The Tweakable Block Cipher Family QARMAv2

Roberto Avanzi\textsuperscript{1,2}, Subhadeep Banik\textsuperscript{3}, Orr Dunkelman\textsuperscript{4}, Maria Eichlseder\textsuperscript{5}, Shibam Ghosh\textsuperscript{1}, Marcel Nageler\textsuperscript{5} and Francesco Regazzoni\textsuperscript{3,6}

\textsuperscript{1} Caesarea Rothschild Institute, University of Haifa, Israel, roberto.avanzi@gmail.com
\textsuperscript{2} Arm Germany, GmbH – roberto.avanzi@arm.com
\textsuperscript{3} Università della Svizzera Italiana, Lugano, Switzerland
subhadeep.banik@usi.ch, regazzoni@alari.ch
\textsuperscript{4} Computer Science Department, University of Haifa, Israel
orr@cs.haifa.ac.il, sghosh03@campus.haifa.ac.il
\textsuperscript{5} Graz University of Technology, Austria
maria.eichlseder@iaik.tugraz.at, marcel.nageler@iaik.tugraz.at
\textsuperscript{6} University of Amsterdam, The Netherlands – f.regazzoni@uva.nl

Abstract. We introduce QARMAv2, a redesign of the tweakable block cipher QARMA to provide more robust security bounds and allow for longer tweaks, while keeping very similar latency and area values. The longer tweaks serve to address specific use cases and facilitate the design of modes of operation with higher security bounds. This is achieved by adopting new key and tweak schedules, by revising the choice of S-Box and linear layer, and by making some changes to the 128-bit versions, as well as by performing a deeper security analysis.

The resulting cipher offers competitive latency and area in fully unrolled HW implementations in various processes.

Some of our results may be of independent interest. This includes new MILP models of certain classes of diffusion matrices, the comparative analysis of a full reflection cipher against an iterative half-cipher, and our boomerang attack framework.

Keywords: Tweakable Block Ciphers · Reflection Ciphers · Memory Encryption · Memory Integrity · Pointer Authentication · Short Hashes

1 Introduction

A significant current industry trend is strong isolation between mutually untrusted processes running on the same computing device: Intel’s SGX [Gue16] and TDX [Int21], AMD’s SEV [KPW16], and Arm’s CCA [MPS+21] provide access control mechanisms to protect the execution of programs from attacks originating in hostile peer and even higher-privileged software. Some of these technologies also defend against agents with physical access to the device. This is achieved by deploying cryptographic memory protection, such as encryption and integrity checks. This is an evolution of the well-established best practice of providing cryptographic memory hardening to smart cards, microcontrollers, and secure processors. It is driven by the security requirements of cloud computing services, online gaming platforms, and premium content distribution. In particular, during the last several years, several solutions with a small block size of 64 bits have been proposed, such as PRINCE [BCG+12], MANTIS [BJK+16], QARMA [Ava17], and PRINCE\textsuperscript{v2} [BEK+20].

One aspect that sets MANTIS and QARMA apart is that they are tweakable block ciphers [LRW02] (TBC): In addition to the secret key and a text, they accept a third input, called the tweak. The tweak, together with the key, selects the permutation computed by the cipher, but, unlike the key, it is assumed that the tweak may be under adversarial control. TBCs ease the design of modes of operation, and in fact one of their first applications
The Tweakable Block Cipher Family QARMAv2 has been to memory encryption modes [HT13]. QARMA also exists in a version with a block size of 128 bits and keys up to 256 bits, to meet the needs of general-purpose computing. The small version of QARMA is also used to implement PAC, the Pointer Authentication Code, a Control Flow Integrity mechanism on the Arm architecture [Arm16, QPS17].

QARMAv2 (pronounced “karma vee two”), is a redesign of QARMA to allow for longer tweaks and tighter security margins. It exists in two general-purpose versions with block lengths of \( b = 64 \) and 128 bits, denoted by QARMAv2-\( b \)-s, where \( s \) is the bit size of the key (more correctly, the claimed security level in bits), as well as in a slightly lighter 64-bit block version to be used for the PAC or memory integrity. For \( b = 128 \), the design allows for key sizes of \( s = 128 \), 192 or 256 bits. For \( b = 64 \), \( s \) is always 128 and can be omitted.

This redesign is not a whim: Since the introduction of QARMA, after many years of research but also trial and error, we have both achieved a better understanding of how to design block ciphers, and of the requirements on them coming from practical applications. The first understanding led, among other things, to a different key schedule. The second one led to the implementation of longer tweaks: We now explain why these are needed.

1.1 Rationale for Longer Tweaks

Let us start by looking at AES in XTS mode. It allows the use of a 128-bit nonce, which is encrypted, used to mask plaintext and ciphertext, and then updated via a LFSR for each subsequent block. A longer nonce must be hashed to 128 bits. Hence, care is due in order to prevent tweak reuse. Even worse, a counter mode like the GCM [MV04] uses a 96-bit IV, to which it appends a 32-bit counter. In these constructions the block size of the underlying cipher determines the space for the permutations that are parametrised, for a fixed key, by the tweak. Ciphers with 256-bit blocks have been suggested [JSV17], this allowing longer uncompressed IVs.

256-bit “native” tweaks allow the use of random or synthetic nonces/IVs without having to worry about repeated permutations. A tweaked codebook mode could be used in place of the XEX construction, where a 32-bit, resp. 64-bit counter is appended to a 224-bit, resp. 192-bit IV. For a counter mode, the whole 256-bit tweak can be used for the IV and the text input for the counter. Since collisions between 192 to 256-bit random values are negligible in practice, even longer tweaks are not needed. The 128-bit text input is not restricted and this allows messages up to \( 2^{64} \) blocks before the keystream becomes distinguishable from random bits. A 256-bit block size would allow up to \( 2^{128} \) blocks to be sent instead. However, such a long message is overkill. Instead, re-keying can be forced on a TBC with 128-bit blocks before \( 2^{64} \) blocks are encrypted without making it a too frequently occurring event. Note, also, that a counter-in-plaintext mode could still use some of the bits of a sufficiently large tweak to store a small counter overflow.

We also observe that a smaller block size carries advantages: linear and differential key recovery attacks are thwarted by first ensuring that the likelihood of a trail, resp. characteristic is small enough that it cannot be used as a distinguisher. This likelihood depends on the block size \( b \), and it should be \( 2^{-b} \) or less. If \( b \) is increased from 128 to 256, then, depending on the design, significantly more rounds may be needed to guarantee a 256-bit security level than the additional rounds needed to give the same security margin to the 128-bit cipher. Furthermore, the wider cipher may need more rounds to achieve full diffusion, meaning that any distinguisher can be extended by more rounds. Everything else being similar, this means increased latency, area, and power consumption.

Future-proofing memory-encryption use cases also support the availability of 256-bit tweaks in QARMAv2-128. Even without bringing 128-bit addressing into the picture, just by including 48- or 52-bit addresses, 56-bit counters and, say, 32-bit process domain IDs in the tweak, we already need more than 128 bits. For QARMA-64, however, 128-bit tweaks are likely to be sufficient for most other embedded applications. For PAC, 128-bit tweaks...
are needed for improved versions of the feature where the tag on a pointer is computed using more context information than, say, the current stack pointer.

1.2 Use Cases, Security Model, and Their Implications

The goal of the QARMAv2 design is to provide a general-purpose TBC that is also ideally suitable to memory encryption and fast computation of short tags. The design allows efficient hardware implementations, but at the same time optimized software implementations should be straightforward (for instance, using bitslicing on the 16 or 32 parallel instances of the same S-Box).

Several constructions of TBCs from non-tweakable block ciphers exist [Rog04, LRW11, LST12, Men15, JLM+17, Men18, JN20], but they all vastly increase latency.

A different school of thought modifies the design of the cipher itself, such as the TWEAKEY framework [JNP14] or TNT [BGGS20], achieving lower overheads. Our approach is close in spirit to the latter two. However, we should note that TWEAKEY unifies key and tweak, treating the resulting tweakey as a single undifferentiated quantity. While this gives total flexibility to adjust the relative sizes of the TWEAKEY’s secret and public parts, we argue that real-world security requirements on key and tweak are simply too different for such an approach to be optimal in practice. In most applications we can assume that the key, secret, is not changed often. Thus, its generation can be physically hardened without a significant impact on a system’s performance. Therefore we can assume that the adversary will not be able to significantly alter the key value, and we do not need to include related-value attacks. On the other hand, the public tweak is changed often and its security model includes known or even chosen tweaks. As a result, related-value attacks must be considered. If a common schedule is used for both key and tweak, it may necessitate to take into account related-value attacks for both, leading to an overestimation of the required number of rounds to achieve the desired level of security.

These considerations prompt us to keep separate key and tweak schedules, as in QARMA. We keep a very simple key schedule, where orthomorphisms and round constants are used to harden against various attacks and pose hurdles to key recovery. However, devising a suitable tweak schedule, that achieves the desired security goals while not critically affecting latency and area, is the most critical part of the design of QARMAv2.

1.3 Results

The TBC QARMAv2 is a reflector construction, i.e. it is split in two halves, where a round function is iterated in the first half and its inverse in the second half, with another simple function in the middle. This allows the use of a single function for both encryption and decryption. In fact, we kept the data path as similar as possible to QARMA’s, a structure that has been proven so far very resistant to cryptanalysis. Despite the similarities, the new design provides significantly improved security levels with respect to QARMA, as well as better suitability for practical applications through the availability of longer tweaks.

This is achieved through the introduction of new key and tweak schedules, a re-definition of the 128-bit version, a deeper analysis of the properties of some of the building blocks, and a more thorough cryptanalysis in general. Just as QARMA can reuse part of the cryptanalysis of MIDORI and MANTIS, QARMAv2 reuses parts of the cryptanalysis of QARMA.

The new tweak schedule updates the tweak in the same way through the whole cipher. In this way, any relation between the tweak updates and the round function will have to work with the inverse round function as well, reducing its likelihood.

The design has conservative parameters (for instance, we could have chosen a lighter S-Box with slightly worse cryptographic properties, or used a few less rounds), but we feel that we have still improved its competitiveness since the tweak length has been increased to a much more useful value while not significantly impacting performance.
Some aspects of our security analysis have independent value:
1. We analyze both half-cipher and full-cipher and therefore we can assess the impact of the reflector construction;
2. The new MILP model for our Almost-MDS matrix can prove useful also in the study of other ciphers that use a similar matrix; and
3. We propose full MILP models for various attacks, and some of them are, to the best of our knowledge, applied to reflector constructions for the first time, e.g., for boomerang distinguishers.

QARMAv2, just as its predecessor, offers significantly better energy-per-bit values than most ciphers in fully unrolled implementations, including the AES. This makes it ideal for always-on applications such as memory encryption. The cipher is flexible enough to be deployed for general purpose usage, with variable security levels. Finally, we also allow the use of 128- and 192-bit keys for QARMAv2-128, besides 256-bit keys.

Outline of the Paper. In Section 2, we present the specification of QARMAv2. In Section 3, we discuss how the components have been chosen. In Section 4, we evaluate the resistance of the design to various cryptanalytic approaches. In Section 5, we provide hardware implementation results. Finally, in Section 6 we conclude and state open questions.

2 Definition of the Cipher

2.1 Structure

The design of QARMAv2 follows the reflector construction of PRINCE, MANTIS and QARMA, represented in Fig. 1: It is the composition of a forward function, of a symmetric central construction (also called the reflector), and of a backward function. The latter is the functional inverse of the forward function. This allows to use the same circuit for both encryption and decryption with a relatively minor set-up step. In Fig. 1, we represent the initial and final rounds separately: They consist of just a key addition and a layer of identical S-Boxes, and are not tweaked. The central construction is also not tweaked. The values $K^{(i)}$, resp. $T^{(i)}$ are derived from the key $K$, resp. tweak $T$ by simple operations.

The function $F$ is an iterated cipher with a keyed and tweaked round function $f$. The last operation of the round function, and thus also the last of $F$, is an S-Box layer.

A bar over a function denotes its inverse, for instance the inverse of $f$ is $\bar{f}$.

We give the Algorithm as pseudo-code in Fig. 2 on the facing page, and graphically in Fig. 3 on page 7 for odd $r$. The notation is explained next.

2.2 Cells, Blocks, Layers and the Internal State

In Fig. 2 on the facing page, $\mathcal{F}$ is the internal state of the cipher. It is $b$ bits long. A $b$-bit value is called a block and is represented as a three-dimensional array, consisting of...
A permutation \( \pi \)

It is understood that

of layer number

2.3 Permutations and Shuffles

The number \( \ell \) can take the values 1 and 2. Each layer is a \( 4 \times 4 \) matrix of cells, and it can also be viewed as an array of 16 elements:

\[
\mathcal{L} = c_{0}^{(i)} \parallel c_{1}^{(i)} \parallel \cdots \parallel c_{14}^{(i)} \parallel c_{15}^{(i)} = \begin{pmatrix}
    c_{0} & c_{1} & c_{2} & c_{3} \\
    c_{4} & c_{5} & c_{6} & c_{7} \\
    c_{8} & c_{9} & c_{10} & c_{11} \\
    c_{12} & c_{13} & c_{14} & c_{15}
\end{pmatrix} = \begin{pmatrix}
    c_{0,0} & c_{0,1} & c_{0,2} & c_{0,3} \\
    c_{1,0} & c_{1,1} & c_{1,2} & c_{1,3} \\
    c_{2,0} & c_{2,1} & c_{2,2} & c_{2,3} \\
    c_{3,0} & c_{3,1} & c_{3,2} & c_{3,3}
\end{pmatrix}.
\]

It is understood that \( c_{i,j} = c_{4i+j} \). With respect to cell numbering, the \( b \) bits of a block are indexed in big endian order: Hence, bits \( b-1, \ldots, b-4 \) are contained in the first cell of layer number 0, and bits 3, \ldots, 0 are in the fifteenth cell of the last layer. The bits in a cell are indexed in little endian order. The data obfuscation path is one block wide.

2.3 Permutations and Shuffles

A permutation \( \pi \) on \([0, \ldots, 15]\) acts on a layer as follows:

\[
(\pi(\mathcal{L}))_{i} = c_{\pi(i)} \quad \text{for} \quad 0 \leq i < 16.
\]

Our choice for the state shuffle \( \tau \) is MIDORI’s shuffle

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

i.e. it acts on each layer as follows

\[
\mathcal{L} = \begin{pmatrix}
    c_{0} & c_{1} & c_{2} & c_{3} \\
    c_{4} & c_{5} & c_{6} & c_{7} \\
    c_{8} & c_{9} & c_{10} & c_{11} \\
    c_{12} & c_{13} & c_{14} & c_{15}
\end{pmatrix} \mapsto \begin{pmatrix}
    c_{0} & c_{11} & c_{6} & c_{13} \\
    c_{10} & c_{1} & c_{12} & c_{7} \\
    c_{5} & c_{14} & c_{3} & c_{8} \\
    c_{15} & c_{4} & c_{9} & c_{2}
\end{pmatrix} = \tau(\mathcal{L}).
\]
The action of a permutation on $[0, \ldots, 16\ell - 1]$ on a block can be similarly defined using a matrix with 4 columns and $4\ell$ rows. For $\ell = 2$, a permutation on $[0, \ldots, 15]$ acts on all the layers in a block in parallel.

2.4 The Round Functions

QARMAv2 reuses the round functions of the original QARMA, with a single change required when using two layers. There are four types of round: the full round and the half round, and their inverses. A full round has the following structure:

$$k
x \tau M S X y \text{ i.e. } x \tau R X y,$$

where $R = S \circ M \circ \tau$, The letters $k$, resp. $t$ denote a round key, resp. tweak block, and $\tau$ is the StateShuffle operation, defined as a permutation on 16 elements, acting on the state layer-wise. The $4 \times 4$ matrix $M$ operates column-wise by left multiplication on each layer of a block. $S$ is the parallel application of $16\ell$ identical S-Boxes to the $16\ell$ cells of the state. $X$ denotes the eXchangeRows operation, which applies only for $\ell = 2$. The eXchangeRows operation swaps the first two rows between the two layers. It performed every other full round, where two eXchangeRows’s always flank the central construction (see below). $X$ and $S$ clearly commute. The half round function consists only of a round key addition and a S-Box layer and is used only for the first and last rounds of the cipher.

2.5 The Central Construction

The central construction is similar to that of QARMA:

$$x \tau M \tau y,$$

where $w_0, w_1$ are two round key blocks. This function can be implemented using the original QARMA central construction, by using the round key $M \cdot w_0 + w_1$.

2.6 The Round Constants

In Section 3.5 on page 13 we define the sequence $(c_i)_{i=0}^\infty$ of single block wide round constants with $c_0 = c_1 = 0$, as well as an additional constant $\alpha$.

2.7 The Round Keys

First, the key $K = K_0||K_1$ is split into two halves $K_0$ and $K_1$. Then, put $W_0 = \sigma^2(K_0)$ and $W_1 = \sigma^{-2}(K_1)$.

The schedule can be implemented by two registers $R_0$ and $R_1$ that are alternatingly added to the state. For encryption they are initialized with the values $K_0$ and $K_1$. During the central construction, they are subject to the transformation

$$\iota : (K_0, K_1) \mapsto (L_0, L_1) := (\sigma(K_0) + \alpha, \sigma^{-1}(K_1 + \alpha)),$$
where $\sigma$ is the PRİNCERE orthomorphism (5), defined in Section 3.1 on the following page. Then, $L_0$ and $L_1$ are the round keys alternatingly added to the backward rounds.

Note that the choice of the indexes of the $W_i$ and $L_i$ reflects whether the round keys are derived from $K_0$ or $K_1$. The tweak blocks $T_0$ and $T_1$ will follow the same indexing.

If $R_0$ and $R_1$ are instead initially set to the values $L_1 = \sigma^{-1}(K_1 + \alpha)$ and $L_0 = \sigma(K_0) + \alpha$ respectively, the same transformation will yield
\[
\iota((L_1, L_0)) = \iota(\sigma^{-1}(K_1 + \alpha, \sigma(K_0) + \alpha)) = (\sigma(\sigma^{-1}(K_1 + \alpha)) + \alpha, \sigma^{-1}(\sigma(K_0) + \alpha + \alpha)) = (K_1, K_0).
\]

The values $R_i$ for decryption are thus obtained by starting with $R_0 = L_1$ and $R_1 = L_0$ in place of $K_0$ and $K_1$, with the schedule exactly as for the encryption, provided we properly set up $W_0$ and $W_1$. The latter is easily done: since for encryption we have $W_0 = \sigma^2(K_0)$ and $W_1 = \sigma^{-2}(K_1)$, for decryption we just swap the values, i.e. $W_1 = \sigma^2(K_0)$ and $W_0 = \sigma^{-2}(K_1)$. We cannot yet use the same circuit for encryption and decryption, because we have not yet taken the tweaks into account. This is done next.

### 2.8 The Tweak Schedule

We define a tweak-alternating tweak schedule for a two-block tweak, where each tweak block is updated after each use. The update functions are applied during the forward and backward rounds unchanged, i.e., without undoing the update steps as in QARMA. Let $t_i$ be the round tweak added at round number $i$. The schedule is the following: for each integer $i \geq 2$ we set $t_{2i} = \varphi^{-1}(t_{2i-2})$ and $t_{2i+1} = \varphi(t_{2i-1})$ for some function $\varphi$. Starting with $t_1 = T_1$ and $t_2 = \varphi^{-1}(T_0)$, we obtain the sequence
\[
[T_1, \varphi^{-1}(T_0), \varphi(T_1), \varphi^{-2}(T_0), \varphi^2(T_1), \varphi^{-3}(T_0), \ldots, \varphi^{r-1}(T_1), T_0]
\]
where each value in the sequence is used in a successive round from Round 1 to Round $2r$. Swapping $T_1$ and $T_0$ and applying the same sequence of transformations gives the reverse schedule. Combining this with the inversion of the key schedule, the same circuit can be used for encryption and decryption. In Section 3.4 on page 10, the following tweak shuffles are chosen: $\varphi = \tau_f$ for $\ell = 1$, and $\varphi = \tau_f$ for $\ell = 2$.
2.9 The Complete Circuit

The circuit has 7 input registers and one output register, all one block wide. The inputs are the two key blocks, the two center key blocks, the two tweak blocks, and the input text (plaintext $PT$ or ciphertext $CT$). The output is the output text (ciphertext $CT$ or plaintext $PT$, respectively). For a key $K = K_1 || K_0$ and a tweak $T = T_1 || T_0$, where the $K_i$ and the $T_i$ are blocks, we have:

- For encryption, the input vector is set as follows, with $W_0 = o^2(K_0)$, $W_1 = o^{-2}(K_1)$:
  \[
  [K_0, K_1, W_{r+1} \mod 2, W_r \mod 2, \phi^{r-1}(T_0), T_1, PT].
  \]

- For decryption, the input vector is set to
  \[
  [o^{-1}(K_1 + \alpha), o(K_0) + \alpha, W_r \mod 2, W_{r+1} \mod 2, \phi^{-1}(T_1), T_0, CT].
  \]

The input vector for encryption and decryption can be set up by a simple wrapper circuit. If a system uses a single key (or rarely changes it) and relies on tweaks to differentiate uses, then the values $W_0, W_1, L_0, and L_1$ can be precomputed and cached.

The only component of the cipher which has not yet been defined yet is the S-Box. This is done in Section 3.6 on page 13.

2.10 Stretching Shorter Keys

For $\ell = 1$, the only admissible key size is 128 bits.

For $\ell = 2$, we note that the encryption algorithm is always defined with two full 128-bit inputs for the key, i.e. for a key of 256 bits. Only the security margins change, together with the value of the parameter $r$, as defined later in Section 4.9. Therefore, also the test vectors given in Appendix H on page 47 use full 256-bit key values.

This said, we define procedures to stretch 128 and 192 bit keys to 256 bits. For a 128-bit key $K$, set $K_0 = K_1 = K$. For a 192-bit key $K$, write $K = Y_0 || Y_1 || Y_2$ where the $Y_i$ are 64-bit values, and then put $K_0 = Y_0 || Y_1$ and $K_1 = Y_2 || (\text{MAJ}(Y_0, Y_1, Y_2) \gg 17)$, where \text{MAJ} is the majority function.

2.11 Stretching Single Block Tweaks

The single block length tweak case is handled by setting $T_1 = \phi(T_0)$, where $T_0 = T$. This “ties” the two tweak blocks just before the centre for encryption, i.e. the tweak values added at Rounds $r - 1$ and $r$ are equal.

We tried various approaches to derive the second tweak block from the first, and we have chosen the approach that gives the best bounds (cf. Tables 10 and 11 on page 38).

However, since the difference in rounds to achieve 32 or 64 active cells is not radically different from the general case, in the cryptanalysis we evaluate the security of the cipher only for independently active tweak blocks. This provides a lower bound for the security also when additional constraints on the tweaks apply. This said, we shall evaluate whether it is a valid option to reduce the number of rounds by 2 for the single-block tweak case.

3 Choices of the Functions

3.1 The Orthomorphism

Let $b = 64 \ell$. The following map is an orthomorphism, defined in PRINCE:

\[
os(w) := (w \gg 1) + (w \gg (b - 1)).\]
The operator “+” on keys, tweaks, and states shall always denote the binary XOR.

A useful property of orthomorphisms is following one:

**Theorem.** Over characteristic 2 algebras, if \( o() \) is an orthomorphism, then so is \( o^2() \).

**Proof:** First, note that \( o^2() \) is clearly a bijective linear map. By definition \( p(x) = x + o(x) \)

is a bijection, and so is its iterate: \( p^2(x) = (x + o(x)) + o(x + o(x)) = x + o(x) + o(x) + o^2(x) = x + o^2(x) \). Hence \( o^2() \) is an orthomorphism.

Therefore, \( o^4(), o^8(), o^{16}() \) are orthomorphisms as well. In particular, none of these maps can be the identity map. If they operate on a finite algebra, there are only finitely many such maps. This implies also that for \( i \neq j \), \( o^i(x) \mapsto o^j(x) \) is an orthomorphism or the identity, and \( x \mapsto o^i(x) + o^j(x) \) is a linear bijection or the zero map. Such functions can be used for key derivation to make expected differences between round keys uniform and collision-free. This will reduce correlations between similar functions, for instance in Even-Mansour constructions [EM91], or reduce the likelihood of self-differentials that could be used in reflection attacks [Kar08, SBY15].

### 3.2 The Diffusion Matrix

We considered the same Almost-MDS matrices as in QARMA. Let \( R_2 \) be the quotient ring \( \mathbb{F}_2[X]/(X^4+1) \), and \( \rho \) the image of \( X \) in \( R_4 \). We have \( \rho^4 = 1 \), and thus \( \{1, \rho, \rho^2, \ldots, \rho^{m-1}\} \) is a basis for \( R_m \) as an \( \mathbb{F}_2 \)-algebra. With respect to this basis, multiplication by \( \rho \) in \( R_m \) corresponds to a simple circular left rotation of the coordinates of a field element, given as individual bits in a cell. We define

\[
\text{circ}(0, \rho^a, \rho^b, \rho^c) = \begin{pmatrix}
0 & \rho^a & \rho^b & \rho^c \\
\rho^c & 0 & \rho^a & \rho^b \\
\rho^b & \rho^c & 0 & \rho^a \\
\rho^a & \rho^b & \rho^c & 0
\end{pmatrix}.
\]

(6)

For the design of QARMAv2, besides the MIDORI circulant \( M_0 := \text{circ}(0, 1, 1, 1) \) we considered

\[
M_{1,1} = \text{circ}(0, \rho, \rho^2, \rho^3) \quad \text{and} \quad M_{4,2} = \text{circ}(0, \rho, \rho^2, \rho)
\]

which are all involutory. Following the QARMA paper, they are grouped into classes depending on their transition patterns: **Class I** includes \( M_0 \) and \( M_{1,1} \); and \( M_{4,2} \) is a **Class II** matrix. Their transition patterns are displayed in Fig. 8 on page 34 in the Appendix, next to a simpler XOR model. Class I matrices have a slightly better diffusion model, but in QARMA Class II matrix \( M_{4,2} \) was chosen on the basis of certain heuristics. For QARMAv2 we use Class I matrix \( M_{4,1} \) instead, validating this choice with a deeper cryptanalysis.

### 3.3 The StateShuffle

The MIDORI StateShuffle [BBI15] satisfies a very important property (that we have not found mentioned elsewhere): the four cells in each column are mapped by \( \tau \) to pairwise different columns and rows, and the same holds for the four cells in each row. This property is satisfied by \( \tau \) as well. Thus, the four elements in each row (or column) of a state after applying \( \tau \) come from four different columns and from four different rows of the state before \( \tau \) was applied. Consequently, a following MixColumns does not mix any two elements from the same column (or row) of the state before \( \tau \). (And in a cipher design one could also use a “mix rows” transformation as well.) The mapping has order 4, which explains why the Conditions 2 and 3 of the MIDORI paper [BBI15] are satisfied.

Let us consider the square function of \( \tau \),

\[
\tau^2 : [0, 8, 12, 4, 3, 11, 15, 7, 1, 9, 13, 5, 2, 10, 14, 6].
\]
It is essentially the transpose of the state, preceded and followed by permutations of the rows that are the inverse of each other.

Now, in [ABI+18] alternative state shuffles have been found that produce linear trails of higher weight than \( \tau \). So it is natural to ask why we did not switch to the improved shuffles ourselves. The main reason is that, in studying the interaction of state shuffle and tweak shuffle, we make use of the above strong property — in fact for both columns and rows. Also, we privilege shuffles that give a faster full diffusion, so we restrict to those that give a 3-round full diffusion property (when \( \ell = 1 \)) together with the chosen diffusion matrix. All this leads back to the permutations studied in M\textsc{idori}. For the limited number of rounds that we consider, the chosen \( \tau \), i.e. (2), is in any case almost always optimal or nearly optimal. Therefore, we focus our attention instead on the tweak shuffles.

### 3.4 The Tweak Shuffles

A TBC requires more rounds than a non-tweakable block cipher based on the same round function to attain a comparable security level. The reason is that the tweak gives an adversary more control. Even comparing related-key to related-tweak attacks, the first are in practice much more difficult to mount, to the point that they are often ignored when assessing the security, say, of the AES. But, related-tweak attacks cannot be ignored.

Estimating the number of active cells in linear trails and differential characteristics plays an important role when choosing the tweak schedule. Evaluations of ShiftRows alternatives exist up to a very high number of rounds [ABI+18]. However, these only consider simple linear trails, or non-related-key, non-related-tweak differential characteristics. These active cell counts are usually performed using Matsui’s Algorithm 1 [Mat94]. We modified it to include tweaks. However, the much higher number of initial states and possible transitions make the algorithm far less efficient, to the point that MILP solvers [MWGP11] or SAT solvers such as CryptoMiniSAT [SNC09] are significantly faster.

#### 3.4.1 Single Layer Blocks

The M\textsc{antis}/Q\textsc{arma} tweak shuffle was the result of an extended search on a subset of all permutations on the sixteen cells. The active cell count on a half round and five full rounds for “several thousand choices for the permutation \( h \)” [BJK+16] were determined using a MILP model. Among the shuffles reaching the maximum cell count of 16, the one was chosen maximizing the active cells in M\textsc{antis}_5 first, and then in M\textsc{antis}_6. The resulting shuffle \( h_0 = h \) is

\[
h_0 : \begin{bmatrix} 6, & 5, & 14, & 15, & 0, & 1, & 2, & 3, & 7, & 12, & 13, & 4, & 8, & 9, & 10, & 11 \end{bmatrix},
\]

which the product of two cycles of lengths 2 and 14 and has a period of 14.

Q\textsc{arma} and Q\textsc{arma}v2 have very different tweak schedules. It is not obvious whether Q\textsc{arma}’s \( h \) is optimal also for Q\textsc{arma}v2, and we may want to pick a better shuffle if not.\footnote{Furthermore, under \( h \) two tweak cells are only combined with the two corresponding cells of the key, creating a partitioning of the tweak and key bits with a very small group, resulting in low weight algebraic relations between these few bits. Even though we could not find any exploitable such relation, we want to find alternative shuffles with a single cycle or fewer, longer cycles.}

The search is aided by a cell-wise MILP model of the cipher, which is run for various choices of \( r \) and of the tweak shuffle. With respect to Q\textsc{arma}, Q\textsc{arma}v2’s longer tweak translates to a higher number of variables and relations in MILP models. As a consequence, these increasingly take a significantly longer time to solve than Q\textsc{arma}’s models. Hence, we can only test fewer shuffles in the same time and we have to rely more on heuristics. After running several experiments, we observe that:
1. Shuffles with fewer and longer cycles, as well as schedules without structured, fixed or repeated state patterns, seem to lead to heavier optimal characteristics. In fact, if these conditions hold, self-cancellation conditions are expected to repeat less often.

2. Similarly, shuffles that map aligned groups of cells – i.e. cells that lie either in the same column or in the same row – to aligned groups as much as possible also seem to lead to heavier optimal characteristics.

With this intuition we immediately found a promising family of shuffles. These are of length 16 and are obtained by permuting all rows cyclically, for instance by sending row $i$ to row $i + 1$, with the index considered modulo 4, and then applying a simple rotation to a single row (usually the top one), or three distinct cell swaps involving just two rows, similarly to $h$. Of these the following stands out after our lengthy MILP runs:

$$h_4 : [13, 14, 15, 12, 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11]$$

of order 16. Table 8 on page 37 is a comparison of $h$, this map and other permutations.

A consideration of cancellation patterns leads to a better family of shuffles. Since we use an alternating tweak schedule, the idea is to consider how a cell affects the state after two rounds. So, let $t_n$ be the round tweak added at round $n$, such that $t_{n+2} = \varphi(t_n)$ is added at round $n + 2$. Now, suppose the $i$-th cell of the state is activated by the tweak addition in round $n$. In order to avoid self-cancellations, the tweak shuffle should not map the $t_n$’s $i$-th cell to a position $\varphi^{-1}(i)$ which, in the state, is affected by the $i$-th cell of the state in round $n + 2$. In Fig. 4a on the next page we show how an active single cell evolves through two forward rounds, for all sixteen cells in a layer. We see that after two rounds, seven cells are still unaffected. Ideally, a tweak shuffle $\varphi$ should map each cell to one of the seven cells it does not affect.

Note that, even if $\varphi$ is optimal in this sense, the map $\varphi^{-1}$ used to derive $t_{n+3}$ from $t_{n+1}$ is not guaranteed to be optimal. In fact, most likely it will not be, but we conjecture that $\varphi^{-1}$ behaves like a randomly chosen permutation. The tweak schedule progresses beyond the central construction unmodified, so $\varphi^{-1}$ becomes the “optimal” map and $\varphi$ the “mostly randomly-behaving” one. As we shall see, this strategy seems to be working.

In Fig. 4b we display how $\tau$ transforms entire rows and columns. Beside the remark already made in Section 3.4 that $\tau^2$ maps rows to columns (and vice versa) we note also that: The square of the composite map $M \circ \tau$ lets each cell of a given row act on only nine cells of the state, leaving a fixed column always unaffected, and any two cells in the same row will affect all twelve cells in the three columns affected columns. This unaffected column is the very one to which the cells of the row are mapped under $\tau^2$.

This seems to suggest to use $\tau^2$ as the tweak shuffle. However, this does not work well, because $\tau^2$ is an involution, leading to many fixed states and short iterative characteristics. Note that we could permute the cells of each column of $\tau^2$ independently, leading to $(4!)^4 = 331776$ different shuffles for which the no-self-cancellation property still holds. This is too large a search space. Hence, we proceed as follows:

1. We consider the compositions of $\tau^2$ with the 24 row permutations. Their order is at most 8 and only six of them decompose as a product of two cycles of length 8.

2. Among these six order 8 shuffles, we select the permutation $\hat{\tau}$, with the highest active cell counts in related-tweak characteristics for the first few values of $r$.

3. We look at the order 16 cyclic shuffles obtained by composing $\hat{\tau}$ with a single cell swap within a column. This produces another set of just six shuffles, $\tau_{d1}$ to $\tau_{d6}$.

4. Among these, we pick the shuffle such that the cell counts for optimal related-tweak characteristics of the shuffle, and in then for its inverse, are maximal. Considering the inverse is equivalent to inverting the order of application of $\varphi$ and $\varphi^{-1}$. All cell counts are reported in Table 8 on page 37.
As a result, we choose

$$\hat{\tau} = \tau_{d4} : \begin{bmatrix} 2, 10, 14, 6, & 1, 9, 13, 5, & 0, 8, 12, 4, & 3, 11, 15, 7 \end{bmatrix}.$$  

Finally, amongst the order 16 shuffles obtained by applying a cell swap to $\tau_{d4}$, we again select the one with the best weights in related-tweak differential characteristic for full-cipher models with $r = 5$, then 6. This shuffle is obtained from $\tau_{d4}$ by swapping the topmost two cells in the first column:

$$\tau_f : \begin{bmatrix} 1, 10, 14, 6, & 2, 9, 13, 5, & 0, 8, 12, 4, & 3, 11, 15, 7 \end{bmatrix}.$$  

(9)
3.4.2 Two-Layer Blocks

The chosen function is derived from $\tau^2$ as well. We start with $\tau_f$, acting on each layer in parallel, then we swap some cells in the same position between the the two layers, provided the order of the resulting shuffle is 32. In our experiments, swapping about a half of the cells leads to better shuffles, but we must swap an odd number of cells to get an order 32 map. We obtain the best results among the sampled shuffles by exchanging the full second and fourth rows, and the cells in position 3 and 19:

$$
\tau_F : \begin{bmatrix}
1, 10, 14, 22, 18, 25, 29, 21, 0, 8, 12, 4, 19, 27, 31, 23, \\
17, 26, 30, 6, 2, 9, 13, 5, 16, 24, 28, 20, 3, 11, 15, 7
\end{bmatrix}.
$$

(10)

3.4.3 Full Diffusion Properties

Theorem 1. In QARMAv2 with $\ell = 1$, any input bit nonlinearly affects all bits of the state after three rounds, intended as the first half round, two full rounds, and the diffusion layer of the following one.

For $\ell = 2$, any input bit nonlinearly affects all bits of the state after four full rounds.

The first claim is inherited from QARMA. The second is proved in Appendix E on page 44.

3.4.4 Two-Layer Blocks and QARMA-128

Remark 1. The $m = 4, \ell = 2$ construction is just a variant of QARMA’s construction of the 128-bit cipher, with a single layer and 8-bit cells. Upon pairing cell $i$ with cell $i + 16$ in the state and tweak blocks, i.e. a cell in the first layer with the corresponding one in the second layer, we see that all operations work on these 8-bit cells. In particular, $\tau_F$ operates like $\tau_f$ followed by a 4-bit rotation of 9 cells.

This design requires four rounds for full diffusion instead of three, so from this point of view it is less efficient than QARMA-128. However, it allows us to count the active nibbles instead of the active bytes, getting better security margin estimates.

3.5 The Round Constants

To facilitate lightweight implementations in SW or even round-based implementations in HW, the QARMAv2 constants are generated programmatically, using a Galois LFSR seeded by the first 16 digits of the hexadecimal expansions of the fractional part of $\pi$ and $e$.

The 64 most significant bits of the round constant $RC_2$ (the first non-zero round constant) are given by 0x243F6A885A308D3ULL, and each successive value is obtained from the LFSR’s state after applying the LFSR 23 times. We call this update function $\Psi$. 128-bit values are generated by concatenating two consecutive values, the first one being the most significant. Round constants are given in big endian order.

For the 64-bit cipher, $\alpha = 0x13198A2E03707344ULL$. For the 128-bit cipher $\alpha$ is the concatenation of this value with its image under $\Psi$.

The Galois LFSR is defined by the primitive polynomial $X^{64} + X^{50} + X^{33} + X^{19} + 1$ over GF(2). The polynomial was chosen first to allow a mixing of all the bits each time, to reduce repeated subsequences, and then to make its implementation as easy as possible in software, even on 16 bit microcontrollers. Code is given in Fig. 5 on the following page.

3.6 The S-Box

For the general-purpose versions of QARMAv2, we choose the following S-Box

$$
\mathcal{S} = \begin{bmatrix}
5 & E & 9 & 1 & 8 & A & C & B & 7 & 6 & D & 4 & 2 & 0 & F & 3
\end{bmatrix}
$$
instead of reusing one of QARMA’s S-Boxes. The reason for this is that if we use QARMA’s
\( \sigma_1 \) together with \( M_{4,1} \), then a nontrivial nonlinear invariant exists for the unkeyed round
function of the cipher [Bey23], which is invariant under translation by a set of \( 2^{32} \) constants.
The invariant does not seem to threaten the security of the design and is only likely to
improve some reduced round integral attacks. However, we see no reason to finalize a
design with such an invariant.

Using \( \sigma_0 \) would have prevented such invariants, but the S-Box has worse cryptographic
properties and gives Boomerang attacks one extra round of reach. An alternative could
have been to keep \( \sigma_1 \) and revert to Class II matrix \( M_{4,2} \). However, with \( M_{4,2} \) the number
of active cells in optimal related-tweak differential characteristics is significantly lower
than with \( M_{4,1} \), more so than in QARMA.

The S-Box \( \sigma_2 \) would have worked as well, but it is larger and slower.
The S-Box \( \wp \) has the same optimal cryptographic properties as QARMA’s \( \sigma_1 \), namely:

(i) The maximal probability of a differential is \( 1/4 \) and there are 15 such differentials;
(ii) The maximal absolute bias of a linear approximation is \( 1/4 \) and there are 30 such
linear approximations;
(iii) Each of the 15 non-zero component functions has algebraic degree 3;
(iv) Each input bit of the S-Box influences each output bit non-linearly.

The search for \( \wp \) imposed additional constraints on the S-Box, namely:

(v) For each output bit of the S-Box and of its inverse either the SOP (sum-of-products)
or the NOT-SOP of its logic negation (corresponding to a product-of-sums) are sums
of at most 3 monomials of degree at most 3; and

(vi) The sum of the degrees of these monomials is bounded by 8.

An extended search on randomly constructed S-Boxes yields only a few candidates, which
are all affine equivalent to each other. Their lightness is confirmed by performing synthesis
of the full cipher. Any strengthening of conditions (v) and (vi) yields no S-Boxes.

As a permutation, \( \wp \) is the product of four cycles of length 4. The rationale is that
we want to reduce the maximum number of non-linear invariants that the S-box- and
linear layer may have in common. Following [Bey18], the number of invariants of the
S-Box is related to the number of its cycles, whence it makes sense to minimize this
number. However, with the restriction of having fewer than 4 cycles – for instance with
cycle structures \( (5,5,6), (8,8) \) and \( (16) \) – we found no S-Boxes that satisfied our strict
constraints on the SOP and NOT-SOP. With a \( (4,4,4,4) \) cycle structure, however, good S-
Boxes were found. Finally, with a computer search, we verified that \( \wp \) (as well as \( \sigma_0 \), except
for its two fixed points) has no non-linear invariants, no closed-loop invariants over two
rounds, and no closed-loop quadratic invariants up to four rounds [WYWP18, WRP20].

Interestingly, \( \wp \) leads to slightly lighter and faster implementations than \( \sigma_1 \).
3.7 Alternative Constructions

We experimented also with a symmetric tweak schedule, where the backward rounds reverse the tweak schedule, as in QARMA. It used two different “optimal” tweak update functions for \( T_0 \) and \( T_1 \) – but sufficiently different from each other to reduce the likelihood of repeated mutual cancellations. For the two-layer version of the cipher, we looked at variants where only one row is swapped instead of two, and/or the swap is performed every round or every three rounds. In all these constructions, the number of active cells turned out to be significantly lower for all the parameter choices we tested.

4 Cryptanalysis

In this section we will perform a security analysis of the QARMAv2 design. This will lead to the choices for the parameter \( r \) (and thus for the number \( 2r + 2 \) of rounds) for each targeted security level, as summarized in Table 1.

<table>
<thead>
<tr>
<th>Variant</th>
<th>Block Size</th>
<th>Key Size</th>
<th>Time</th>
<th>Data</th>
<th>Parameter</th>
<th>Rounds</th>
</tr>
</thead>
<tbody>
<tr>
<td>QARMAv2-64-128</td>
<td>64 bits</td>
<td>128 bits</td>
<td>( 2^{128} )</td>
<td>( 2^{56} )</td>
<td>( r = 9 )</td>
<td>20</td>
</tr>
<tr>
<td>QARMAv2-128-128</td>
<td>128 bits</td>
<td>128 bits</td>
<td>( 2^{128} )</td>
<td>( 2^{80} )</td>
<td>( r = 11 )</td>
<td>24</td>
</tr>
<tr>
<td>QARMAv2-128-192</td>
<td>128 bits</td>
<td>192 bits</td>
<td>( 2^{192} )</td>
<td>( 2^{80} )</td>
<td>( r = 13 )</td>
<td>28</td>
</tr>
<tr>
<td>QARMAv2-128-256</td>
<td>128 bits</td>
<td>256 bits</td>
<td>( 2^{256} )</td>
<td>( 2^{80} )</td>
<td>( r = 15 )</td>
<td>32</td>
</tr>
</tbody>
</table>

4.1 Differential Cryptanalysis and Boomerang Attacks

4.1.1 Basic Properties and Cell-Level Bounds

The maximum differential probability (MDP) of QARMAv2’s 4-bit S-box is \( 2^{-2} \). This maximal probability is reached by 15 differential transitions. The complete difference distribution table (DDT) is illustrated in Fig. 6a. The boomerang connectivity table (BCT) [CHP+18] in Fig. 6b has a maximum probability of 10/16. The MixColumns matrices are in Class I and have a differential branch number of 4. Their cell-wise differential behavior is summarized in Fig. 8c.

![Differential properties of S-box and MixColumns.](image)

We bound the maximum probability of differential characteristics for QARMAv2 with the help of a MILP model of the cell-wise differential behavior. For a detailed description of the modeling approach, we refer to Appendix A.1 on page 34. The obtained bounds for all variants are summarized in Table 2. We list both results for the simple iterated round function with an additional initial half round (“half-cipher”, where \( r \) corresponds to \( r + 1 \).
S-box layers) and for the full reflective construction (“full-cipher”, where \( r \) corresponds to \( 2r + 2 \) S-box layers). The differential bounds are in a related-tweak setting, while the linear bounds are identical to single-key, single-tweak differential bounds.

An important remark is that, if we compare, say, a \( 2r \frac{1}{2} \) rounds half-cipher to a \( 2r \frac{1}{2} \) rounds full-cipher for “good” shuffles they seem to have a similar number of active cells (see also Table 8 on page 37) for sufficiently large \( r \). This is a bit surprising because it means that, while the reflector construction does offer advantages in other areas, it does not make the cipher intrinsically more secure - or less. Still, the reflector construction proves itself useful because for smaller \( r \) the weights seem to ramp up much faster than in a normal iterative construction, in which case it provides a clear advantage, for instance for round-reduced versions for special applications which require security compromises.

Table 2: Minimum number of active S-boxes for related-tweak differential characteristics and linear characteristics for QARMAv2. A bound of \( s \) active S-boxes implies a maximum probability (or squared correlation) of \( 2^{-2s} \) for differential (or linear) characteristics.

<table>
<thead>
<tr>
<th>( \ell )</th>
<th>Rounds</th>
<th>Half-cipher</th>
<th>Full-cipher</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>2 4 8 12 16 22 24 27 32 36</td>
<td>2 3 4 5 6</td>
</tr>
<tr>
<td>Diff.</td>
<td></td>
<td>2 6 11 17 26 34 44 50 55 59</td>
<td>5</td>
</tr>
<tr>
<td>Linear</td>
<td></td>
<td>16 23 30 35 38 41 50 57 62 67</td>
<td>5</td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>16 25 36 48 58 68 72 80 88 100</td>
<td>24 44 56 80 96</td>
</tr>
</tbody>
</table>

4.1.2 Key Recovery

To estimate the amount of rounds an attacker can append to a distinguisher, we analyse the differential distinguisher for QARMAv2-64 and \( r = 4 \) in Fig. 9 on page 39. While this distinguisher has 26 active cells, which is 2 more than the optimum of 24, it has no active cells in the state which allow us to append more rounds. If we assume that for this truncated differential pattern an optimal characteristic with \( p = 2^{-52} \) exists, we are able to sketch a key-recovery attack for \( r = 6 \) by appending 2 rounds each at the beginning and end of the distinguisher.

For this attack, we generate two structures of size \( 2^{53} \) with \( 2^{52} \) pairs each. Then for each structure, we sort the ciphertext based on the 12 inactive bits into buckets. For each pair in each bucket we generate the set of keys compatible with the required difference at the end of the distinguisher. As guessing all involved key-bits would require more than \( 2^{128} \) operations, we instead iterate over the \( \leq 2^{15} \) differences after one round was appended and then generate the keys that are compatible with these differences. The number of differences is at most \( 2^{15} \) because there are five active cells and after the S-box each active cell can have at most \( 8 = 2^3 \) differences. After we fix this difference one round past the distinguisher, we expect on average one candidate for the 13 active cells in \( o(K_1) \) and on average \( 2^{5.7} \) candidates for the five active cells in \( \text{MixColumns}(o(K_0)) \). The expected number of candidates for \( o(K_0) \) is higher because we fixed the difference to a known good value. Then, for the rounds added before the distinguisher, we need to guess 35 extra bits of \( K_0 \) and three extra bits of \( \text{MixColumns}(K_1) \). By independently verifying the guess in each column, we can keep the complexity below \( 2^{128} \). Finally, after the guesses have been verified, we can increase a counter for each surviving key candidate and then pick the candidate with the highest counter. We estimate that this attack needs \( 2^{24} \) encryption queries and \( 2^{124} \) time. While the time complexity of this sketched attack could be slightly
decreased and also depends on the specifics differences in the distinguisher, we believe that is not possible to add more than two rounds on each end to a differential distinguisher.

For the two-layer version, adding three rounds on each end might be possible because full diffusion is only reached after four rounds.

In conclusion, we expect that QARMAv2-64 with $r = 7$ resists differential attacks when data is limited to $2^{56}$ blocks. For 128-bit blocks with $\ell = 2$, we expect that no differential distinguisher exists for $r = 7$ when data is limited to $2^{80}$ blocks, and thus that the block cipher with $r = 11$ is secure against differential attacks with data $\leq 2^{80}$ and time $\leq 2^{256}$.

4.1.3 Boomerang Attacks

We bound the probability of characteristics for sandwich distinguishers with the help of a MILP model. We consider sandwich distinguishers where the cipher is decomposed into three parts $E_0 \circ E_m \circ E_1$, as illustrated in Fig. 7. Since $E_m$ covers many rounds, we assume that the center construction $C$ is part of $E_m$.

For the MILP model, we roughly follow the approach of Hadipour et al. [HNE22]: The outer parts $E_0, E_1$ are modeled with the same cell-wise probabilistic differential model as above, while the inner part $E_m$ models deterministic differential propagation (i.e., truncated propagation with probability 1) in both directions to identify jointly active S-boxes. The overall probability $p^2 r q^2$ of the sandwich distinguisher can then be derived based on the differential probability of the S-boxes in the outer rounds (bounded by the MDP of $4/16$) for $p, q$ and the boomerang probability of the jointly active S-boxes in the inner rounds (bounded by the maximum nontrivial entries in the BCT of $10/16$) for $r$.

For simplicity, instead of weights $2 \cdot 2 = 4$ in the outer part and $10/16$ in the inner part, we scale the weights to 6 in the outer part and 1 in the inner part, so the final bounds in Table 3 on the next page need to be scaled by a factor of about $2^3$ to get the boomerang probability. Because we are analyzing a reflector-based cipher, we not only have to evaluate different combinations for the round number of $E_0, E_m, E_1$, but also different configurations of where exactly the central construction is located. We also optimize the position of the split between $E_0, E_m, E_1$ to be in the middle of a round, since this yields higher probabilities. Our results are summarized in Table 3. We provide selected characteristics in Figs. 11 to 13 on pages 41–43 in the Appendix. Each table entry is the minimum over all possible boomerang configurations for the given round configuration. We observe that the best results for a given total number of rounds are typically for relatively unbalanced round configurations, where the forward part is either much longer or shorter than the backward part, and are thus likely not useful for attacks on the full construction. Again, these bounds rely on cell-wise differential bounds for individual characteristics for the outer boomerang rounds; exploiting a potential clustering for specific bitwise differences effect may slightly increase the resulting probabilities. On the other hand, the bound for the inner part $E_m$ is likely too optimistic, i.e., the actual probability of any conforming characteristics is likely lower than the bound. If we do not consider a separate switching layer and instead derive the bounds directly as $p^2 q^2$ using...
Table 3: Bounds for sandwich characteristics for QARMAv2 with $r_F$ forward and $r_B$ backward rounds, where an entry $x$ corresponds to a boomerang probability of about $2^{-(2/3)x}$

<table>
<thead>
<tr>
<th>$r_F \setminus r_B$</th>
<th>$0 \frac{1}{2}$</th>
<th>$1 \frac{1}{2}$</th>
<th>$2 \frac{1}{2}$</th>
<th>$3 \frac{1}{2}$</th>
<th>$4 \frac{1}{2}$</th>
<th>$5 \frac{1}{2}$</th>
<th>$6 \frac{1}{2}$</th>
<th>$7 \frac{1}{2}$</th>
<th>$8 \frac{1}{2}$</th>
<th>$9 \frac{1}{2}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$0 \frac{1}{2}$</td>
<td>6</td>
<td>6</td>
<td>6</td>
<td>6</td>
<td>6</td>
<td>15</td>
<td>21</td>
<td>32</td>
<td>44</td>
<td></td>
</tr>
<tr>
<td>$1 \frac{1}{2}$</td>
<td>6</td>
<td>6</td>
<td>6</td>
<td>6</td>
<td>7</td>
<td>15</td>
<td>21</td>
<td>32</td>
<td>44</td>
<td>60</td>
</tr>
<tr>
<td>$2 \frac{1}{2}$</td>
<td>6</td>
<td>6</td>
<td>6</td>
<td>8</td>
<td>18</td>
<td>24</td>
<td>35</td>
<td>47</td>
<td>62</td>
<td>76</td>
</tr>
<tr>
<td>$3 \frac{1}{2}$</td>
<td>6</td>
<td>9</td>
<td>15</td>
<td>28</td>
<td>30</td>
<td>38</td>
<td>48</td>
<td>63</td>
<td>76</td>
<td></td>
</tr>
<tr>
<td>$4 \frac{1}{2}$</td>
<td>10</td>
<td>12</td>
<td>18</td>
<td>30</td>
<td>46</td>
<td>58</td>
<td>73</td>
<td>76</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$5 \frac{1}{2}$</td>
<td>12</td>
<td>15</td>
<td>26</td>
<td>38</td>
<td>51</td>
<td>82</td>
<td>85</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$6 \frac{1}{2}$</td>
<td>15</td>
<td>21</td>
<td>35</td>
<td>48</td>
<td>73</td>
<td>97</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$7 \frac{1}{2}$</td>
<td>21</td>
<td>32</td>
<td>49</td>
<td>63</td>
<td>81</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$8 \frac{1}{2}$</td>
<td>32</td>
<td>46</td>
<td>62</td>
<td>75</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$9 \frac{1}{2}$</td>
<td>46</td>
<td>60</td>
<td>78</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

the differential bounds, we see that boomerang attacks are expected to perform worse than differential attacks due to the roughly linear growth of differential bounds.

4.2 Linear Cryptanalysis

4.2.1 Basic Properties and Cell-Level Bounds

The maximum squared correlation of QARMAv2’s 4-bit S-box is $2^{-2}$, which is reached by 30 linear approximations. The MixColumns matrices are involutive ($M = M^{-1}$) and symmetric ($M = M^\top$). This implies that the linear behaviour of MixColumns on linear masks is the same as its differential behaviour on differences. In particular, the linear branch number is 4 and the cell-wise linear behaviour is the same as in Fig. 6c. As a consequence, we can reuse the same cell-wise model as for the differential case to model the linear propagation in the state. Since the tweak schedule is linear, we do not need to take it into account. Thus, the linear model is equivalent to the differential single-key, single-tweak model. The resulting bounds are summarized in Table 2. For $\ell = 1$, at least 32 active S-boxes, corresponding to a squared correlation of at most $2^{-64}$, are reached after $6 \frac{1}{2}$ rounds of the half-cipher (parameter $r = 6$) or $6 \frac{2}{3}$ rounds of the full-cipher ($r = 3$). For $\ell = 2$, at least 64 active S-boxes are reached after $8 \frac{1}{2}$ rounds of the half-cipher ($r = 8$) or $10 \frac{2}{3}$ rounds of the full-cipher ($r = 5$).

4.2.2 Key Recovery

Efficient key recovery for linear distinguishers can take advantage of the FFT technique [CSQ07]. This technique was recently improved further to define an optimized framework [BCFG21, FGNP20]. Taking the number of rounds for full diffusion (3 rounds for $\ell = 1$, 4 rounds for $\ell = 2$) as a bound for key-recovery rounds in the beginning and end of the cipher, we expect a solid security margin for linear cryptanalysis.

4.3 Impossible-Differential and Zero-Correlation Cryptanalysis

The resistance of QARMAv2 against impossible-differential and zero-correlation cryptanalysis is evaluated using an automated model inspired by the framework in [HSE23].

We split our cipher in two halves: the upper and the lower half. Then, we model propagation with probability 1 in forward/backward direction for the upper/lower half, respectively. As we want to find contradictions, where the two halves meet, we model each cell of the block cipher to be in one of four states: inactive, active with a single
difference/mask, active with nonzero difference/mask, and active with unknown (possibly) zero difference. This representation allows us to find contradictions where a cell that is inactive in one half is active with a single or any nonzero difference in the other half.

When running this model for $\ell = 1$, we find that the longest impossible differential distinguisher spans 9 rounds as depicted in Fig. 10. Concretely, it spans 1 half round, 4 full forward rounds, the centre construction and 3 full backwards rounds and one backward half round. This implies that for $r = 4$ no impossible differential distinguisher exists.

When considering $\ell = 2$, 4 rounds provide full diffusion; so with $r = 5$, no impossible differential distinguisher exists. This is because we can skip the initial half round and one of the full rounds by leaving the plaintext and one of the tweaks inactive. Due to the full diffusion, when the two halves meet, no miss-in-the-middle is possible.

When analysing the zero-correlation setting with $\ell = 1$ a fixed tweak, the longest distinguisher we can find covers 6 rounds, i.e. QARMAv2 with $r = 2$.

In the case of a variable tweak with $\ell = 1$, we need to consider that a contradiction might also arise in the tweak schedule. However, for $r = 4$, no zero-correlation distinguisher exists. This is because after 3 applications of the state shuffle and MixColumns, all linear masks must be active with unknown masks. Then, with another round this implies the same unknown masks in the tweak. As our cipher is symmetric this holds for $T_0$ and $T_0$. For $\ell = 2$, the same argument applies for $r = 5$ due to the slower diffusion.

Considering the number of rounds for full diffusion ($3$ rounds for $\ell = 1$, $4$ rounds for $\ell = 2$) as a bound for key-recovery rounds in the beginning and end of the cipher, we expect a solid security margin for impossible-differential and zero-correlation cryptanalysis.

### 4.4 Integral Cryptanalysis and the Division Property

Integral cryptanalysis operates on the principle of identifying distinct properties exhibited by a collection of ciphertexts, which correspond to a predetermined set of plaintexts with a specific structure. From this point of view, it is a generalization of differential cryptanalysis, where the collection of plaintexts has cardinality two.

The attacker typically assembles a collection of $N$ chosen plaintexts and generates the corresponding ciphertexts using the target block cipher. By observing the $\text{eXchangeRows}$ of the ciphertexts in specific word positions, if the result is $0$, it indicates the presence of an integral characteristic within the block cipher when subjected to the $N$ chosen plaintexts.

Essentially, integral cryptanalysis exploits low algebraic degrees of a cryptographic primitive, and countering it starts with the proper choice of S-Box, for instance by making sure that all its outputs have sufficiently large degree (see Section 3.6 on page 13).

The Division property is a generalization of the integral property that was proposed by Todo at Eurocrypt 2015 [Todo15]. This tool allows for a more systematic way of finding the integral properties by considering their propagation through the various operations of a cipher. This way, the attacker can ascertain the presence of an integral characteristic.

While the original Division property was word-based, Todo and Morii [TM16] introduced the bit-based division property, which enables a more precise analysis.

**Definition 1.** (Bit-based Division Property [TM16]) A multi-set $X \subseteq \mathbb{F}_2^n$ is said to have the division property $\mathbb{K}$ for some set of $n$-dimensional vectors $\mathbb{K}$ if for all $u \in \mathbb{F}_2^n$, it fulfills

$$\bigoplus_{x \in X} x^u = \begin{cases} \text{unknown, if there is } k \in \mathbb{K} \text{ s.t. } u \geq k \\ 0, \text{ otherwise} \end{cases}$$

**Definition 2.** (Balanced Position) Let $Y \subseteq \mathbb{F}_2^n$ be a multi-set of vectors. A coordinate position $0 \leq i < n$ is called balanced position if $\bigoplus_{y \in Y} y_i = 0$.

The bit-based division property is an effective technique for determining whether a particular monomial is present (the unknown case) or absent (the zero case) in the polynomial representation of the product of output bits. Once we have successfully identified...
a set of balanced positions that correspond to a specific input division property \( k \), we can use this property to distinguish a cipher \( E \) from a randomly chosen permutation. We first construct a set \( X \) of plaintexts that form an affine subspace, aligning with the input division property \( k \). For each vector \( x = (x_0, x_1, \cdots, x_{n-1}) \) within the set \( X \), we set the \( i \)-th coordinate to a fixed constant \( c_i \) from the binary set \( \{0, 1\} \) if the \( i \)-th coordinate of \( k \) is 0. If the \( i \)-th coordinate of \( k \) is 1, we allow \( x_i \) to take on any value within the binary set \( \{0, 1\} \). The size of \( X \) is \( 2^{wt(k)} \), where \( wt(k) \) is the Hamming weight of \( k \).

While originally a direct programming approach was used to find bit-based division properties for 32-bit block ciphers [Tod17], this is impractical for larger block sizes. Instead, the propagation rules can be encoded for solving with automatic tools, as proposed by Xiang et al. using MILP [XZBL16], and Sun et al. using SAT/SMT [SWW17].

We apply the SAT/SMT approach to \( \text{QARMAv2-64} \) and convert the propagation rules into Conjunctive Normal Form (CNF) to be solved using CryptoMiniSAT [SNC09]. The longest integral characteristic found with the bit-based division property covers 5 forward rounds of the cipher. The best division properties are obtained from an input with 11 active S-boxes (44 active bits) and at the output we get 32 balanced bits. There are total 48 such characteristics, one of which is given below (with “B” denoting balanced and “U” denoting unknown bits):

\[
\begin{pmatrix}
0000 & 0000 & 1111 & 1111 \\
0000 & 1111 & 1111 & 1111 \\
1111 & 1111 & 1111 & 0000 \\
1111 & 1111 & 0000 & 1111
\end{pmatrix}
\rightarrow
\begin{pmatrix}
BBBB & UUUU & UUUU & BBBB \\
UUUU & UUUU & UUUU & BBBB \\
BBBB & UUUU & BBBB & BBBB \\
UUUU & UUUU & BBBB & BBBB
\end{pmatrix}
\]

We also identified another integral characteristic for 5 rounds with 59 consecutive active bits (that can start from any bit positions). From each such input division property, we get all 64 bits of the output state as balanced bits. For 6 forward rounds of \( \text{QARMAv2-64} \), we did not find any integral distinguishers.

If we use the S-Box \( \sigma_0 \) we find integral characteristic for up to 6 rounds and none on 7 rounds. With \( \sigma_1 \) we stop at 5 rounds like with \( p \).

### 4.5 Slide, Meet-in-the-Middle and other Structural Attacks

Slide attacks [BW99] and variants [BW99, Kar07] exploit similarity between sequences of rounds. The main idea behind these attacks is that once sequences of rounds induce the same permutation, then one can attack the rounds around the similar permutation by using slid pairs, i.e., pairs of values that have the same input to the same permutation.

For \( \text{QARMAv2} \) the issue is mitigated by several components. The cipher uses four different types of rounds plus the central construction. The round constants are different at each round, and since they are not light nor dense, one cannot find complementation properties or slide properties that hold with significant probability. Finally, the central function breaks the cipher into two parts. Even if one succeeds to find a combination of (key, tweak) and round constants such that some rounds are the same (in the related-key settings), then the cipher’s design implies that this similarity should hold through the central construction, which is independent of the tweak, and uses a second key, and through the inverse rounds, again with a different key and the round constants in reverse order.

The best MitM attacks on \( \text{QARMA} \) [ZD16] can be adapted to \( \text{QARMAv2} \) and we do not expect them to be more successful. Furthermore, the authors of [ZD16] do not include the time to prepare the data tables in their time complexity – giving the latter only as “encryption units.” Therefore their 10-round key recovery attack on \( \text{QARMA-64} \) has data complexity of \( 2^{53} \) chosen plaintexts and time complexity of \( 2^{116} \), and on \( \text{QARMA-128} \) (also 10 rounds) the complexity is \( 2^{105} \) data and time \( 2^{232} \). Since there is a large gap between the number of attacked rounds and the parameters given in Table 1 on page 15, we posit that similar attacks are not a threat to the security of \( \text{QARMAv2} \).
4.6 Invariant Subspace Cryptanalysis and Non-Linear Invariants

In order to determine whether (untweaked) QARMAv2-64 or QARMAv2-128 have invariant subspaces we can repeat the arguments in [Ava17], which are similar to the approach in [LMR15], i.e. we construct a vector space that includes the differences between the round constants and their transforms through round function. We added support for QARMAv2 to the software and we could not find any small dimensional invariant subspace of the un-tweaked ciphers: for forward function of QARMAv2-64 does not have non-trivial invariant subspaces as soon as we consider three full rounds, and QARMAv2-128 as soon as we consider four full rounds. We note that the software assumes a normal iterative cipher, however if a subspace is invariant for a round function, it is also invariant for the inverse function, so in fact we only omit the central construction $C$. Including the latter would not reduce the dimension of the invariant subspaces. The only problem with this approach is that it does not handle the tweak.

A different approach consists in looking for symmetric non-linear invariants, i.e. invariants of the form $\sum_i g_1(c_i) = \sum_i g_2((S \circ M \circ \tau)(\mathcal{F})|_{i})$ where the sum is taken over one layer or the whole state. The non-existence of such invariants has been ruled out in Section 3.6 on page 13 for up to two rounds in general and up to four rounds for quadratic invariants.

However, we can go further. Since non-linear invariants are support functions of sets of points, we can address them by considering the trails of their enveloping subspaces.

4.6.1 Subspace Coset Trails

Subspace trail cryptanalysis keeps track of subspaces of the state space – in our case $V = (F_2^d)^d = \prod_{i=0}^{d-1} V_i$ where $V_i = F_2^n$ and $d = 16\ell$ – that remain invariant, resp. are translated, under the action of the unkeyed round function. We slightly generalise the definition from [GRR16].

**Definition 3.** Let $F : V \to V$ be a round function, and $r$ a natural number. A sequence of $r+1$ linear subspaces $Z^{(0)}, Z^{(1)}, \ldots, Z^{(r)} \subseteq V$ is called a $r$-round subspace coset trail if for each $i = 1, 2, \ldots, r$ there is at least one pair of elements $(a_i, b_i) \in V^2$ such that $F(Z^{(i-1)} + a_i) \subseteq Z^{(i)} + b_i$. The points $a_i, b_i$ are called input and output offsets.

We always need to consider each subspace together with an offset, because the images of two different cosets of the same subspace under the round function are not necessarily translates of one another, or both contained in a proper (coset of a) subspace of $V$. We call such a (subspace, offset) pair a (subspace) coset for brevity.

If a round tweakey (i.e. the sum of round key, round tweak, and round constant) is contained $Z^{(i)} + b_i + a_{i+1}$, then it can be added to the state and the property for a given state value to belong to one of the subspace cosets $Z^{(i)} + b_i$ can propagate one more round. By observing the input and output values, it can thus be determined whether the round tweakeys belong to a smaller set of weak tweakeys – the sets $Z^{(i)} + b_i + a_{i+1}$. Therefore we want to study how subspaces propagate through a block cipher. There are obvious trivial trails, for instance where all $Z^{(i)} = \{0\}$ and those where all $Z^{(i)} = V$. Such trails are not useful, so we shall assume that trails do not start with the zero space and do not end with $V$. Our goal is to prove that such relations cannot extend over more than a few rounds.

We shall assume that $Z = \bigotimes_{i=0}^{d-1} Z_i$, where the $Z_i \subseteq V_i$ are proper subsets for $i \in \mathcal{I}$.

First of all, if any of the $Z_i = F_2^n$, after three resp. four rounds the subspace $Z^{(i)}$ is equal to $V$. So we can assume that all $Z_i \neq F_2^n$.

The cosets of dimensions 0, resp. 1 are just individual values, resp. pairs of values, and tracking trail that contains only these types of cosets in the cells is done in more generality (i.e. with likelihoods not necessarily equal to 1) by linear and differential cryptanalysis. Thus, we restrict ourselves to trails that contain at least one cell with a dimension 2 or 3 coset. With a simple sage script we listed the images of the cosets in $F_2^4$ under $\mathcal{P}$. No coset...
of dimension 3 maps to a coset of dimension 3, and only the following cosets of dimension 2 map to other cosets of dimension 2:

\[
\begin{align*}
0 + (2, 8) & \xleftarrow{p} 4 + (1, 8) \\
4 + (3, C) & \xleftarrow{p} 4 + (3, C) \\
3 + (5, 8) & \xleftarrow{p} 1 + (6, 8) \\
2 + (5, B) & \xleftarrow{p} 2 + (4, A) \\
1 + (7, 9) & \xleftarrow{p} 3 + (7, A)
\end{align*}
\]

(11)

The fact that a coset \( Z \) is mapped by the round function to a coset \( Z' \), i.e. \( Z' = (S \circ M \circ \tau)(Z) \), can be expressed via a set of relations such as

\[
Z'_0 = p(\rho(Z_{10}) + \rho^2(Z_8) + \rho^3(Z_{15})).
\]

(12)

Let us have a closer look at this relation. As we are chaining states that contain dimension 2 cosets, \( Z'_0 \) and at least one of the \( Z_i \) on the r.h.s. have cardinality four. If, say, \( \#Z_{10} = 4 \), we have \( Z'_0 = p(\rho(Z_{10}) + \delta) \) where \( \delta \) is the sum of an element of \( \rho^2(Z_8) \) and one of \( \rho^3(Z_{15}) \). In general, a cell \( Z_i \) contributes to \( Z' \) only after a rotation by one to three bits. The admissible mappings of underlying vector subspaces corresponding to (11) are

\[
\begin{align*}
\langle 2, 8 \rangle & \xleftarrow{p} \langle 7, B \rangle, \ (1, 2) \cdot \ (2, 4) \cdot \ (4, 8) \\
\langle 3, C \rangle & \xleftarrow{p} \langle 3, C \rangle, \ (6, 9) \\
\langle 5, 8 \rangle & \xleftarrow{p} \langle 6, B \rangle, \ (1, C) \cdot \ (2, 9) \cdot \ (3, 4) \\
\langle 5, B \rangle & \xleftarrow{p} \langle 3, A \rangle, \ (5, 8) \cdot \ (1, A) \cdot \ (2, 5) \\
\langle 7, 9 \rangle & \xleftarrow{p} \langle 7, A \rangle, \ (5, B)
\end{align*}
\]

Note that in the first, third and fourth relation the first subspace on the r.h.s. cannot occur as one of the \( Z_i \)'s, since they are all rotated by non-trivial amounts before being added together. In other two relations the first subspace can occur since it is invariant upon rotation by two bits. These relations form a tree with longest path

\[
\rho^i ((6, 8)) \mapsto \langle 5, 8 \rangle \mapsto \langle 5, B \rangle \mapsto \langle 7, 9 \rangle
\]

(13)

(with either \( i = 1, 2, \) or 3), except for the (single) cycle corresponding to the cosets \( 4 + \langle 3, C \rangle \xleftarrow{p} 4 + \langle 3, C \rangle \). We want to prove that the latter cannot occur. The coset \( 4 + \langle 3, C \rangle \) is not invariant under rotation by one bit. This means that in order to have it as a summand in the r.h.s of a relation like (12), also its rotation by one bit (or, equivalently three bits) \( 2 + \langle 6, 9 \rangle \) will contribute to the values of at least one more cell in the same output column. However

\[
p(2 + \langle 6, 9 \rangle) = p(\{2, 4, B, D\}) = \{0, 4, 8, 9\}
\]

is not a dimension 2 coset, and in fact it is contained in a coset of dimension at least 3. Since \( p \) maps no coset of dimension 3 onto another, after one more round entire cells will be saturated and after three or four more rounds the subspace in the trail will cover the whole state (by the full-diffusion property). Therefore (13) places an upper bound on the length of any constant subspace trail: in at most three (or four, when \( \ell = 2 \)) further rounds, the subspace will be the whole \( V \). The number of rounds in a useful subspace coset trail is therefore bounded by 6 for \( \ell = 1 \) and by 7 for \( \ell = 2 \).
4.7 Algebraic Attacks

We consider the applicability of algebraic attacks [CP02] on QARMA\textsuperscript{2-b}.

As in [Ava17, Section 4.2], QARMA\textsuperscript{2-b} has sufficiently many rounds to reach maximum algebraic degree \( b - 1 \). In fact, the exact same argument holds, and QARMA\textsuperscript{2-64}, resp. QARMA\textsuperscript{2-128} reach the upper bound \( b - 1 \) after 7, resp. 8 rounds. QARMA\textsuperscript{2-64} and QARMA\textsuperscript{2-128} have at least 16 and 20 rounds in their most aggressive general purpose versions, hence they should have sufficiently many rounds to reach maximum degree.

Let us now count how many quadratic equations and variables are necessary to describe the two variants of QARMA\textsuperscript{2}. Following [CP02], it is straightforward to verify that both \( \sigma_1 \) and \( \sigma \) are described by \( e = 21 \) quadratic equations in the 8 input and output variables over \( \mathbb{F}_2 \). Hence, the entire system for a fixed-key QARMA\textsuperscript{2-b} permutation consists of \( b^4 \cdot (2r+2) \cdot e \) quadratic equations in \( b^4 \cdot (2r+2) \cdot v \) variables. For QARMA\textsuperscript{2-64} with \( r = 9 \) this translates to 6720 equations in 2560 variables, and for QARMA\textsuperscript{2-128} with \( r = 11 \) to 16128 equations in 6144 variables. For comparison, a fixed-key AES-128 (resp. AES-256) permutation consists of 6400 (resp. 8960) equations in 2560 (resp. 3584) variables.

The same conclusions as in [Ava17] should stand, i.e. that QARMA\textsuperscript{2-64} and QARMA\textsuperscript{2-128} should offer good resistance against algebraic cryptanalysis if also the AES does.

4.8 Security Implications of the Central Construction

The arguments in [Ava17] hold in a stronger form. For each addition of \( K_0 \) (resp. \( K_1 \)) on the forward path, \( o(K_0) \) (resp. \( o^{-1}(K_1) \)) is added in the backward path. Hence, any reflection-like relation encompassing the central construction \( C \) and any (odd) amount of rounds on each of the two sides would be disrupted (i.e. its likelihood reduced) in a similar way as described in [Ava17]. This means that for each \( i \in \{0, 1\} \), the map \( K_i \rightarrow K_i + o^{-1}(K_i) \) is bijective (we can ignore the constant \( \alpha \)), and thus it does not give rise to potentially equivalent or weak classes of keys. The fact that we add \( o^2(K_0) \) and \( o^{-2}(K_1) \) at the center allows to apply the same argument with an odd number of rounds at a single side of the central construction.

In practice, we have observed that key recovery is more difficult with the application of orthomorphisms to the keys than without.

4.9 Security Parameters and Security Claims

Based on our cryptanalysis we propose one variant with 64 bit blocks, QARMA\textsuperscript{2-64-128}, and three variants with 128 bit blocks and varying key sizes, QARMA\textsuperscript{2-128-128}, QARMA\textsuperscript{2-128-192}, QARMA\textsuperscript{2-128-256}. We impose a data limit of \( 2^{56} \) blocks per key for the small block size and \( 2^{80} \) blocks per key for the large block size. This is consistent with existing calls for algorithms like the NIST call for lightweight cryptographic algorithms. Furthermore, we believe that these limits are high enough to not affect any real world application. Our security claims and required number of rounds are summarized in Table 1.

Note that the time complexities must include also offline precomputations, for the simple rationale that they are necessary in any case to mount a first attack, and if that were not possible, this would make any further attack also not possible.

4.10 Comparison to QARMA

QARMA\textsuperscript{2}'s parameter \( r \) seems, superficially, larger than QARMA's at the same security level. When we will later compare the latencies directly, indeed we lose a bit to QARMA, but we posit that this seems worthy since we get in exchange a longer tweak sizes. Furthermore, we have more robust security margins in the sense that the 128, 192, and 256 bit levels are
defined as time $\Omega(2^{128})$, $\Omega(2^{192})$, and $\Omega(2^{256})$, respectively, to mount a successful attack on the cipher. For QARMA this was the product of data and time.

With the data limits from Table 1 on page 15, the 128-bit security level for QARMA-64 represented a time $\Omega(2^{72})$, and the two security levels of 192 and 256 bits for QARMA-128 translate to time $\Omega(2^{112})$ and $\Omega(2^{176})$. This is due to the fact that QARMA is formally a three-round EM construction [EM91] and in fact barely more than a single round design because of the simplicity of the second round that comprises only the central construction. On the other hand, QARMAv2 is a key alternating design. This said, the implementor may reduce the parameter $r$ by two for QARMAv2 when used with a single tweak, i.e. when the second tweak block is derived from the first.

5 Hardware Implementation and Evaluation

We follow the evaluation framework outlined in [BIL+21]. Our target construction is a low-latency block cipher, hence we evaluate the metrics of a fully unrolled circuit from the input to output ports. To perform a fair evaluation we compare our design with the following ciphers: AES-{128|192|256}, QARMA(v1)-{64|128}, PRINCE, Orthros, SPEEDY-{6|7}, MIDORI-{64|128}, PRESENT-{80|128}, and SKINNY-{64|128}. We also compare our design with the ASCON-p12 permutation used in a single key Even-Mansour mode. We skip PRINCEv2 because its latency and area are almost identical to the original PRINCE.

In order to guarantee a fair comparison to SKINNY [BJK+16] we use the more recent parameters that are used in Romulus for the NIST Lightweight Cryptography Competition [IKMP19a, IKMP19b]. We observe that in related-tweakkey characteristics for SKINNY-128-384, a probability not exceeding $2^{-128}$ is reached first with 26 rounds. If we add then 10 rounds because of the 6-round full-diffusion we reach 36 rounds and this means that the 40 rounds used in Romulus represent a margin of 10%. Applying the same reasoning to SKINNY-64-192, we see that probability $2^{-64}$ is achieved after 18 rounds. Adding 10 rounds and a margin of 10% we reach 31 rounds, which we round up to 32, since this is the minimum suggested in the SKINNY paper anyway.

Now, the relative area and latency advantages between different primitives may vary considerably with the manufacturing processes. Hence, a single data point is not sufficient to justify a company or standardization body’s business decision of “betting” on a primitive over another. Hence, we compare our choice of ciphers at three different processes:

1. At 15nm lithography with the Nangate 15nm Open Cell Library [MMR+15];
2. At 90nm lithography with the STM cell library CORE90GPSVT.CMOS090LP; and
3. At the TSMC 5nm lithography with the tsmc_sch280pp57_cln05fb41001 library, courtesy of the Arm implementation team.

For a fair evaluation we adhered to the following design flow for all the ciphers:

1. The ciphers have been implemented in VHDL, and a functional simulation was done using the Modelsim software. Correctness has been verified against test vectors.
2. For the 90nm and 15nm processes we synthesize the circuits with Synopsys Design Compiler. For the 5nm process, the VHDL code is first converted to Verilog using GHDL with the yosys plugin (available at https://github.com/ghdl/ghdl-yosys-plugin), and then compiled using Cadence Genus.
3. Optimizing for area is a computationally heavy stage that outputs a circuit with, usually, near minimal area. We use this as the datapoint for the area-optimized circuit. This step also outputs the total critical path of the circuit.
4. Timing simulations have then been performed on the synthesized netlist. Correctness was again verified against test vectors.
5. For the 90nm and 15nm processes, the switching activity of each gate of the circuit
was collected during post-synthesis simulation. The average power was obtained using Synopsys Design Compiler, using the back annotated switching activity.

6. To generate datapoints for the latency-optimized circuits, we repeat the above processes, but constrain the total signal delay between the input/output ports to some impossibly low value, such as 1ps. The compiler of course fails, but it still outputs a circuit with a likely near minimal critical path.

A further note about the implementation methodology: With the exception of the AES, we described the S-Boxes as tables. Modern synthesis software can optimize overall implementations involving 4-bit S-Boxes often better or at most only slightly sub-optimally starting from a LUT than using ad-hoc implementations (either hand-optimized or generated with software like PEIGEN). Larger S-Boxes are a different matter, because of their higher intrinsic complexity. For the AES we used the Maximov-Ekdahl circuit [ME19]. For SKINNY-128 we tried both tables and the ad-hoc circuit from its designers, and the table-based description resulted in significantly better area and latency results.

All results are presented in Tables 4 to 6 on the following page. We include an energy-related (inverse) Figure of Merit (FoM), namely the product of Delay and Power, divided by block size, normalized as a percentage of the corresponding value for the AES-128. Note that with this definition, lower values are better. This is just one of the many possible Figures of Merit that can be associated to a circuit: For instance, in [BDN+10], the FoM is defined as throughput divided by the square of the area.

Even though our FoM necessarily leaves out some factors, such as overheads associated to other operations in modes of operation, it is still quite useful. For instance, since in XEX we need two invocations of a non-tweakable block cipher to encrypt a block, we can just double the latter’s FoM for comparison purposes.

We are not allowed to report power consumption at 5nm. Hence, we use the product of area and delay divided by the block size as a FoM. Even though area and power are correlated, power depends also on the type of gates used. A cipher with an 8-bit S-Box will be synthesized with a different mix of gates than one with a 4-bit S-Box, a fact that is ignored by using the area in place of the power.

Regarding the 90nm results, we note that the synthesis step traverses a wide design space before producing a circuit that optimizes an internal objective function. This may sometimes yield strange results, like the area of AES-256 being less than that of AES-192. This only means that the compile step selected circuits which it deemed best according to some internal objective function among the explored options.

Our results show that QARMAv2 offers strong advantages with respect to all other ciphers we tested in fully unrolled implementations. For instance, at the 64-bit block level, it is almost always faster than tweaked circuits designed around MIDORI-64, PRINCE, or PRESENT, because the latter, for instance if used in a XEX construction, would require two cascaded invocations of the primitive. At the 128-bit block level, the advantage is even more striking. At all security levels, QARMAv2-128 is both smaller and faster than the corresponding AES. In fact, at all security levels the Power-Delay-per-bit FoM advantage of QARMAv2 over the AES is always between 3 and 10, before even taking into account the extra operations needed to build modes of operation. MIDORI-128 is a bit smaller and faster but it is not tweakable. PRESENT-128 is on the whole comparable, sometimes faster, sometimes slower, sometimes larger, sometimes smaller, but it is also not tweakable. SPEEDY is faster, but its security margins need to be re-calculated, its block size is awkward for most applications, and its inverse is slower. SKINNY is designed to have a minimalistic round function to perform ideally in very small round based implementations. It is therefore expected that both performance and area will suffer significantly in a fully unrolled implementation, and our results clearly confirm that it is aimed at a different use cases and deployments. While we do not expect round-based implementations of QARMA and QARMAv2 to be overtly large or slow, they may not compare favourably to SKINNY.
### Table 4: Comparative Evaluation Metrics for the Nangate 15nm Process and Library

<table>
<thead>
<tr>
<th>Cipher</th>
<th>Rounds</th>
<th>Area µm²</th>
<th>GE</th>
<th>Delay ps</th>
<th>Power mW</th>
<th>FoM %</th>
<th>AES</th>
</tr>
</thead>
<tbody>
<tr>
<td>AES-128</td>
<td>10</td>
<td>22478</td>
<td>114392</td>
<td>2728</td>
<td>26.63</td>
<td>100</td>
<td></td>
</tr>
<tr>
<td>AES-192</td>
<td>12</td>
<td>26098</td>
<td>135796</td>
<td>2981</td>
<td>34.96</td>
<td>157</td>
<td></td>
</tr>
<tr>
<td>AES-256</td>
<td>14</td>
<td>37445</td>
<td>161549</td>
<td>3199</td>
<td>47.64</td>
<td>240</td>
<td></td>
</tr>
<tr>
<td>PRESENT-80</td>
<td>31</td>
<td>3808</td>
<td>19370</td>
<td>1001</td>
<td>2.87</td>
<td>11.2</td>
<td></td>
</tr>
<tr>
<td>PRESENT-128</td>
<td>31</td>
<td>3333</td>
<td>20105</td>
<td>960</td>
<td>2.67</td>
<td>12.5</td>
<td></td>
</tr>
<tr>
<td>MIDORI-64</td>
<td>16</td>
<td>1929</td>
<td>9814</td>
<td>629</td>
<td>0.96</td>
<td>2.61</td>
<td></td>
</tr>
<tr>
<td>MIDAIRON-128</td>
<td>20</td>
<td>10120</td>
<td>29502</td>
<td>854</td>
<td>3.35</td>
<td>6.20</td>
<td></td>
</tr>
<tr>
<td>PRINCE</td>
<td>12</td>
<td>1664</td>
<td>8465</td>
<td>356</td>
<td>0.67</td>
<td>1.58</td>
<td></td>
</tr>
<tr>
<td>Orthros</td>
<td>12</td>
<td>5766</td>
<td>29329</td>
<td>468</td>
<td>2.33</td>
<td>4.66</td>
<td></td>
</tr>
<tr>
<td>SPEEDY-6</td>
<td>6</td>
<td>5220</td>
<td>26551</td>
<td>545</td>
<td>2.06</td>
<td>1.63</td>
<td></td>
</tr>
<tr>
<td>SPEEDY-7</td>
<td>7</td>
<td>6160</td>
<td>36018</td>
<td>659</td>
<td>3.18</td>
<td>3.02</td>
<td></td>
</tr>
<tr>
<td>AES-128</td>
<td>12</td>
<td>8565</td>
<td>44519</td>
<td>436</td>
<td>2.07</td>
<td>1.63</td>
<td></td>
</tr>
<tr>
<td>LION-128</td>
<td>32</td>
<td>4069</td>
<td>20693</td>
<td>1953</td>
<td>5.03</td>
<td>42.6</td>
<td></td>
</tr>
<tr>
<td>SKINNY-64-192</td>
<td>32</td>
<td>15820</td>
<td>94200</td>
<td>3862</td>
<td>4.09</td>
<td>48.91</td>
<td></td>
</tr>
<tr>
<td>SKINNY-128-256</td>
<td>40</td>
<td>30480</td>
<td>10498</td>
<td>4156</td>
<td>3.82</td>
<td>52.87</td>
<td></td>
</tr>
</tbody>
</table>

### Table 5: Comparative Evaluation Metrics for the STM 90nm Process and Library

<table>
<thead>
<tr>
<th>Cipher</th>
<th>Rounds</th>
<th>Area µm²</th>
<th>GE</th>
<th>Delay ps</th>
<th>Power mW</th>
<th>FoM %</th>
<th>AES</th>
</tr>
</thead>
<tbody>
<tr>
<td>AES-128</td>
<td>10</td>
<td>30477</td>
<td>80199</td>
<td>4211</td>
<td>87.12</td>
<td>100</td>
<td></td>
</tr>
<tr>
<td>AES-192</td>
<td>12</td>
<td>46700</td>
<td>100609</td>
<td>49.54</td>
<td>112.22</td>
<td>151</td>
<td></td>
</tr>
<tr>
<td>AES-256</td>
<td>14</td>
<td>38858</td>
<td>88377</td>
<td>75.79</td>
<td>151.61</td>
<td>315</td>
<td></td>
</tr>
<tr>
<td>PRESENT-80</td>
<td>31</td>
<td>6059</td>
<td>14786</td>
<td>24.15</td>
<td>9.93</td>
<td>13.1</td>
<td></td>
</tr>
<tr>
<td>PRESENT-128</td>
<td>31</td>
<td>6852</td>
<td>15575</td>
<td>23.63</td>
<td>10.49</td>
<td>13.5</td>
<td></td>
</tr>
<tr>
<td>MIDORI-64</td>
<td>16</td>
<td>3251</td>
<td>7294</td>
<td>11.94</td>
<td>2.72</td>
<td>2.06</td>
<td></td>
</tr>
<tr>
<td>MIDAIRON-128</td>
<td>20</td>
<td>83308</td>
<td>19388</td>
<td>18.82</td>
<td>10.09</td>
<td>5.19</td>
<td></td>
</tr>
<tr>
<td>PRINCE</td>
<td>12</td>
<td>87908</td>
<td>6340</td>
<td>11.56</td>
<td>2.21</td>
<td>8.92</td>
<td></td>
</tr>
<tr>
<td>Orthros</td>
<td>12</td>
<td>9851</td>
<td>22307</td>
<td>10.60</td>
<td>6.57</td>
<td>1.90</td>
<td></td>
</tr>
<tr>
<td>SPEEDY-6</td>
<td>6</td>
<td>112098</td>
<td>25477</td>
<td>12.36</td>
<td>9.82</td>
<td>2.20</td>
<td></td>
</tr>
<tr>
<td>SPEEDY-7</td>
<td>7</td>
<td>107058</td>
<td>24536</td>
<td>15.02</td>
<td>9.45</td>
<td>2.58</td>
<td></td>
</tr>
<tr>
<td>ASCON-128-256</td>
<td>32</td>
<td>13835</td>
<td>30421</td>
<td>12.85</td>
<td>5.45</td>
<td>1.91</td>
<td></td>
</tr>
<tr>
<td>SKINNY-64-192</td>
<td>32</td>
<td>69696</td>
<td>15674</td>
<td>42.76</td>
<td>15.07</td>
<td>17.6</td>
<td></td>
</tr>
<tr>
<td>SKINNY-128-256</td>
<td>40</td>
<td>335075</td>
<td>76533</td>
<td>102.71</td>
<td>148.94</td>
<td>417</td>
<td></td>
</tr>
</tbody>
</table>

---

**Comparative Evaluation Metrics**

- **Area:** Area µm²
- **GE:** GE
- **Delay:** Delay ns
- **Power:** Power mW
- **FoM % AES:** FoM %

**Area optimized Latency optimized**

### Additional Details

- **Area optimized Latency optimized**
- **Area optimized:** Area µm²
- **Latency optimized:** Latency µm²
- **Area:** Area µm²
- **Power:** Power mW
- **FoM % AES:** FoM %

---

**Table 4:** Comparative Evaluation Metrics for the Nangate 15nm Process and Library

**Table 5:** Comparative Evaluation Metrics for the STM 90nm Process and Library
### Table 6: Comparative Evaluation Metrics for the TSMC 5nm Process and Library

<table>
<thead>
<tr>
<th>Cipher</th>
<th>Rounds</th>
<th>Area (\mu m^2)</th>
<th>Delay ps</th>
<th>FoM % AES</th>
<th>Area (\mu m^2)</th>
<th>Delay ps</th>
<th>FoM % AES</th>
</tr>
</thead>
<tbody>
<tr>
<td>AES-128</td>
<td>10</td>
<td>2304.1</td>
<td>28873</td>
<td>3064</td>
<td>100</td>
<td>4520.6</td>
<td>56648</td>
</tr>
<tr>
<td>AES-192</td>
<td>12</td>
<td>2635.4</td>
<td>33925</td>
<td>3686</td>
<td>138</td>
<td>5023.6</td>
<td>62952</td>
</tr>
<tr>
<td>AES-256</td>
<td>14</td>
<td>3238.7</td>
<td>40585</td>
<td>4290</td>
<td>197</td>
<td>6191.5</td>
<td>77587</td>
</tr>
<tr>
<td>PRESENT-80</td>
<td>31</td>
<td>812.0</td>
<td>10175</td>
<td>1836</td>
<td>42.2</td>
<td>1815.7</td>
<td>23723</td>
</tr>
<tr>
<td>PRESENT-128</td>
<td>31</td>
<td>848.8</td>
<td>10636</td>
<td>1841</td>
<td>44.2</td>
<td>1824.1</td>
<td>22858</td>
</tr>
<tr>
<td>MIDDLE-64</td>
<td>16</td>
<td>443.5</td>
<td>5557</td>
<td>921</td>
<td>10.6</td>
<td>761.8</td>
<td>9546</td>
</tr>
<tr>
<td>MIDDLE-128</td>
<td>20</td>
<td>1085.1</td>
<td>13597</td>
<td>1156</td>
<td>17.8</td>
<td>1954.5</td>
<td>24492</td>
</tr>
<tr>
<td>PRINCE-6</td>
<td>12</td>
<td>334.6</td>
<td>4193</td>
<td>710</td>
<td>6.74</td>
<td>672.1</td>
<td>8422</td>
</tr>
<tr>
<td>Orthros-6</td>
<td>12</td>
<td>1262.5</td>
<td>15820</td>
<td>681</td>
<td>12.2</td>
<td>2284.3</td>
<td>38624</td>
</tr>
<tr>
<td>SPEEDY-6</td>
<td>6</td>
<td>1795.5</td>
<td>22899</td>
<td>787</td>
<td>13.3</td>
<td>3133.8</td>
<td>39270</td>
</tr>
<tr>
<td>SPEEDY-7</td>
<td>7</td>
<td>2109.2</td>
<td>26431</td>
<td>924</td>
<td>18.4</td>
<td>3596.9</td>
<td>45107</td>
</tr>
<tr>
<td>ASCOR-12^2</td>
<td>12</td>
<td>2228.3</td>
<td>27923</td>
<td>826</td>
<td>26.1</td>
<td>2766.8</td>
<td>34671</td>
</tr>
<tr>
<td>SKIBLY-64-192</td>
<td>32</td>
<td>918.3</td>
<td>11507</td>
<td>1951</td>
<td>25.4</td>
<td>1682.0</td>
<td>21078</td>
</tr>
<tr>
<td>SKIBLY-128-128</td>
<td>40</td>
<td>3886.3</td>
<td>49953</td>
<td>4371</td>
<td>247</td>
<td>9241.0</td>
<td>115800</td>
</tr>
<tr>
<td>SKIBLY-128-384</td>
<td>40</td>
<td>4518.6</td>
<td>56960</td>
<td>4348</td>
<td>278</td>
<td>9527.5</td>
<td>119391</td>
</tr>
<tr>
<td>QARMA-64-σ9 (r = 5, PAC)</td>
<td>12</td>
<td>394.7</td>
<td>4946</td>
<td>728</td>
<td>8.14</td>
<td>707.0</td>
<td>8860</td>
</tr>
<tr>
<td>QARMA-64 (r = 7)</td>
<td>16</td>
<td>551.7</td>
<td>6913</td>
<td>1030</td>
<td>16.1</td>
<td>996.6</td>
<td>12489</td>
</tr>
<tr>
<td>QARMA-128 (r = 9)</td>
<td>20</td>
<td>1422.3</td>
<td>17823</td>
<td>1290</td>
<td>26</td>
<td>2535.8</td>
<td>31776</td>
</tr>
<tr>
<td>QARMA-128 (r = 11)</td>
<td>24</td>
<td>1635.6</td>
<td>20496</td>
<td>1561</td>
<td>36.2</td>
<td>3078.3</td>
<td>38575</td>
</tr>
<tr>
<td>QARMA-128-σ9 (r = 4)</td>
<td>10</td>
<td>309.7</td>
<td>3881</td>
<td>606</td>
<td>5.32</td>
<td>495.9</td>
<td>6214</td>
</tr>
<tr>
<td>QARMA-128-σ9 (r = 5)</td>
<td>12</td>
<td>374.6</td>
<td>4694</td>
<td>721</td>
<td>7.65</td>
<td>600.8</td>
<td>7529</td>
</tr>
<tr>
<td>QARMA-128-σ9 (r = 6)</td>
<td>14</td>
<td>435.4</td>
<td>5456</td>
<td>829</td>
<td>10.2</td>
<td>721.2</td>
<td>9038</td>
</tr>
<tr>
<td>QARMA-128 (r = 7)</td>
<td>16</td>
<td>560.5</td>
<td>7024</td>
<td>1048</td>
<td>16.6</td>
<td>1022.1</td>
<td>12808</td>
</tr>
<tr>
<td>QARMA-128 (r = 9)</td>
<td>20</td>
<td>704.5</td>
<td>8829</td>
<td>1295</td>
<td>25.9</td>
<td>1281.4</td>
<td>16058</td>
</tr>
<tr>
<td>QARMA-128-128 (r = 9)</td>
<td>20</td>
<td>1405.7</td>
<td>17615</td>
<td>1294</td>
<td>25.8</td>
<td>2533.9</td>
<td>31752</td>
</tr>
<tr>
<td>QARMA-128-128 (r = 11)</td>
<td>24</td>
<td>1689.5</td>
<td>21171</td>
<td>1540</td>
<td>36.9</td>
<td>3063.1</td>
<td>38365</td>
</tr>
<tr>
<td>QARMA-128-128 (r = 13)</td>
<td>28</td>
<td>1975.4</td>
<td>24755</td>
<td>1796</td>
<td>50.3</td>
<td>3577.3</td>
<td>44828</td>
</tr>
<tr>
<td>QARMA-128-256 (r = 15)</td>
<td>32</td>
<td>2259.6</td>
<td>28315</td>
<td>2042</td>
<td>65.4</td>
<td>4140.0</td>
<td>51879</td>
</tr>
</tbody>
</table>

**Remark 2.** An important observation here is that, as the process shrinks, the advantage of several ciphers over the AES decreases, as the FoM for QARMAv2 shows. There is a very simple explanation for this fact. The ciphers that rely on smaller and lighter S-Boxes need more rounds to achieve the same level of security as ciphers that use larger and deeper S-Boxes. As a result, more linear layers and key/tweak additions are needed. In turn, this implies a higher proportion of XOR operations and an increase of long wiring between cells. In particular the latter become progressively more expensive as the process shrinks, both in terms of feature size and the need for more buffer gates to counter increased leakage. A consequence is that providing implementation data for older processes, like 90nm or 180nm, does no longer provide an up-to-date assessment of the merits of a design in HW. This issues as well as dark silicon [EBA+12] should be taken into account by designers and implementers of cryptographic primitives to address near-future challenges.

### 6 Conclusions

In this paper we introduced the TBC family QARMAv2. It is a re-design of the QARMA TBC that reconsiders all components and adopts new key and tweak schedules, while keeping the overall structure of the data obfuscation path nearly unchanged. The latter point is a critical design choice, since the structure has proven itself solid borrowing elements from MIDORI (whose weaknesses were not due to the round function itself, but rather to the choice of round constants), and the reflection structure from PRINCE. The modified design allows for longer tweaks and no longer relies on time/data tradeoffs to define security levels. The choice of components is supported by extensive analysis of the individual elements and of the whole design. While this cannot rule out the potential for novel cryptanalytic attacks.
attacks, it should raise the bar for attackers.

Like QARMA, QARMAv2 is aimed at low-latency fully unrolled implementations. As such, it is ideally optimized for memory encryption, as well as for applications such as the generation of very short tags for hardware-assisted prevention of software exploitation, and the construction of keyed hash functions — bit it is suitable for general purpose use, except perhaps when a round-based implementation is required to be the smallest possible. We followed a conservative approach: for instance, we could have reduced the number of rounds, or used a lighter S-Box. Instead, we preferred to keep healthy security margins while still addressing the needs of the intended applications. Both our HW implementations and our cryptanalysis suggest that the goals have been met.

Finally, some of the techniques used in this paper may be of independent interest. This includes new MILP models of certain classes of diffusion matrices, the comparative analysis of a full reflection cipher against an iterative half-cipher, and our boomerang attack framework that includes the reflector in the middle section.

Our research raises some interesting questions. Is it possible to prove a correlation between the order of the cycles of a S-Box as a permutations and some form of its complexity (it could be multiplicative complexity, depth, or area)? The longer the cycles, the more difficult it seems to find lightweight S-Boxes, provided that the usual cryptographic properties are satisfied. A second question is whether better tweak schedules be found. For instance, is it possible that a more “randomly looking” combination of different shuffle maps may be more significantly more effective? We tried also other heuristics beside those in Section 3.4 on page 10, and these did not deliver better shuffles. The last, and perhaps more important matter, is how the design of lightweight cryptographic primitives can work around the issues of ever shrinking lithography processes.

Acknowledgements. The authors want to express their gratitude to Gary Gorman for providing the implementation data on the TSMC 5nm process, and to Andreas Sandberg for many interesting conversations on cryptographic memory protection. We thank Tim Beyne for a very useful discussion on nonlinear invariants of round functions. The authors also wholeheartedly thank Gurobi Optimization, LLC, for generously providing research licenses of their MILP Solver, and Arm Limited, for sponsoring cloud computing time.

Intellectual property rights statement. All the designs introduced in this document are placed in the public domain. The authors and their employers do not claim any intellectual property rights over the design. To the best of our knowledge no one else has any relevant intellectual property rights either.

Disclaimer. All information in this document is provided as is, and no guarantee or warranty is given or implied that the information is fit for any particular purpose. The user thereof uses the information at their sole risk and liability.

References


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


A MILP modeling

A.1 Modeling MixColumns

In this entire section we only consider the action of a matrix on a single column of the state, viewed as a vector. Let $x_i$, resp. $y_i$ for $0 \leq i < 4$ be the cells in a column vector, resp. the cells in the image of said column under MixColumns. The weight of a vector is the number of active cells in it. The weight of a transition $y = M \cdot x$ is the sum of the weights of $x$ and $y$.

In Fig. 8 we show the admissible transitions of Class I and II matrices. We now set to turn them into MILP models.

Figure 8: Differential properties of MixColumns.

A.1.1 Relations for both Class I and Class II Matrices

To model the fact that Class I and Class II matrices have branch number 4, including that an inactive input vector cannot map to an active output vector and vice versa, we introduce an integer auxiliary variable $d$ and add relations

$$x_i, y_i \leq d \quad \text{for} \quad 0 \leq i < 4, \quad \text{and} \quad \sum_i x_i + \sum_i y_i \geq 4d, \quad \sum_i x_i \geq d \quad \text{and} \quad \sum_i y_i \geq d. \quad (14)$$

The simple XOR model of MixColumns is given by the following relations for $0 \leq i < 4$:

$$y_i + \sum_{j \neq i} x_j \geq 2y_i \quad \text{and} \quad y_i + \sum_{j \neq i} x_j \geq 2x_j \quad \text{for all} \quad j' \neq i$$

$$x_i + \sum_{j \neq i} y_j \geq 2x_i \quad \text{and} \quad x_i + \sum_{j \neq i} y_j \geq 2y_j \quad \text{for all} \quad j' \neq i. \quad (15)$$

If the cell in row $k$ is active in an input, then at least one of the three cells in the rows $\neq k$ is active in output column (and the reverse direction is also true). This can be expressed as:

$$\sum_{i \neq k} y_i \geq x_k \quad \text{and} \quad \sum_{i \neq k} x_i \geq y_k \quad \text{for} \quad 0 \leq k < 4. \quad (16)$$

A further property of both Class I and Class II matrices is: For an active input cell in row $i$, and any output cell in a different row $j$, at least one among that output cell and the input cells in rows $\neq i, j$ must be active. (This constraint follows directly from the XOR model.) In equations:

$$y_j + \sum_{k \neq i, j} x_k \geq x_i \quad \text{and} \quad x_j + \sum_{k \neq i, j} y_k \geq y_i \quad \text{for all} \quad i \neq j, \quad 0 \leq i, j < 4. \quad (17)$$
A.1.2 Class I Specific Relations

It can also easily be seen that a Class I matrix satisfies

\[ x_i + x_j \geq y_i - y_j \quad \text{and} \quad y_i + y_j \geq x_i - x_j \quad \text{for all} \quad i \neq j, \quad 0 \leq i, j < 4, \]  

(18)

i.e. For any two rows \( i, j \), if output cell \( i \) is active and \( j \) is not, then at least one of the input cells in rows \( i \) and \( j \) must be active.

We can even express the fact that Class I matrices have no weight five transitions in a very compact way. To do this, we introduce an integer auxiliary variable \( d_i \) for each \( i \) and the constraints:

\[ \sum_i x_i + \sum_i y_i \leq 4 + 4d_i \quad \text{and} \quad \sum_i x_i + \sum_i y_i \geq 6d_i \quad \text{for} \quad 0 \leq i < 4. \]  

(19)

Indeed, if the total weight of an active transition is not four, then it must be at least five and \( d \geq 1 \) by the first relation, and by the second relation the total weight is at least six. If the total weight of the transition is four, then \( d \) must be 0 by the second relation.

Weight two columns only map to columns with the same active cells or to fully active columns. This is achieved by forcing the same cells to be active in this case and the fact that weight five transitions are not possible (cf. Relation (19)). Let \( d_{ij}, d'_{ij} \) be distinct integer auxiliary variables. Then, for all \( i, j \) with \( 0 \leq i < j < 4 \), the following relations must be satisfied:

\[ d_{ij} \geq x_i + x_j - \sum_{k \neq i,j} x_k - 1, \quad y_i \geq d_{ij} \quad \text{and} \quad y_j \geq d_{ij} \]

\[ d'_{ij} \geq y_i + y_j - \sum_{k \neq i,j} y_k - 1, \quad x_i \geq d'_{ij} \quad \text{and} \quad x_j \geq d'_{ij}. \]  

(20)

A.1.3 Class II Specific Relations

Relation (18) becomes a constraint for Class II matrices by replacing \( i \neq j \) with \(|i - j| = 2\).

A.1.4 Explicitly Excluding Transitions

Models for any diffusion matrix can be obtained by simply providing a list of forbidden transitions and adding a corresponding relation for each one of them. In order to model that a transition cannot occur, the corresponding relation is created as follows. Let \( I \subseteq \{0,1,2,3\} \), resp. \( J \subseteq \{0,1,2,3\} \) be the set of row of cells that are active in an input, resp. output column. We forbid the transition from \( \bigcup_{i \in I} x_i \) to \( \bigcup_{j \in J} y_j \) by adding the following relation

\[ \left( \sum_{i \in I} x_i + \sum_{i \notin I} (1 - x_i) \right) + \left( \sum_{j \in J} y_j + \sum_{j \notin J} (1 - y_j) \right) \leq 7. \]  

(21)

A diffusion matrix should not be modeled by this approach only, as the resulting model would be quite large and probably very slow to solve. Usually, the XOR model and the branch number are modeled first, then any other non-occuring transition is forbidden.

A.1.5 Models for Class I and Class II Matrices

Various equivalent models for Class I matrices can be obtained by combining various subsets of the above relations, for instance in the original \textsc{MANtis} and \textsc{QARma} papers the matrix was defined by relations (14), (16), (18) and (17) – in particular the XOR model is not used. This set of relations becomes a model for Class II matrices if in Relation
(18) we replace the condition \( i \neq j \) with \(|i - j| = 2\). These two models can be slightly accelerated by adding Relation (20).

Our starting point consisted of the XOR Relations (15) and the branch number Condition (14), removing the inadmissible transitions as described in Appendix A.1.4. For Class II matrices we exclude 8 transitions, producing a compact and fast model. However, for Class I matrices we must exclude 24 transitions, leading to a large, slow model.

For Class I matrices two alternative approaches result in fast models. The first approach is also the smallest and is obtained by using only Relations (15) and (14) and (19). A larger, but sometimes significantly faster model with Gurobi’s default settings, uses Relations (14), (19), (16) and (20). A possible explanation for the performance of the last model is that, eschewing the XOR representation of the matrix, it reduces the total amount of XORs in the MILP program, which are often considered the culprit for bad performance. However, using the smallest model with Gurobi’s parameters MIPFocus and Cuts both set to 2 results in the fastest solving times, especially for the largest models.

### A.2 Extending Solutions Inductively

For \( \ell = 1 \) and half-cipher, a MILP program for \( r \) is just an extension at the end of the corresponding program for \( r - 1 \). This ensures that the minimum active cell count in characteristics is an increasing function of \( r \) all other parameters being equal.

For \( \ell = 1 \) and full-cipher, it is clear that a MILP program for \( r \) is not an extension of the program for \( r - 1 \). This explains why sometimes the minimum active cell count in characteristics decreases for increasing \( r \). However, the program for \( r \) it is an extension of the program for \( r - 2 \), where the two additional rounds are added at each end of the cipher, therefore minimum active cell count is a monotonic function of \( r \) restricted to the even or to the odd \( r \). Furthermore, a MILP program for \( r \) is an extension of the corresponding program for \( r - 1 \) and \( \varphi \) replaced by its inverse \( \varphi^{-1} \) (and the roles of the two tweak blocks exchanged). So if one merges the counts for a given \( \varphi \) with, say, even \( r \) with the counts for \( \varphi^{-1} \) for odd \( r \) we obtain a monotonic sequence of minimum active cell counts.

For \( \ell = 2 \), because of the eXchangeRows operations every other round, a MILP program for \( r \) is always an extension of the corresponding program for \( r - 2 \).

We exploit these properties to provide initial starting solutions to the MILP solvers.

### B Tables of Trail Weights

In Table 8 on the next page, resp. Table 9 we tabulate the weights of optimal linear and related-tweak differential trails for both half-cipher and full-cipher QARMAv2-64, resp. QARMAv2-128 for various values of \( r \) and various choices of the tweak update function. Tables 10 and 11 are the corresponding tables for the case of single block tweak.

| Table 7: The six shuffles \( \tau_{d1} \) to \( \tau_{d6} \) |
|-----------------|-----------------|-----------------|-----------------|-----------------|
| \( \tau_{d1} \) : | 2, 10, 14, 6, 0, 8, 12, 4, | 3, 11, 15, 7, 1, 9, 13, 5 |
| \( \tau_{d2} \) : | 1, 9, 13, 5, 0, 8, 12, 4, | 2, 10, 14, 6, 3, 11, 15, 7 |
| \( \tau_{d3} \) : | 3, 11, 15, 7, 2, 10, 14, 6, | 0, 8, 12, 4, 1, 9, 13, 5 |
| \( \tau_{d4} \) : | 2, 10, 14, 6, 1, 9, 13, 5, | 0, 8, 12, 4, 3, 11, 15, 7 |
| \( \tau_{d5} \) : | 3, 11, 15, 7, 1, 9, 13, 5, | 2, 10, 14, 6, 0, 8, 12, 4 |
| \( \tau_{d6} \) : | 1, 9, 13, 5, 2, 10, 14, 6, | 3, 11, 15, 7, 0, 8, 12, 4 |
Table 8: Active cell counts for tweak-related differential characteristics and linear trails ($\ell = 1$, with two independent tweak blocks)

<table>
<thead>
<tr>
<th>$r$</th>
<th>Half Cipher</th>
<th>Full Cipher</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\tau$</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>$\tau^2$</td>
<td>2</td>
<td>6</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Related-tweak Differential for given $\phi$</th>
<th>Half Cipher</th>
<th>Full Cipher</th>
</tr>
</thead>
<tbody>
<tr>
<td>$h$</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>$h^{-1}$</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>$h_4$</td>
<td>2</td>
<td>4</td>
</tr>
<tr>
<td>$h_{4c}$</td>
<td>2</td>
<td>4</td>
</tr>
<tr>
<td>$h_5$</td>
<td>2</td>
<td>4</td>
</tr>
<tr>
<td>$h_8$</td>
<td>2</td>
<td>4</td>
</tr>
</tbody>
</table>

Table 9: Active cell counts for tweak-related differential characteristics and linear trails ($\ell = 2$, with two independent tweak blocks)

<table>
<thead>
<tr>
<th>$r$</th>
<th>Half Cipher</th>
<th>Full Cipher</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\tau$</td>
<td>2</td>
<td>4</td>
</tr>
<tr>
<td>$\tau^2$</td>
<td>2</td>
<td>4</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Related-tweak Differential</th>
<th>Half Cipher</th>
<th>Full Cipher</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\tau_{d1}$</td>
<td>2</td>
<td>4</td>
</tr>
<tr>
<td>$\tau_{d2}$</td>
<td>2</td>
<td>5</td>
</tr>
<tr>
<td>$\tau_{d3}$</td>
<td>2</td>
<td>5</td>
</tr>
<tr>
<td>$\tau_{d4}$</td>
<td>2</td>
<td>4</td>
</tr>
<tr>
<td>$\tau_{d5}$</td>
<td>2</td>
<td>4</td>
</tr>
<tr>
<td>$\tau_{d6}$</td>
<td>2</td>
<td>4</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Related-tweak Differential</th>
<th>Half Cipher</th>
<th>Full Cipher</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\tau_{f1}$</td>
<td>2</td>
<td>4</td>
</tr>
<tr>
<td>$\tau_{f2}$</td>
<td>2</td>
<td>4</td>
</tr>
<tr>
<td>$\tau_{h}$</td>
<td>2</td>
<td>4</td>
</tr>
<tr>
<td>$\tau_{h4}$</td>
<td>2</td>
<td>4</td>
</tr>
</tbody>
</table>

| Linear | 16 | 23 | 30 | 35 | 38 | 41 | 50 | 57 | 62 | 67 | 5 | 32 | 50 | 64 | 72 |

<table>
<thead>
<tr>
<th>$r$</th>
<th>Half Cipher</th>
<th>Full Cipher</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\tau$</td>
<td>2</td>
<td>4</td>
</tr>
<tr>
<td>$\tau^2$</td>
<td>2</td>
<td>6</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Related-tweak Differential</th>
<th>Half Cipher</th>
<th>Full Cipher</th>
</tr>
</thead>
<tbody>
<tr>
<td>$h$</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>$h_4$</td>
<td>2</td>
<td>4</td>
</tr>
<tr>
<td>$\tau_f$</td>
<td>2</td>
<td>4</td>
</tr>
</tbody>
</table>

| Linear | 16 | 25 | 36 | 48 | 58 | 68 | 72 | 80 | 88 | 100 | 24 | 44 | 56 | 80 | 104 |
Table 10: Active cell counts for tweak-related differential characteristics
and linear trails (ℓ = 1, t = 1 with tweak blocks tied near center)

<table>
<thead>
<tr>
<th>r</th>
<th>Half Cipher</th>
<th>Full Cipher</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>3 4 5 6 7 8 9 10 11 12</td>
<td>2 3 4 5 6</td>
</tr>
<tr>
<td>4</td>
<td>4\frac{1}{2} 5\frac{1}{2} 6\frac{1}{2} 7\frac{1}{2} 8\frac{1}{2} 9\frac{1}{2} 10\frac{1}{2} 11\frac{1}{2} 12\frac{1}{2}</td>
<td>4\frac{1}{2} 6\frac{1}{2} 8\frac{1}{2} 10\frac{1}{2} 12\frac{1}{2}</td>
</tr>
<tr>
<td>5</td>
<td>5 6 7 8 9 10 11 12</td>
<td>6 12 16 20 24</td>
</tr>
<tr>
<td>6</td>
<td>6 7 8 9 10 11 12</td>
<td>6 22 24 36 36</td>
</tr>
<tr>
<td>7</td>
<td>7 8 9 10 11 12</td>
<td>8 20 24 32 36</td>
</tr>
<tr>
<td>8</td>
<td>8 9 10 11 12</td>
<td>8 20 31 38 47</td>
</tr>
<tr>
<td>9</td>
<td>9 10 11 12</td>
<td>6 22 33 38 45</td>
</tr>
<tr>
<td>10</td>
<td>10 11 12</td>
<td>6 24 32 39 47</td>
</tr>
<tr>
<td>11</td>
<td>11 12</td>
<td>6 24 32 39 47</td>
</tr>
<tr>
<td>12</td>
<td>12</td>
<td>6 24 32 39 47</td>
</tr>
</tbody>
</table>

Table 11: Active cell counts for tweak-related differential characteristics
and linear trails (ℓ = 2, t = 1 with tweak blocks tied near center)

<table>
<thead>
<tr>
<th>r</th>
<th>Half Cipher</th>
<th>Full Cipher</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>3 4 5 6 7 8 9 10 11 12</td>
<td>2 3 4 5 6</td>
</tr>
<tr>
<td>4</td>
<td>4\frac{1}{2} 5\frac{1}{2} 6\frac{1}{2} 7\frac{1}{2} 8\frac{1}{2} 9\frac{1}{2} 10\frac{1}{2} 11\frac{1}{2} 12\frac{1}{2}</td>
<td>4\frac{1}{2} 6\frac{1}{2} 8\frac{1}{2} 10\frac{1}{2} 12\frac{1}{2}</td>
</tr>
<tr>
<td>5</td>
<td>5 6 7 8 9 10 11 12</td>
<td>6 16 20 24 28</td>
</tr>
<tr>
<td>6</td>
<td>6 7 8 9 10 11 12</td>
<td>6 24 24 36 36</td>
</tr>
<tr>
<td>7</td>
<td>7 8 9 10 11 12</td>
<td>8 24 24 36 36</td>
</tr>
<tr>
<td>8</td>
<td>8 9 10 11 12</td>
<td>10 16 30 47 56</td>
</tr>
<tr>
<td>9</td>
<td>9 10 11 12</td>
<td>10 16 30 47 56</td>
</tr>
<tr>
<td>10</td>
<td>10 11 12</td>
<td>10 16 30 47 56</td>
</tr>
<tr>
<td>11</td>
<td>11 12</td>
<td>10 16 30 47 56</td>
</tr>
<tr>
<td>12</td>
<td>12</td>
<td>10 16 30 47 56</td>
</tr>
</tbody>
</table>

C Characteristics for Differential, Impossible Differential,
and Boomerang Attacks

We provide detailed characteristics for several of the cryptanalytic results of Section 4.1.
Fig. 9 illustrates how a key-recovery attack on \(r = 6\) rounds of QARMAv2-64 could be
designed based on a characteristics following the optimal truncated differential pattern
for \(r = 4\). Fig. 10 illustrates the longest impossible differential distinguisher for QARMAv2-64
under a cell-based miss-in-the-middle approach. Figures 11–13 show several potential
boomerang distinguishers identified by a cell-wise model for a total of \(10^2\) rounds. The
more balanced setups yield lower bounds.

D A Remark on Minimal Weight Linear Trails

In Fig. 14 on page 44 we see an example of a minimal weight iterative cell-based linear trail
for single-layer QARMAv2. Several similar trails exist, all with and \(1 + 3 + 9 + 9 + 9 + 3 = 34\)
active cells over 6 rounds (if a weight nine state is in the last round, it and possibly also
the previous state may be replaced by lighter ones, giving a 6-round trail of weight 30).
This results in an average of \(17/3 \approx 5.67\) cells per round, whereas an MDS matrix gives
\(1 + 4 + 16 + 4 = 25\) cells over 4 rounds, i.e. on average 6.25 per round. This means that, in
order to have a comparable minimal number of active cells, a design whose linear layer is
based on a a Class I Almost MDS matrix over a ring \(R_m\) and the MIDORI StateShuffle.
Figure 9: Sketch of key recovery attack on QARMAv2 with $r = 6$ based on differential distinguisher for $r = 4$. 

- **Involved in key recovery**
- **Active cell**
- **Active S-box in distinguisher**
Figure 10: Longest impossible differential distinguisher for QARMAv2

- Upper differential trail
- Lower differential trail
- Nonzero difference
- Fixed (single) difference
- Unknown difference

The Tweakable Block Cipher Family QARMAv2
Figure 11: Boomerang distinguisher of weight 73 for $6\frac{1}{2} + 4\frac{1}{2}$ rounds of QARMA\textsuperscript{v2-64}, corresponding to a probability bound of $2^{-10 \cdot 4 \cdot \left(\frac{10}{16}\right)^{13}} \approx 2^{-48.8}$. 

| active cell in $E_0$ | active cell in $E_1$ | upper characteristic in $E_m$ | lower characteristic in $E_m$ | active S-box in $E_1 / E_1$ or jointly active S-box in $E_m$ |
Figure 12: Boomerang distinguisher of weight 82 for 5.12 + 5.12 rounds of QARMAv2, corresponding to a probability bound of $2^{-55.1}$. 

Figure 12: Boomerang distinguisher of weight 82 for 5.12 + 5.12 rounds of QARMAv2, corresponding to a probability bound of $2^{-55.1}$. 

$\approx \mathbb{P} \left( \frac{1}{2} \right)$.
Figure 13: Boomerang distinguisher weight 60 for $9\frac{1}{2} + 1\frac{1}{2}$ rounds of QARMAv2-64, corresponding to a probability bound of $2^{-5 \times 4 \cdot \left(\frac{10}{16}\right)^{30}} \approx 2^{-40.3}$.
needs $\left(\frac{25}{17}\right) - 1 \approx 10\%$ more rounds than a design based on a MDS matrix over a finite field and an AES-like ShiftRows, all other things being equal. However, the rounds of the latter are much lighter, especially if the S-Boxes are small, and therefore it may still result in a smaller and faster design.

E Full Diffusion Property for the Two-Layer Version

A four-round diffusion property for the $\ell = 2$ version of the cipher can be easily proved with a simple computer program, which can be easily made to output the TikZ source of the selected trails displayed in Fig. 15. In the figure we show only the four trails corresponding to the first row in the case where eXchangeRows occurs after the second round, and the corresponding ones in the case where eXchangeRows occurs after the first round. The remaining trails are similar.

Now, these properties hold with all S-Boxes $\sigma_0$, $\sigma_1$, $\sigma_2$ and $\sigma$ as well as with the corresponding 8-bit versions. It is easy to verify: assume only one bit is active in the single cell in the first state. After the first $M$ there is an S-Box layer and therefore there are at least 3 non-linearly affected output bits per affected cell - 4 if the S-Box is not $\sigma_0$. After the second $M$-layer there is a second S-Box layer, and the number non-linearly affected output bits per affected cell is always at least 4, and it is at least 7 with the 8-bit S-Boxes. After the S-Box layer following the third $M$-layer the number non-linearly affected output bits per affected cell is always 8 for the 8-bit S-Boxes, and after the fourth last $M$-layer all cells are thusly affected.

F Version for Pointer Authentication and Memory Integrity

For the computation of Pointer Authentication Codes (PACs) and memory integrity tags (following [JLK+23]) we instantiate $\text{QARMAv2-64}$ with $\text{QARMA}$’s S-Box $\sigma_0$ [Ava17], i.e.:

$$\sigma_0 := [0 \ E \ 2 \ A \ 9 \ F \ 8 \ B \ 6 \ 4 \ 3 \ 7 \ D \ C \ 1 \ 5] .$$

If the tweak has a length of only one block, or, in the PAC instruction set terminology, only one "salt" is used, then we put $T_1 = \tau^{-1}(T_0)$. The round constants for this version are the consecutive digits of the hexadecimal expansions of the fractional part of the constant $\pi$ as in $\text{QARMA}$. The values of the parameter $r$ is 6 for memory integrity and this is also the recommended value for pointer authentication. If $r = 6$ cannot be reached for pointer authentication without significant performance penalties, for instance on small in-order cores (like some low end R-class Arm cores), then it is admissible to use $r = 5$ or $r = 4$.

We now briefly describe how $\text{QARMAv2-64-}\sigma_0$ can be used for memory integrity. Let $M_0|M_1|\cdots|M_{r-1}$ be a cache line partitioned in 64-bit blocks. Depending on whether the
memory encryption algorithm offers freshness or not, we can use one of two integrity algorithms depicted in Figs. 16 and 17 on the next page. They are “tPMACs”, i.e. tweakable Parallelizable MACs. The first one is for the case where the system does not provide freshness to encryption, and the second one is to be used when freshness is available. In the diagrams, \( \alpha_i \) is the physical address of the block that is being encrypted as a contribution to the memory region’s tag, and \( \nu \) is the so-called freshness information.

We say that an encryption or authentication function provides temporal uniqueness, or freshness, when repeated writes of the same plaintext to the same location result in different outputs. This is achieved by associating a counter with each cache line and including it in the computation of the function. An encryption function providing freshness suffers from text expansion, but at the same time it allows shallower critical paths.

The output of the QARMAv2 function needs to be truncated to be used as a PAC. To generate a \( z \)-bit tag with \( 1 \leq z \leq 32 \), first reorder the output bits as follows:

\[
[0, 8, 16, 24, 32, 40, 48, 56] + [0, 8, \ldots, 56] + [0, 8, \ldots, 56] + [0, 8, \ldots, 56].
\]

Then, pick the first \( z \) bits in this sequence.
The Tweakable Block Cipher Family QARMAv2

G Wider Versions

Here we describe how to design versions of QARMAv2 with 256- and 512-blocks: these are not meant to be “official” constructions of the QARMAv2 family, but rather just examples of how to further extend and explore the ideas presented in this paper.

These are based on the two-layer design and employ wider 8-bit and 16-bit S-Boxes. The 8-bit S-Box is constructed exactly as in QARMA, whereas the 16-bit S-Box is constructed by placing four copies of \( \sigma_1 \) side by side, operating respectively on bits \([0..3], [4..7], [8..11]\) and \([12..15]\). The outputs are then intertwined by sending output bit \( 4i + j \) to bit \( i + 4j \).

The full diffusion properties hold unchanged because if we consider just one bit in a cell at the beginning, after the next S-Box layer it will have non-linearly affected four bits in each cell it affects, and after the third S-Box layer eight, or sixteen bits.

Key and tweak is in both cases a single block, so \( K_0 = K_1 = K \). Keys and tweaks shorter than 256, resp. 512 bits are zero-padded.

For the tweak schedule, we “tie” the values of the two tweak blocks just before the centre for encryption, namely \( t_r = t_{r-1} \). For both even and odd \( r \), the relation between the input tweak blocks is thus \( T_0 = \varphi^{-1}(T_1) \). The security claims are the same as for QARMAv2-128, for \( r \geq 13 \), even though the tweak is constrained.
H Test Vectors

H.1 QARMAv2-64

\[ P = 0000000000000000 \]

\[ K_0,K_1 = 0123456789abcdef,fedcba9876543210 \]

\[ T_0,T_1 = 7e5c3a18f6d4b290,1eb852fc9630da74 \]

\[ C = a49c0a683065cbc1 \]

H.2 QARMAv2-128

\[ P = 00000000000000000000000000000000 \]

\[ K_0,K_1 = 0010203040506070809a0b0c0d0e0f0,0f0e0d0c0b0a09080706050403020100 \]

\[ T_0,T_1 = 7e5c3a18f6d4b290e5c3a18f6d4b2907,1eb852fc630da741b852fc960da741eb \]

H.2.1 QARMAv2-128-128

\[ C = 8088b7d14c7e014df984d508bf6ed5dd \]

H.2.2 QARMAv2-128-192

\[ C = 3726d4269342f14827b68d11d42e24d6 \]

H.2.3 QARMAv2-128-256

\[ C = c470640c3d31cb0cf8a19ee1a016e934 \]

H.3 QARMAv2-64-\( \sigma_0 \)

\[ P = 00000000000000000000000000000000 \]

\[ K_0,K_1 = 0123456789abcdef,fedcba9876543210 \]

\[ T_0,T_1 = 7e5c3a18f6d4b290,1eb852fc9630da74 \]

H.3.1 \( r = 4 \)

\[ C = cd4911ecd3d4de33 \]

H.3.2 \( r = 5 \)

\[ C = b3e402122fe60820 \]

H.3.3 \( r = 6 \)

\[ C = 2b590ec5954eaa43 \]