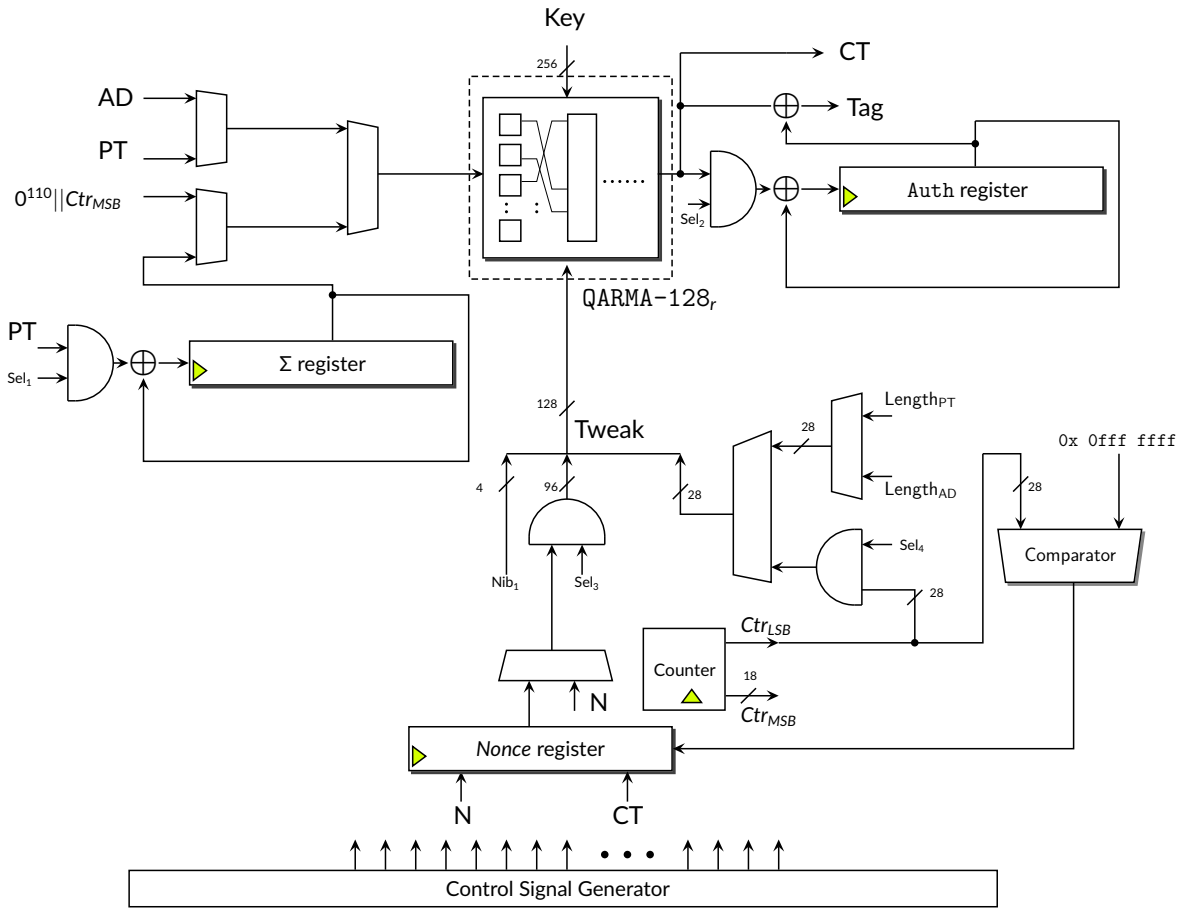

Figure 6.1: Hardware circuit for Qameleon

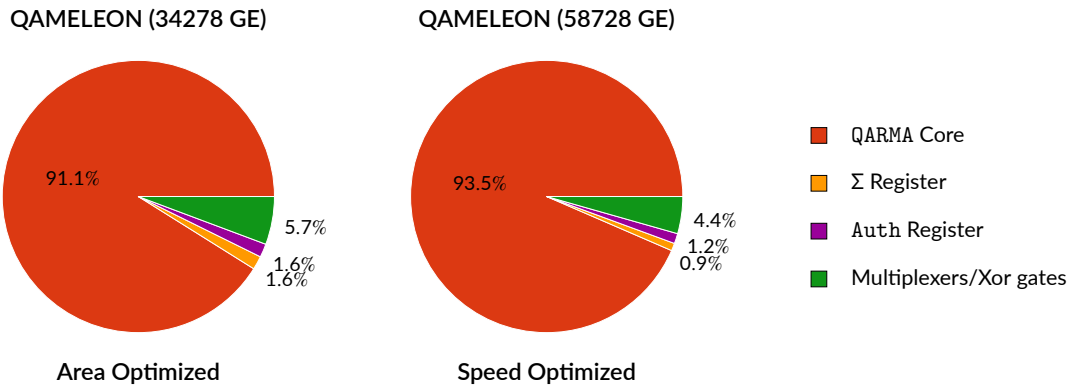

Figure 6.2: Componentwise area requirements for Qameleon-128 using QARMA-128$_{14}$

	Variant	Block Cipher	Optimization	Area (GE)	Power(*m*W)	Delay(ns)
(A)	qameleon12812896gpv1	QARMA-128$_{11}$	Area	35053	35.0	20.98
			Speed	59678	69.4	9.92
		QARMA-128$_{12}$	Area	37637	41.2	22.49
			Speed	66987	85.5	10.41
		QARMA-128$_{13}$	Area	40225	46.8	23.93
			Speed	73131	99.6	10.87
		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

Table 6.2: Implementation results for Qameleon variants (Power reported at 10 MHz)

6.2.2 Timing

The QARMA-128$_r$ block ciphers are implemented fully unrolled, and hence it takes only 1 cycle to complete one encryption function. Thus each block of AD or plaintext is processed in a single cylce. If the number of blocks of AD and plaintext are n_a and n_m respectively then a total of $T = n_a + n_m + 1$ cycles are required for a single encryption pass. The additonal cycle is required to execute the final encryption pass that produces the tag.

6.2.3 Performance

In Table 6.2 on the previous page we present the synthesis results for the designs. The following design flow was used: first the design was implemented in VHDL. Then, a functional verification was first done using Mentor Graphics Modelsim software. The designs were synthesized using the standard cell library of the 90nm logic process of STM (CORE90GPHVT v2.1.a) with the Synopsys Design Compiler, with the compiler being specifically instructed to optimize the circuit for area. A timing simulation was done on the synthesized netlist. The switching activity of each gate of the circuit was collected while running post-synthesis simulation. The average power was obtained using Synopsys Power Compiler, using the back annotated switching activity.

We implemented Qameleon-128 using two design philosophies: the first optimizes area and the second speed. Our implementation of Qameleon-128 using QARMA-128$_{14}$ optimized by area/speed occupies 42811/74177 GE repectively. A component-wise breakup of the circuit is givem in Figure 6.2 on page 43. In Table 6.2 on the previous page we present detailed comparison of Qameleon-128 with ΘCB using DeoxysBC-384. We measure various performance characteristics for area and speed optimized circuits. It can be seen that in terms of area, power consumption and signal delay Qameleon is better placed in the design space.

7 Summary of Features

Our summary of features is written in the style of Dutch/Belgian Dissertation *Stellingen*:

1. Qameleon is an efficient general purpose AEAD cipher.
2. Qameleon is built on known and well established design principles.
3. All design aspects are documented and the origin of all constants is described.
4. Qameleon offers BBB security, because of its instantiation of PANORAmA (a variant of ΘCB) with a TBC. This is in stark contrast to OCB, that only provides security up to the birthday bound.
5. The security of Qameleon is lost if nonces are reused.
6. The structure of its underlying TBC QARMA is also very fine grained (i.e. it is based on many very lightweight rounds, instead of few heavier ones) so that it can be very easily partitioned in various ways for pipelining, depending on use case constraints.
7. The mode of operation is highly parallelisable.
8. Because of the last two features, the design is ideal for memory encryption applications, where the performance impact is mostly determined by the initial latency (i.e. the latency of producing the first block), provided that the implementation can then keep up with the memory bus bandwidth.
9. Implementations can be tuned to make energy consumption very low. This makes the cipher suitable for low power use cases (for instance, *Internet of Things (IOT)* devices).
10. The lack of set up time, the absence of complex key schedule, and the low initial latency make Qameleon suitable for the processing of short messages as well.
11. All blocks are processed in the same way – there is no fractional block processing. This makes implementation easier and less error prone.
12. In the ARM ecosystem robust implementations of QARMA are already being used for the purpose of pointer authentication, which means that the corresponding cryptographic hardware is already available. It is to be expected that QARMA encryption instruction could be deployed very quickly in case of standardisation (or de-facto standardisation) of the design.
13. Qameleon comes with two technologies that further increase its versatility.
    (a) The first is *nonce rotation*, that allows to increase the nonce and block counter spaces without having to design versions of the underlying TBC that process longer tweaks. We prove that this does not represent a security risk.
    (b) The second technology is *tweak compression*, for the cases where a larger tweak space is still needed. As the name suggests, a longer tweak is compressed to a shorter one using a proven PRF-as-XOR-of-PRPs construction. In order to make collisions between short tweaks non detectable and non useable, masking using the values of the component functions of the PRF construction, modified with orthomorphisms, is added to the actual "extended tweak" cipher.
14. Qameleon is a cool name.
15. Qameleon has a song (see next chapter).

Acknowledgments

We thank Itai Dinur, Maria Eichlseder, and Jérémy Jean for fruitful discussions. This work was also made possible by the tools developed by Leo Perrin (author of `sboxU`), Oleksandr Kazymyrov, Maksim Storetvedt, and Anna Maria Eilertsen (authors of the S-Box analysis tool at https://github.com/okazymyrov/sbox), Stjepan Picek, Lejla Batina, Domagoj Jakobović, Barış Ege, and Marin Golub (authors of SET) and the SageMath and GUROBI developers.

We also thank Arrigo Triulzi who suggested the name `Qameleon`, after the Culture Club song "Karma Chameleon." `Qameleon` has an official song. The lyrics, written by the submitters and Arrigo Triulzi, are given below. They are meant to be sung to the tune of aforementioned Culture Club hit.

```
   Full encryption in your RAM all the way
   If I hack onto your bus, would you say:
   There's some data with no decryption,
   With a tag that doesn't match,
   How to sail the contradiction?
   Just crypt and go, just crypt and go.

   QARMA QARMA QARMA QARMA QARMA Qameleon:
   Just crypt and go, just crypt and go.
   'crypting would be easy if all ciphers were like my dreams:
   With tweak and key, with tweak and key.

   Didn't you keep your keys safe every day
   And you used to be so careful I heard you say
   That your data was your addiction?
   When we encrypt, our data is strong,
   Erase the key, it's gone forever:
   Just crypt and go, just crypt and go.

   QARMA QARMA QARMA QARMA QARMA Qameleon:
   Just crypt and go, just crypt and go.
   'crypting would be easy if all ciphers were like my dreams:
   With tweak and key, with tweak and key.

   Every bit is like survival,
   Choose my cipher, not my rivals'.
   Every bit is like survival,
   Choose my cipher, not my rivals'.

   There's some data with no decryption,
   With a tag that doesn't match,
   How to sail the contradiction?
   Just crypt and go, just crypt and go.

|: QARMA QARMA QARMA QARMA QARMA Qameleon:
   Just crypt and go, just crypt and go.
   'crypting would be easy if all ciphers were like my dreams:
   With tweak and key, with tweak and key. :| [repeat and fade]
```

Bibliography

[ABL+14] Elena Andreeva, Andrey Bogdanov, Atul Luykx, Bart Mennink, Nicky Mouha, and Kan Yasuda. How to securely release unverified plaintext in authenticated encryption. In Sarkar and Iwata [SI14], pages 105–125.
Cited in this document on page 32.

[ADG+19] Ralph Ankele, Christoph Dobraunig, Jian Guo, Eran Lambooij, Gregor Leander, and Yosuke Todo. Zero-correlation attacks on tweakable block ciphers with linear tweakey expansion. *IACR Transactions on Symmetric Cryptology*, 2019(1):192–235, Mar. 2019.
Cited in this document on page 35.

[AKT14] Can Acar, Arvind Krishnaswamy, and Robert Turner. Code pointer authentication for hardware flow control, October 2014. United States Patent US9514305 B2. Assignee: QUALCOMM Incorporated.
Cited in this document on page 2.

[And01] Ross J. Anderson. *Security engineering - a guide to building dependable distributed systems*. Wiley, 2001.
Cited in this document on page 30.

[ARM16] ARM Connected blog. ARMv8-A architecture – 2016 additions. https://www.community.arm.com/processors/b/blog/posts/armv8-a-architecture-2016-additions, October 2016.
Cited in this document on page 2.

[Ava17] Roberto Avanzi. The QARMA block cipher family. almost MDS matrices over rings with zero divisors, nearly symmetric even-mansour constructions with non-involutory central rounds, and search heuristics for low-latency s-boxes. *IACR Trans. Symmetric Cryptol.*, 2017(1):4–44, 2017.
Cited in this document on pages 2, 7, 10, 13, 15, 17, 26, 27, 34, and 41.

[BBI+15] Subhadeep Banik, Andrey Bogdanov, Takanori Isobe, Kyoji Shibutani, Harunaga Hiwatari, Toru Akishita, and Francesco Regazzoni. Midori: A Block Cipher for Low Energy. In Tetsu Iwata and Jung Hee Cheon, editors, *Advances in Cryptology - ASIACRYPT 2015 - 21st International Conference on the Theory and Application of Cryptology and Information Security, Auckland, New Zealand, November 29 - December 3, 2015, Proceedings, Part II*, volume 9453 of *Lecture Notes in Computer Science*, pages 411–436. Springer, 2015.
Cited in this document on pages 16, 33, and 42.

[BC18] Christina Boura and Anne Canteaut. On the boomerang uniformity of cryptographic sboxes. *IACR Transactions on Symmetric Cryptology*, 2018(3):290–310, Sep. 2018.
Cited in this document on page 27.

[BCG+12a] Julia Borghoff, Anne Canteaut, Tim Güneysu, Elif Bilge Kavun, Miroslav Knezevic, Lars R. Knudsen, Gregor Leander, Ventzislav Nikov, Christof Paar, Christian Rechberger, Peter Rombouts, Søren S. Thomsen, and Tolga Yalçin. PRINCE - A Low-Latency Block Cipher for Pervasive Computing Applications - Extended Abstract. In Xiaoyun Wang and Kazue Sako, editors, *ASIACRYPT 2012*, volume 7658 of *Lecture Notes in Computer Science*, pages 208–225. Springer, 2012.
Cited in this document on pages 13 and 41.

[BCG+12b] Julia Borghoff, Anne Canteaut, Tim Güneysu, Elif Bilge Kavun, Miroslav Knezevic, Lars R. Knudsen, Gregor Leander, Ventzislav Nikov, Christof Paar, Christian Rechberger, Peter Rombouts, Søren S. Thomsen, and Tolga Yalçin. PRINCE - A Low-latency Block Cipher for Pervasive Computing Applications (Full version). *IACR Cryptology ePrint Archive*, 2012:529, 2012.
Cited in this document on page 41.

[Bei18] Christof Beierle. *Design and analysis of lightweight block ciphers: a focus on the linear layer*. PhD thesis, Ruhr University Bochum, Germany, 2018.
Cited in this document on page 34.

[BGN+14] Begül Bilgin, Benedikt Gierlichs, Svetla Nikova, Ventzislav Nikov, and Vincent Rijmen. Higher-order threshold implementations. In Palash Sarkar and Tetsu Iwata, editors, *Advances in Cryptology - ASIACRYPT 2014 - 20th International Conference on the Theory and Application of Cryptology and Information Security, Kaoshiung, Taiwan, R.O.C., December 7-11, 2014, Proceedings, Part II*, volume 8874 of *Lecture Notes in Computer Science*, pages 326–343. Springer, 2014.
Cited in this document on page 40.

[BJK+16] Christof Beierle, Jérémy Jean, Stefan Kölbl, Gregor Leander, Amir Moradi, Thomas Peyrin, Yu Sasaki, Pascal Sasdrich, and Siang Meng Sim. The SKINNY Family of Block Ciphers and Its Low-Latency Variant MANTIS. In Matthew Robshaw and Jonathan Katz, editors, *Advances in Cryptology - CRYPTO 2016 - 36th Annual International Cryptology Conference,*

Santa Barbara, CA, USA, August 14-18, 2016, Proceedings, Part II, volume 9815 of *Lecture Notes in Computer Science*, pages 123–153. Springer, 2016.
Cited in this document on page 33.

[BNN+12] Begül Bilgin, Svetla Nikova, Ventzislav Nikov, Vincent Rijmen, and Georg Stütz. Threshold implementations of all 3×3 and 4×4 s-boxes. In Emmanuel Prouff and Patrick Schaumont, editors, *Cryptographic Hardware and Embedded Systems - CHES 2012 - 14th International Workshop, Leuven, Belgium, September 9-12, 2012. Proceedings*, volume 7428 of *Lecture Notes in Computer Science*, pages 76–91. Springer, 2012.
Cited in this document on page 40.

[BNN+15] Begül Bilgin, Svetla Nikova, Ventzislav Nikov, Vincent Rijmen, Natalia N. Tokareva, and Valeriya Vitkup. Threshold implementations of small s-boxes. *Cryptography and Communications*, 7(1):3–33, 2015.
Cited in this document on page 40.

[CHP+18] Carlos Cid, Tao Huang, Thomas Peyrin, Yu Sasaki, and Ling Song. Boomerang connectivity table: A new cryptanalysis tool. In Jesper Buus Nielsen and Vincent Rijmen, editors, *Advances in Cryptology - EUROCRYPT 2018 - 37th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Tel Aviv, Israel, April 29 - May 3, 2018 Proceedings, Part II*, volume 10821 of *Lecture Notes in Computer Science*, pages 683–714. Springer, 2018.
Cited in this document on page 27.

[CLL+14] Shan Chen, Rodolphe Lampe, Jooyoung Lee, Yannick Seurin, and John P. Steinberger. Minimizing the two-round Even-Mansour cipher. In Juan A. Garay and Rosario Gennaro, editors, *Advances in Cryptology - CRYPTO 2014 - 34th Annual Cryptology Conference, Santa Barbara, CA, USA, August 17-21, 2014, Proceedings, Part I*, volume 8616 of *Lecture Notes in Computer Science*, pages 39–56. Springer, 2014.
Cited in this document on page 35.

[CLP14] Benoit Cogliati, Rodolphe Lampe, and Jacques Patarin. The Indistinguishability of the XOR of k Permutations. In Carlos Cid and Christian Rechberger, editors, *Fast Software Encryption - 21st International Workshop, FSE 2014, London, UK, March 3-5, 2014. Revised Selected Papers*, volume 8540 of *Lecture Notes in Computer Science*, pages 285–302. Springer, 2014.
Cited in this document on page 37.

[CPGR05] Jim Chow, Ben Pfaff, Tal Garfinkel, and Mendel Rosenblum. Shredding your garbage: Reducing data lifetime through secure deallocation. In Patrick D. McDaniel, editor, *Proceedings of the 14th USENIX Security Symposium, Baltimore, MD, USA, July 31 - August 5, 2005*. USENIX Association, 2005.
Cited in this document on page 30.

[CRGB19] Lucian Cojocar, Kaveh Razavi, Cristiano Giuffrida, and Herbert Bos. Exploiting Correcting Codes: On the Effectiveness of ECC Memory Against Rowhammer Attacks. In *Proceedings of the IEEE Symposium on Security and Privacy (S&P) 2019. To appear*, 2019. Available from https://cs.vu.nl/~lcr220/ecc/ecc-rh-paper-sp2019-cr.pdf.
Cited in this document on page 20.

[DDKS13] Itai Dinur, Orr Dunkelman, Nathan Keller, and Adi Shamir. Key Recovery Attacks on 3-round Even-Mansour, 8-step LED-128, and Full AES^2. In Kazue Sako and Palash Sarkar, editors, *Advances in Cryptology - ASIACRYPT 2013 - 19th International Conference on the Theory and Application of Cryptology and Information Security, Bengaluru, India, December 1-5, 2013, Proceedings, Part I*, volume 8269 of *Lecture Notes in Computer Science*, pages 337–356. Springer, 2013.
Cited in this document on page 35.

[DDKS14] Itai Dinur, Orr Dunkelman, Nathan Keller, and Adi Shamir. Cryptanalysis of iterated Even-Mansour schemes with two keys. In Sarkar and Iwata [SI14], pages 439–457.
Cited in this document on page 36.

[DDKS15] Itai Dinur, Orr Dunkelman, Nathan Keller, and Adi Shamir. Reflections on slide with a twist attacks. *Des. Codes Cryptography*, 77(2-3):633–651, 2015.
Cited in this document on page 36.

[DEKM17] Christoph Dobraunig, Maria Eichlseder, Daniel Kales, and Florian Mendel. Practical key-recovery attack on mantis5. *IACR Transactions on Symmetric Cryptology*, 2016(2):248–260, 2017.
Cited in this document on page 33.

[DR02] Joan Daemen and Vincent Rijmen. *The Design of Rijndael: AES - The Advanced Encryption Standard*. Information Security and Cryptography. Springer, 2002. http://dx.doi.org/10.1007/978-3-662-04722-4.
Cited in this document on page 16.

[ECL+07] Reouven Elbaz, David Champagne, Ruby B. Lee, Lionel Torres, Gilles Sassatelli, and Pierre Guillemin. Tec-tree: A low-cost, parallelizable tree for efficient defense against memory replay attacks. In Pascal Paillier and Ingrid Verbauwhede, editors, *Cryptographic Hardware and Embedded Systems - CHES 2007, 9th International Workshop, Vienna,*

Austria, September 10-13, 2007, Proceedings, volume 4727 of *Lecture Notes in Computer Science*, pages 289–302. Springer, 2007.
Cited in this document on page 31.

[Eic18] Maria Eichlseder. *Differential Cryptanalysis of Symmetric Primitives*. PhD thesis, Graz University of Technology, March 2018. Available from https://graz.pure.elsevier.com/de/publications/differential-cryptanalysis-of-symmetric-primitives.
Cited in this document on pages 26 and 33.

[EK18] Maria Eichlseder and Daniel Kales. Clustering related-tweak characteristics: Application to mantis-6. *IACR Transactions on Symmetric Cryptology*, 2018(2):111–132, Jun. 2018.
Cited in this document on page 34.

[GJN+17] Jian Guo, Jérémy Jean, Ivica Nikolic, Kexin Qiao, Yu Sasaki, and Siang Meng Sim. Invariant Subspace Attack Against Midori64 and The Resistance Criteria for S-box Designs. *Transaction on Symmetric Cryptanalysis (FSE 2017)*, 2017.
Cited in this document on page 33.

[GMOP15] Andreas Gornik, Amir Moradi, Jürgen Oehm, and Christof Paar. A hardware-based countermeasure to reduce side-channel leakage: Design, implementation, and evaluation. *IEEE Trans. on CAD of Integrated Circuits and Systems*, 34(8):1308–1319, 2015.
Cited in this document on page 40.

[Gue16] Shay Gueron. Intel(r) software guard extensions (intel(r) sgx) memory encryption engine (mee). Talk at Real World Cryptography Conference 2016, 6-8 January 2016, Stanford, CA, USA, 2016. https://drive.google.com/file/d/0Bzm_4XrWnl5zOXdTcUlEMmdZem8/view.
Cited in this document on page 31.

[HJ02] William Eric Hall and Charanjit S. Jutla. Parallelizable authentication trees. *IACR Cryptology ePrint Archive*, 2002:190, 2002.
Cited in this document on page 31.

[HJ05] William Eric Hall and Charanjit S. Jutla. Parallelizable authentication trees. In Bart Preneel and Stafford E. Tavares, editors, *Selected Areas in Cryptography, 12th International Workshop, SAC 2005, Kingston, ON, Canada, August 11-12, 2005, Revised Selected Papers*, volume 3897 of *Lecture Notes in Computer Science*, pages 95–109. Springer, 2005.
Cited in this document on page 31.

[HJ08] William E. Hall and Charajit S. Jutla. US Patent US US7451310 B2: Parallelizable authentication tree for random access storage. http://www.google.com/patents/US7451310, November 2008.
Cited in this document on page 31.

[HSH+08] J. Alex Halderman, Seth D. Schoen, Nadia Heninger, William Clarkson, William Paul, Joseph A. Calandrino, Ariel J. Feldman, Jacob Appelbaum, and Edward W. Felten. Lest we remember: Cold boot attacks on encryption keys. In Paul C. van Oorschot, editor, *Proceedings of the 17th USENIX Security Symposium, July 28-August 1, 2008, San Jose, CA, USA*, pages 45–60. USENIX Association, 2008.
Cited in this document on page 30.

[HT13] Michael Henson and Stephen Taylor. Memory encryption: A survey of existing techniques. *ACM Comput. Surv.*, 46(4):53:1–53:26, 2013.
Cited in this document on page 26.

[IM18] Akiko Inoue and Kazuhiko Minematsu. Cryptanalysis of OCB2. Cryptology ePrint Archive, Report 2018/1040, 2018. https://eprint.iacr.org/2018/1040.
Cited in this document on page 7.

[Iwa18] Tetsu Iwata. Plaintext Recovery Attack of OCB2. Cryptology ePrint Archive, Report 2018/1090, 2018. https://eprint.iacr.org/2018/1090.
Cited in this document on page 7.

[Jen93] Robert J. Jenkins, Jr. ISAAC: a fast cryptographic random number generator, 1993. https://burtleburtle.net/bob/rand/isaacafa.html.
Cited in this document on page 28.

[Jen96] Robert J. Jenkins, Jr. ISAAC. In Dieter Gollmann, editor, *Fast Software Encryption, Third International Workshop, Cambridge, UK, February 21-23, 1996, Proceedings*, volume 1039 of *Lecture Notes in Computer Science*, pages 41–49. Springer, 1996.
Cited in this document on page 28.

[JNPS16] Jérémy Jean, Ivica Nikolic, Thomas Peyrin, and Yannick Seurin. Deoxys v1.4.1, October 2016. Available from: http://www1.spms.ntu.edu.sg/~syllab/m/index.php/Deoxys.
Cited in this document on pages 7, 20, 28, and 41.

[KDK+14] Yoongu Kim, Ross Daly, Jeremie Kim, Chris Fallin, Ji-Hye Lee, Donghyuk Lee, Chris Wilkerson, Konrad Lai, and Onur Mutlu. Flipping bits in memory without accessing them: An experimental study of DRAM disturbance errors. In *ACM/IEEE 41st International Symposium on Computer Architecture, ISCA 2014, Minneapolis, MN, USA, June 14-18, 2014*, pages 361–372. IEEE Computer Society, 2014.
Cited in this document on page 20.

[KGG+18] Paul Kocher, Daniel Genkin, Daniel Gruss, Werner Haas, Mike Hamburg, Moritz Lipp, Stefan Mangard, Thomas Prescher, Michael Schwarz, and Yuval Yarom. Spectre attacks: Exploiting speculative execution. *CoRR*, abs/1801.01203, 2018.
Cited in this document on page 33.

[KR11] Ted Krovetz and Phillip Rogaway. The software performance of authenticated-encryption modes. In Antoine Joux, editor, *Fast Software Encryption - 18th International Workshop, FSE 2011, Lyngby, Denmark, February 13-16, 2011, Revised Selected Papers*, volume 6733 of *Lecture Notes in Computer Science*, pages 306–327. Springer, 2011.
Cited in this document on pages 7 and 21.

[KSP05] Klaus Kursawe, Dries Schellekens, and Bart Preneel. Analyzing trusted platform communication. In *In: ECRYPT Workshop, CRASH – CRyptographic Advances in Secure Hardware*, page 8, 2005.
Cited in this document on page 30.

[LHW19] Muzhou Li, Kai Hu, and Meiqin Wang. Related-tweak statistical saturation cryptanalysis and its application on qarma. *IACR Transactions on Symmetric Cryptology*, 2019(1):236–263, Mar. 2019.
Cited in this document on page 35.

[LJ18] Rongjia Li and Chenhui Jin. Meet-in-the-middle attacks on reduced-round QARMA-64/128. *Comput. J.*, 61(8):1158–1165, 2018.
Cited in this document on pages 34 and 35.

[LM79] Walter Link and Herbert May. Eigenschaften von MOS-Ein-Transistorspeicherzellen bei tiefen Temperaturen. *Archiv für Elektronik und Übertragungstechnik*, 33:229–235, June 1979.
Cited in this document on page 30.

[LP07] Gregor Leander and Axel Poschmann. On the Classification of 4-Bit S-Boxes. In Claude Carlet and Berk Sunar, editors, *Arithmetic of Finite Fields, First International Workshop, WAIFI 2007, Madrid, Spain, June 21-22, 2007, Proceedings*, volume 4547 of *Lecture Notes in Computer Science*, pages 159–176. Springer, 2007.
Cited in this document on page 27.

[LRW02] Moses Liskov, Ronald L. Rivest, and David Wagner. Tweakable Block Ciphers. In Moti Yung, editor, *CRYPTO 2002*, volume 2442 of *Lecture Notes in Computer Science*, pages 31–46. Springer, 2002.
Cited in this document on pages 7, 26, and 36.

[LRW11] Moses Liskov, Ronald L. Rivest, and David Wagner. Tweakable Block Ciphers. *Journal of Cryptology*, 24(3):588–613, 2011.
Cited in this document on pages 7 and 36.

[LSG+18] Moritz Lipp, Michael Schwarz, Daniel Gruss, Thomas Prescher, Werner Haas, Anders Fogh, Jann Horn, Stefan Mangard, Paul Kocher, Daniel Genkin, Yuval Yarom, and Mike Hamburg. Meltdown: Reading kernel memory from user space. In William Enck and Adrienne Porter Felt, editors, *27th USENIX Security Symposium, USENIX Security 2018, Baltimore, MD, USA, August 15-17, 2018.*, pages 973–990. USENIX Association, 2018.
Cited in this document on page 33.

[Luc00] Stefan Lucks. The sum of prps is a secure PRF. In Bart Preneel, editor, *Advances in Cryptology - EUROCRYPT 2000, International Conference on the Theory and Application of Cryptographic Techniques, Bruges, Belgium, May 14-18, 2000, Proceeding*, volume 1807 of *Lecture Notes in Computer Science*, pages 470–484. Springer, 2000.
Cited in this document on page 37.

[Mar10] Luther Martin. XTS: A mode of AES for encrypting hard disks. *IEEE Security & Privacy*, 8(3):68–69, 2010.
Cited in this document on page 26.

[McC56] Edward J. McCluskey. Minimization of Boolean Functions. *Bell System Technical Journal*, 35(6):1417–1444, November 1956.
Cited in this document on page 27.

[Mer82] Ralph C. Merkle. US Patent US US4309569A: Method of providing digital signatures. https://patents.google.com/patent/US4309569, January 1982.
Cited in this document on page 30.

[MP15] Bart Mennink and Bart Preneel. On the XOR of Multiple Random Permutations. In Tal Malkin, Vladimir Kolesnikov, Allison Bishop Lewko, and Michalis Polychronakis, editors, *Applied Cryptography and Network Security - 13th International Conference, ACNS 2015, New York, NY, USA, June 2-5, 2015, Revised Selected Papers*, volume 9092 of *Lecture Notes in Computer Science*, pages 619–634. Springer, 2015.
Cited in this document on page 37.

[MV04] David A. McGrew and John Viega. The Security and Performance of the Galois/Counter Mode (GCM) of Operation. In Anne Canteaut and Kapalee Viswanathan, editors, *Progress in Cryptology - INDOCRYPT 2004, 5th International Conference on Cryptology in India, Chennai, India, December 20-22, 2004, Proceedings*, volume 3348 of *Lecture Notes in Computer Science*, pages 343–355. Springer, 2004.
Cited in this document on page 28.

[MVS00] Umesh Maheshwari, Radek Vingralek, and William Shapiro. How to build a trusted database system on untrusted storage. In Michael B. Jones and M. Frans Kaashoek, editors, *4th Symposium on Operating System Design and Implementation (OSDI 2000), San Diego, California, USA, October 23-25, 2000*, pages 135–150. USENIX Association, 2000.
Cited in this document on page 30.

[NIS18] NIST Cryptographic Technology Group. Submission requirements and evaluation criteria for the lightweight cryptography standardization process, August 2018. https://csrc.nist.gov/CSRC/media/Projects/Lightweight-Cryptography/documents/final-lwc-submission-requirements-august2018.pdf.
Cited in this document on pages 7 and 28.

[Pac18] Renaud Pacale. Probing attacks: On-board probing attacks. EURECOM Hardware Security Training, March 2018. http://soc.eurecom.fr/HWSec/lectures/probing/main.pdf.
Cited in this document on page 30.

[Pat08] Jacques Patarin. A proof of security in o(2n) for the xor of two random permutations. In Reihaneh Safavi-Naini, editor, *Information Theoretic Security, Third International Conference, ICITS 2008, Calgary, Canada, August 10-13, 2008, Proceedings*, volume 5155 of *Lecture Notes in Computer Science*, pages 232–248. Springer, 2008.
Cited in this document on page 37.

[Pat13] Jacques Patarin. Security in $o(2^n)$ for the Xor of Two Random Permutations - Proof with the standard H technique -. *IACR Cryptology ePrint Archive*, 2013:368, 2013.
Cited in this document on page 37.

[Per19] Leo Perrin. sboxu, 2019. Available from https://gforge.inria.fr/projects/sbox-utils/.
Cited in this document on page 27.

[Pet07] Torbjörn Pettersson. Cryptographic key recovery from Linux memory dumps. Presentation, Chaos Communication Camp, August 2007. https://media.ccc.de/v/cccamp07-en-2002-Cryptographic_key_recovery_from_Linux_memory_dumps.
Cited in this document on page 30.

[PM05] Thomas Popp and Stefan Mangard. Masked dual-rail pre-charge logic: Dpa-resistance without routing constraints. In Josyula R. Rao and Berk Sunar, editors, *Cryptographic Hardware and Embedded Systems - CHES 2005, 7th International Workshop, Edinburgh, UK, August 29 - September 1, 2005, Proceedings*, volume 3659 of *Lecture Notes in Computer Science*, pages 172–186. Springer, 2005.
Cited in this document on page 40.

[PM06] Thomas Popp and Stefan Mangard. Implementation aspects of the dpa-resistant logic style MDPL. In *International Symposium on Circuits and Systems (ISCAS 2006), 21-24 May 2006, Island of Kos, Greece*. IEEE, 2006.
Cited in this document on page 40.

[Poe18] Bertram Poettering. Breaking the confidentiality of OCB2. Cryptology ePrint Archive, Report 2018/1087, 2018. https://eprint.iacr.org/2018/1087.
Cited in this document on page 7.

[Pri10] Cyril Prissette. An Algorithm to List All the Fixed-Point Free Involutions on a Finite Set. http://arxiv.org/pdf/1006.3993v1.pdf, 2010.
Cited in this document on page 27.

[PS16] Thomas Peyrin and Yannick Seurin. Counter-in-tweak: Authenticated encryption modes for tweakable block ciphers. In Matthew Robshaw and Jonathan Katz, editors, *Advances in Cryptology - CRYPTO 2016 - 36th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 14-18, 2016, Proceedings, Part I*, volume 9814 of *Lecture Notes in Computer Science*, pages 33–63. Springer, 2016.
Cited in this document on page 7.

[QPS17] Qualcomm Product Security. Pointer Authentication on ARMv8.3 – Design and Analysis of the New Software Security Instructions. https://www.qualcomm.com/documents/whitepaper-pointer-authentication-armv83, January 2017.
Cited in this document on page 2.

[Qui52] Willard Van Orman Quine. The Problem of Simplifying Truth Functions. *The American Mathematical Monthly*, pages 521–531, October 1952.
Cited in this document on page 27.

[RBBK01] Phillip Rogaway, Mihir Bellare, John Black, and Ted Krovetz. OCB: a block-cipher mode of operation for efficient authenticated encryption. In Michael K. Reiter and Pierangela Samarati, editors, *CCS 2001, Proceedings of the 8th ACM Conference on Computer and Communications Security, Philadelphia, Pennsylvania, USA, November 6-8, 2001.*, pages 196–205. ACM, 2001.
Cited in this document on page 7.

[RCPS07] Brian Rogers, Siddhartha Chhabra, Milos Prvulovic, and Yan Solihin. Using Address Independent Seed Encryption and Bonsai Merkle Trees to Make Secure Processors OS- and Performance-Friendly. In *40th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO-40 2007), 1-5 December 2007, Chicago, Illinois, USA*, pages 183–196. IEEE Computer Society, 2007.
Cited in this document on page 31.

[Rog04] Phillip Rogaway. Efficient Instantiations of Tweakable Blockciphers and Refinements to Modes OCB and PMAC. In Pil Joong Lee, editor, *Advances in Cryptology - ASIACRYPT 2004, 10th International Conference on the Theory and Application of Cryptology and Information Security, Jeju Island, Korea, December 5-9, 2004, Proceedings*, volume 3329 of *Lecture Notes in Computer Science*, pages 16–31. Springer, 2004.
Cited in this document on page 2.

[Sage19] The Sage Developers. *SageMath, the Sage Mathematics Software System (Version 8.6)*, 2019. Available from https://www.sagemath.org.
Cited in this document on page 27.

[SD16] Mark Seaborn and Thomas Dullien. Exploiting the DRAM rowhammer bug to gain kernel privileges. Talk at Black Hat 2015, 5-6 August 2015, Las Vegas, NV, USA, 2016. https://www.blackhat.com/us-15/briefings.html.
Cited in this document on page 20.

[Sha00] Adi Shamir. Protecting smart cards from passive power analysis with detached power supplies. In Çetin Kaya Koç and Christof Paar, editors, *Cryptographic Hardware and Embedded Systems - CHES 2000, Second International Workshop, Worcester, MA, USA, August 17-18, 2000, Proceedings*, volume 1965 of *Lecture Notes in Computer Science*, pages 71–77. Springer, 2000.
Cited in this document on page 40.

[SI14] Palash Sarkar and Tetsu Iwata, editors. *Advances in Cryptology - ASIACRYPT 2014 - 20th International Conference on the Theory and Application of Cryptology and Information Security, Kaoshiung, Taiwan, R.O.C., December 7-11, 2014. Proceedings, Part I*, volume 8873 of *Lecture Notes in Computer Science*. Springer, 2014.
Cited in this document on pages 48 and 49.

[Sko02] Sergei P. Skorobogatov. Low temperature data remanence in static RAM. Tech. Rep. UCAM-CL-TR-536, June 2002. http://www.cl.cam.ac.uk/techreports/UCAM-CL-TR-536.pdf.
Cited in this document on page 30.

[SLG+16] Bing Sun, Meicheng Liu, Jian Guo, Vincent Rijmen, and Ruilin Li. Provable security evaluation of structures against impossible differential and zero correlation linear cryptanalysis. In Marc Fischlin and Jean-Sébastien Coron, editors, *Advances in Cryptology - EUROCRYPT 2016 - 35th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Vienna, Austria, May 8-12, 2016, Proceedings, Part I*, volume 9665 of *Lecture Notes in Computer Science*, pages 196–213. Springer, 2016.
Cited in this document on page 34.

[SNR+18] Gururaj Saileshwar, Prashant J. Nair, Prakash Ramrakhyani, Wendy Elsasser, Jose Joao, and Moinuddin K. Qureshi. Morphable counters: Enabling compact integrity trees for low-overhead secure memories. In *51st Annual IEEE/ACM*

International Symposium on Microarchitecture, MICRO 2018, Fukuoka, Japan, October 20-24, 2018, pages 416–427. IEEE Computer Society, 2018.
Cited in this document on page 31.

[TLS16] Yosuke Todo, Gregor Leander, and Yu Sasaki. Nonlinear Invariant Attack - Practical Attack on Full SCREAM, iSCREAM, and Midori64. In Jung Hee Cheon and Tsuyoshi Takagi, editors, *Advances in Cryptology - ASIACRYPT 2016 - 22nd International Conference on the Theory and Application of Cryptology and Information Security, Hanoi, Vietnam, December 4-8, 2016, Proceedings, Part II*, volume 10032 of *Lecture Notes in Computer Science*, pages 3–33, 2016.
Cited in this document on page 33.

[TV04] Kris Tiri and Ingrid Verbauwhede. A logic level design methodology for a secure DPA resistant ASIC or FPGA implementation. In *2004 Design, Automation and Test in Europe Conference and Exposition (DATE 2004), 16-20 February 2004, Paris, France*, pages 246–251. IEEE Computer Society, 2004.
Cited in this document on page 40.

[Win09] Johannes Winter. Eavesdropping Trusted Platform Module Communication. 4th European Trusted Infrastructure Summer school, ETISS, 2009. http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.464.6048.
Cited in this document on page 30.

[YEP+06] Chenyu Yan, Daniel Englender, Milos Prvulovic, Brian Rogers, and Yan Solihin. Improving Cost, Performance, and Security of Memory Encryption and Authentication. In *33rd International Symposium on Computer Architecture (ISCA 2006), June 17-21, 2006, Boston, MA, USA*, pages 179–190. IEEE Computer Society, 2006.
Cited in this document on page 31.

[YQC18] Dong Yang, Wen-Feng Qi, and Hua-Jin Chen. Impossible Differential Attack on QARMA Family of Block Ciphers. Cryptology ePrint Archive, Report 2018/334, 2018. https://eprint.iacr.org/2018/334.
Cited in this document on pages 34 and 35.

[ZD16] Rui Zong and Xiaoyang Dong. Meet-in-the-middle attack on QARMA block cipher. Cryptology ePrint Archive, Report 2016/1160, 2016. http://eprint.iacr.org/2016/1160.
Cited in this document on page 35.

[ZDW18] Rui Zong, Xiaoyang Dong, and Xiaoyun Wang. Milp-aided related-tweak/key impossible differential attack and its applications to qarma, joltik-bc. Cryptology ePrint Archive, Report 2018/142, 2018. https://eprint.iacr.org/2018/142.
Cited in this document on pages 34 and 35.