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{4}, 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
\texttt{subhadeep.banik@usi.ch, regazzoni@alari.ch}
\textsuperscript{4} Computer Science Department, University of Haifa, Israel
\texttt{orrd@cs.haifa.ac.il, sghosh03@campus.haifa.ac.il}
\textsuperscript{5} Graz University of Technology, Austria
\texttt{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 · Pointer Authentication · Short Hashes

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.
Contents

1 Introduction 3

2 Definition of the Cipher 5

3 Choices of the Functions 9
   3.1 Orthomorphisms 9
   3.2 The Almost-MDS Matrix 10
   3.3 The StateShuffle 10
   3.4 Tweak Shuffles 11
   3.5 The Round Constants 14
   3.6 The S-Box 14
   3.7 Alternative Constructions 15

4 Cryptanalysis 16
   4.1 Differential Cryptanalysis and Boomerang Attacks 16
   4.2 Linear Cryptanalysis 19
   4.3 Impossible-Differential and Zero-Correlation Cryptanalysis 19
   4.4 Integral Cryptanalysis and the Division Property 20
   4.5 Slide, Meet-in-the-Middle and other Structural Attacks 20
   4.6 Invariant Subspace Cryptanalysis and Non-Linear Invariants 21
   4.7 Algebraic Attacks 21
   4.8 Security Implications of the Central Construction 21
   4.9 Security Parameters and Security Claims 22
   4.10 Comparison to QARMA 22

5 Hardware Implementation and Evaluation 22

6 Conclusions 26

A MILP Modelling 32
   A.1 Modelling MixColumns 32
   A.2 Extending Solutions Inductively 35

B Tables of Trail Weights 35

C A Remark on Minimal Weight Linear Characteristics 35

D Full Diffusion Property for the Two-Layer Version 37

E Characteristics for Differential, Impossible Differential, and Boomerang Attacks 40

F Version for Pointer Authentication and Memory Integrity 40

G Wider Versions 46

H Test Vectors 48
   H.1 QARMAv2-64 48
   H.2 QARMAv2-128 48
   H.3 QARMAv2-64-σ0 48
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 [KPV16], 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 PRINCEv2 [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 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].

The subject of this paper, the TBC 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 not only achieved a better understanding of how to design block ciphers, but also of the requirements on them coming from practical applications. The first understanding leads us, for instance, to adopt a different key schedule. The second one leads us to increase the maximal tweak size. We now briefly explain the need for longer tweaks.

Let us start looking at AES in XTS mode. It only allows the use of a 128-bit nonce, since it is simply encrypted and added to both plaintext and ciphertext. 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. The issue here is that the block size of the underlying cipher determines the space for the tweaks as well, and thus the space of the functions that are parametrised, for a fixed key, by the tweak. Ciphers with 256-bit blocks have been suggested [JSV17]. They would allow longer IVs, highlighting a potential limitation of the standardized AES parameters.

QARMAv2-128’s 256-bit tweaks (assuming that they are properly integrated in the design) allow the use of random or synthetic nonces/IVs without having to worry about tweak reuse. A tweaked codebook mode could be used in place of the XEX construction, where a 32-bit counter is appended to a 224-bit IV. For a counter mode, the whole 256-bit tweak can be used for the IV and the text input for the counter. The large tweak permits the use of random or synthetic IVs without having to worry for repeated functions. Since collisions between 256-bit random values are practically impossible, we note that 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 lack of repeats will make the keystream distinguishable.
from a random sequence. The smaller block size carries advantages: Potentially faster full diffusion with respect to, say, a 256-bit otherwise similar block cipher means shorter distinguishers to be used in attacks — hence fewer overall rounds, and lower latency.

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 QARMAv2/PAC, 128-bit tweaks are needed for improved versions of the PAC feature where the tag on a pointer is computed using two different “salts,” which can be two full 64-bit pointers, or a full pointer and a counter, or some other form of context.

Finally, we allow the use of shorter keys for QARMAv2-128. For general-purpose use cases, QARMAv2-128-256’s 256-bit key offers ample security margins. We define 128- and 192-bit security levels for QARMAv2-128 to align with the AES and to allow deployers to opt for further latency and area savings while keeping a solid security level.

Use cases, security model, and 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 optimised 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 tweak 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 practically meaningful. In most applications we can assume that the key, secret, is not changed often. Thus, its generation can be hardened without a significant impact on a system’s performance, so that its security model does not need to include related-value attacks. On the other hand, the public tweak is changed often and its security model includes adversary-known or even chosen tweaks, hence related-value attacks are definitely in scope. This implies that designing the tweak into the cipher needs more attention than keying. Using the same schedule for both key and tweak risks to be either overkill for the former or too weak for the latter.

These considerations prompt us to keep separate key and tweak schedules, as in QARMA. As we do not consider related-key attacks, but we take related-tweak attacks into account, we can keep a very simple key schedule, where orthomorphisms and round constants are used to harden against invariant subspace attacks, slide attacks, and structural attacks. However, the tweak must be included in the design in a way that does not impact its security adversely and the total area and latency of the cipher are well behaved. A further constraint is for the tweak schedule to be as lightweight as possible: The construction of the latter is the most critical part of the design of QARMAv2.

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 analyse 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 Area-Delay and Energy consumption values than the AES. The reduced area and latency can provide significant power savings, especially in always-on applications such as memory encryption, or securing communication channels. The cipher is flexible enough to be deployed on microcontrollers or on high-end CPUs, with accordingly different security levels, and can be used to compute short tags of short messages with very low latency.

Outline. In Section 2, we present the specification of QARMAv2. In Section 3, we discuss the design rationale. In Section 4, we evaluate the resistance of the design to various cryptanalytic approaches. In Section 5, we provide hardware implementation results.

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.

2 Definition of the Cipher

The design of QARMAv2 follows the reflector construction of PRINCE, MANTIS and QARMA, represented in Fig. 1 on the following page: 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 on the next page, we represent the initial and final rounds separately: They consist of just a key addition and an S-Box layer, 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 $\mathcal{F}$ 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 $\mathcal{F}$, is an S-Box layer.
A bar over a function denotes its inverse, for instance the inverse of \( f \) is \( \overline{f} \).

We give the Algorithm as pseudo-code in Fig. 2 on the facing page, and graphically in Fig. 3 on page 8 for odd \( r \). The notation is explained next.

Cells, blocks, layers and the internal state. In Fig. 2 on the facing page, \( S \) 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 \( \ell \) layers. 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:

\[
\mathcal{L} = c_0^{(i)} | c_1^{(i)} | \cdots | c_{15}^{(i)} = \begin{pmatrix} c_0 & c_1 & c_2 & c_3 \\ c_4 & c_5 & c_6 & c_7 \\ c_8 & c_9 & c_{10} & c_{11} \\ c_{12} & c_{13} & c_{14} & c_{15} \end{pmatrix} = \begin{pmatrix} c_{0,0} & c_{0,1} & c_{0,2} & c_{0,3} \\ c_{1,0} & c_{1,1} & c_{1,2} & c_{1,3} \\ c_{2,0} & c_{2,1} & c_{2,2} & c_{2,3} \\ c_{3,0} & c_{3,1} & c_{3,2} & c_{3,3} \end{pmatrix} .
\]

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.

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 followe

\[
\mathcal{L} = \begin{pmatrix} c_0 & c_1 & c_2 & c_3 \\ c_4 & c_5 & c_6 & c_7 \\ c_8 & c_9 & c_{10} & c_{11} \\ c_{12} & c_{13} & c_{14} & c_{15} \end{pmatrix} \xrightarrow{\tau} \begin{pmatrix} c_0 & c_{11} & c_6 & c_{13} \\ c_{10} & c_1 & c_{12} & c_7 \\ c_5 & c_{14} & c_3 & c_8 \\ c_{15} & c_4 & c_9 & c_2 \end{pmatrix} = \tau(\mathcal{L}) .
\]

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.

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.
Algorithm \text{QARMA}v2, \([K_0, K_1, W_0, W_1, T_0, T_1, P] \mapsto C\):

1: \(RT_0 \leftarrow T_0\) \hspace{1cm} (Round tweak setup)
2: \(RT_1 \leftarrow T_1\)
3: \(\mathcal{F} \leftarrow P + K_0\) \hspace{1cm} (Round \# 0)
4: \(\mathcal{F} \leftarrow S(\mathcal{F})\)
5: for \(i = 1\) to \(r\) do \hspace{1cm} (Rounds from \#1 to \#r)
6: \(\mathcal{F} \leftarrow \mathcal{F} + K_{i \mod 2} + RT_{i \mod 2} + \tau_i\)
7: \(\mathcal{F} \leftarrow (S \circ M \circ \tau)(\mathcal{F})\)
8: if \(i \equiv 1 \mod 2\) then \(RT_1 \leftarrow \varphi(\mathcal{F})\) else \(RT_0 \leftarrow \varphi(\mathcal{F})\)
9: \{ if \(i \equiv r \mod 2\) then \(\mathcal{F} \leftarrow \text{XR}(\mathcal{F})\) \}
10: \(L_0 \leftarrow o(K_0) + \alpha\) \hspace{1cm} (Round key transform)
11: \(L_1 \leftarrow S(K_1 + \alpha)\)
12: \(\mathcal{F} \leftarrow \tau(\mathcal{F})\) \hspace{1cm} (Central construction)
13: \(\mathcal{F} \leftarrow M \cdot (\mathcal{F} + W_{r+1 \mod 2}) + W_{r \mod 2}\)
14: \(\mathcal{F} \leftarrow \tau(\mathcal{F})\)
15: for \(i = r\) down to \(1\) do \hspace{1cm} (Rounds from \#r + 1 to \#2r)
16: \{ if \(i \equiv r \mod 2\) then \(\mathcal{F} \leftarrow \text{XR}(\mathcal{F})\) \}
17: \(\mathcal{F} \leftarrow \tau(\mathcal{F})\)
18: \(\mathcal{F} \leftarrow \mathcal{F} + L_{i+1 \mod 2} + RT_{i+1 \mod 2} + \tau_i\)
19: if \(i > 1\) and \(i \equiv 0 \mod 2\) then \(RT_1 \leftarrow \varphi(\mathcal{F})\) else \(RT_0 \leftarrow \varphi(\mathcal{F})\)
20: \(\mathcal{F} \leftarrow \varphi(\mathcal{F})\) \hspace{1cm} (Round \#2r + 1)
21: \(C \leftarrow \mathcal{F} + L_1\)
22: return \(C\)

Figure 2: The \text{QARMA}v2 Algorithm

A full round has the following structure:

\[
\begin{align*}
\text{StateShuffle} & \hspace{1cm} \text{i.e.} \hspace{1cm}
\begin{array}{c}
\begin{array}{c}
\scriptstyle x \\
\scriptstyle \tau
\end{array}
\end{array}
\begin{array}{c}
\begin{array}{c}
\scriptstyle \tau
\end{array}
\end{array}
\begin{array}{c}
\begin{array}{c}
\scriptstyle M
\end{array}
\end{array}
\begin{array}{c}
\begin{array}{c}
\scriptstyle X
\end{array}
\end{array}
\begin{array}{c}
\begin{array}{c}
\scriptstyle y
\end{array}
\end{array}
\end{array}
\end{align*}
\]

where \(\mathcal{R} = S \circ M \circ \tau\), The letters \(k\), resp. \(t\) denote a round key, resp. tweak block, and \(\tau\) is the \text{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 \text{eXchangeRows} operation, which applies only for \(\ell = 2\). The \text{eXchangeRows} operation swaps the first two rows between the two layers. It performed every other full round, with the restriction that two \text{eXchangeRows}’s always flank the central construction (see below). \(X\) and \(S\) commute with each other.

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.

The Central Construction. The central construction is similar to that of \text{QARMA}:

\[
\begin{align*}
\begin{array}{c}
\begin{array}{c}
\scriptstyle x
\end{array}
\end{array}
\begin{array}{c}
\begin{array}{c}
\scriptstyle \tau
\end{array}
\end{array}
\begin{array}{c}
\begin{array}{c}
\scriptstyle M
\end{array}
\end{array}
\begin{array}{c}
\begin{array}{c}
\scriptstyle \tau
\end{array}
\end{array}
\begin{array}{c}
\begin{array}{c}
\scriptstyle y
\end{array}
\end{array}
\end{array}
\end{align*}
\]

where \(w_0, w_1\) are two round key blocks. This function can be implemented using the original \text{QARMA} central construction, by using the round key \(M \cdot w_0 + w_1\).
The tweakable block cipher family QARMAv2

The round constants. In Section 3.5 on page 14 we define the sequence \((\epsilon_i)_{i=0}^\infty\) of single block wide round constants with \(\epsilon_0 = \epsilon_1 = 0\), as well as an additional single block wide constant \(\alpha\).

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 alternately added to the state. For encryption they are initialised with the values \(K_0\) and \(K_1\). During the central construction, they are subject to the transformation

\[
\iota : (K_0, K_1) \mapsto (L_0, L_1) := \left(\sigma(K_0) + \alpha, \sigma^{-1}(K_1 + \alpha)\right),
\]

where \(\sigma\) is the PRINCE orthomorphism (3), defined in Section 3.1 on the facing page. Then, \(L_0\) and \(L_1\) are the round keys alternatingly added to the backward rounds.

Note that the values \(K_i\), \(W_i\), and \(L_i\) have been indexed to represent clearly that we are alternating the addition of round keys derived from \(K_0\) and \(K_1\). The tweak blocks \(T_0\) and \(T_1\) will adopt 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

\[
\iota(L_1, L_0) = \iota(\sigma^{-1}(K_1 + \alpha, \sigma(K_0) + \alpha)) = \left(\sigma(\sigma^{-1}(K_1 + \alpha)) + \alpha, \sigma^{-1}(\sigma(K_0) + \alpha + \alpha)\right) = (K_1, K_0) .
\]

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 \(K_1\), and with the schedule exactly as for the encryption, provided we also 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 \(W_0 = \sigma^{-2}(K_1)\). This alone is not sufficient to allow the use of the same circuit for encryption and decryption, because we have not yet taken the tweaks into account. This will be done next.

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.

---

**Figure 3:** QARMAv2 for odd \(r\). If \(r\) is even, \(W_0\) and \(W_1\) are swapped and the forward function starts with \(R\) instead of \(R\{X\}\).
the forward and backward rounds unchanged, i.e., without undoing the update steps as in QARKA. 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 $t_{2i} = \varphi(t_{2i-2})$ and $t_{2i+1} = \varphi^{-1}(t_{2i-1})$ for some function $\varphi$. Starting with $t_1 = T_1$ and $t_2 = \varphi^{-1}(T_0)$, we obtain the sequence 
\[
[T_1, \varphi^{-1}(T_0), \varphi(T_1), \varphi^{-2}(T_0), \varphi^2(T_1), \varphi^{-3}(T_0), \ldots, \varphi^r(T_1), T_0]
\]
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. The function $\varphi$ is chosen in Section 3.4 on page 11.

**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 $W_0 = \sigma^2(K_0)$, $W_1 = \sigma^{-2}(K_1)$:

\[
[K_0, K_1, W_{r+1 mod 2}, W_{mod 2}, \varphi^{-1}(T_0), T_1, PT]
\]

- For decryption, the input vector is set to

\[
[\sigma^{-1}(K_1 + \alpha), \sigma(K_0) + \alpha, W_{mod 2}, W_{r+1 mod 2}, \varphi^{-r}(T_1), T_0, CT]
\]

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.

**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 Subsection 4.9. Therefore, also the test vectors given in Appendix H on page 48 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.

## 3 Choices of the Functions

### 3.1 Orthomorphisms

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

\[
\sigma(w) := (w \gg 1) + (w \gg (b - 1))
\]

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 $\sigma(\cdot)$ is an orthomorphism, then so is $\sigma^2(\cdot)$.

**Proof:** First, note that $\sigma^2(\cdot)$ is clearly a bijective linear map. By definition $p(x) = x + \sigma(x)$ is a bijection, and so is its iterate: $p^2(x) = (x + \sigma(x)) + \sigma(x + \sigma(x)) = x + \sigma(x) + \sigma(x) + \sigma^2(x) = x + \sigma^2(x)$. Hence $\sigma^2(\cdot)$ is an orthomorphism. \(\Box\)
Therefore, $\sigma^i(\cdot), \sigma^j(\cdot), \sigma^k(\cdot)$ 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 $i \neq j$, $\sigma^i(x) \rightarrow \sigma^j(x)$ is an orthomorphism or the identity, and $x \rightarrow \sigma^i(x) + \sigma^j(x)$ 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, SBY+15].

3.2 The Almost-MDS Matrix

We use the same type of almost-MDS matrices as in QARMA. Let $R_4$ 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

$$\text{circ}(0, \rho^a, \rho^b, \rho^c) = \begin{pmatrix}
0 & \rho^a & \rho^b & \rho^c \\
\rho^c & 0 & \rho^a & \rho^b \\
\rho^b & \rho^c & 0 & \rho^a \\
\rho^a & \rho^b & \rho^c & 0
\end{pmatrix}. \tag{4}$$

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

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

which are all involutory. Following the QARMA paper, they are grouped into classes depending on their transition patterns: Class I includes $M_0$ and $M_{4,1}$; and $M_{4,2}$ is a Class II matrix. Their transition patterns are displayed in Fig. 8 on page 33 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 choose instead Class I matrix $M_{4,1}$ and we validate our choice by evaluating the resistance of the cipher against several attacks (cf. Section 4).

3.3 The StateShuffle

The MIDORI StateShuffle [BBI+15] 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 [BBI+15] are satisfied.

Let us consider the square function of $\tau$,

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

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 MIDORI. For the limited number of rounds that we consider, the chosen \( r \), 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 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.

#### Single Layer Blocks

The MANTIS/QARMA 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 maximising the active cells in MANTIS first, and then in MANTIS6. The resulting shuffle \( h_0 = h \) is

\[
    h_0 : [6, 5, 14, 15, 0, 1, 2, 3, 7, 12, 13, 4, 8, 9, 10, 11],
\]

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

QARMA and QARMAv2 have very different tweak schedules. It is not obvious whether QARMA’s \( h \) is optimal also for QARMAv2, and we may want to pick a better shuffle if not.\(^1\) 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 QARMA, QARMAv2’s longer tweak translates in a higher number of variables and relations in MILP models. As a consequence, these increasingly take a much longer time to solve than QARMA’s models. This means that 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

\(^1\)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 few, longer cycles.
We show how an active single cell evolves on page 8.

A closer 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 of a state affects the cells of the state after two rounds. Let \( t_n \) be the round tweak added at round \( n \). Then \( t_{n+2} = \varphi(t_n) \) is added at round \( n \). If cell \( c_i \) affected in round \( n \) by \( t_n \), we want to minimize the likelihood that any of the cells round \( n + 2 \) that are are affected by \( c_i \) will be affected by \( t_{n+2} \) as well, i.e. we want to minimize the likelihood of self-cancellations in characteristics. 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 \( \varphi^{-1} \) may behave like a randomly chosen permutation. The tweak schedule will progress beyond the central construction unmodified, so \( \varphi^{-1} \) will become the “almost optimal” map and \( \varphi \) will become the “mostly randomly-behaving” one. As we shall see, this strategy seems to be working.

In the first column of Fig. 7 on page 32 we show how an active single cell evolves through two forward rounds, for all sixteen cells in a layer. If the tweak shuffle \( \varphi \) maps a first (active) cell to a (second) cell that is not affected by the first cell after two rounds, a self-cancellation cannot occur. In the second column of the figure we compare this to how the \( \tau \) operation transforms entire rows and columns. Beside the remark already made in Section 3.4 on the preceding page that \( \tau^2 \) maps each row to a column (and vice versa) we note also that: The square of the composite map \( M \circ \tau \) lets each cell of a given row act only on 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. This unaffected column is the very one to which the cells of the row are mapped under \( \tau^2 \).

This suggests 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 self-cancellation property still holds. This is too large a search space. If we compose \( \tau^2 \) with a permutation of the rows, we need to look at just 24 permutations. Their maximal order is 8 and only six of them decompose as two cycles of length 8. The latter are our starting point: we first compute their active cell counts for the first few values of \( r \), to choose a best candidate \( \tilde{\tau} \). Then, we look at the order 16 cycles obtained from \( \tilde{\tau} \) by means of an additional cell swap within a single column to see if they have higher trail weights. The initial six shuffles are:

\[
\tau_{d1} : [2, 10, 14, 6, 0, 8, 12, 4, 3, 11, 15, 7, 1, 9, 13, 5], \\
\tau_{d2} : [1, 9, 13, 5, 0, 8, 12, 4, 2, 10, 14, 6, 3, 11, 15, 7], \\
\tau_{d3} : [2, 10, 14, 6, 1, 9, 13, 5, 0, 8, 12, 4, 3, 11, 15, 7], \\
\tau_{d4} : [3, 1, 11, 15, 7, 2, 10, 14, 6, 0, 8, 12, 4, 1, 9, 13, 5], \\
\tau_{d5} : [3, 1, 11, 15, 7, 2, 10, 14, 6, 0, 8, 12, 4, 1, 9, 13, 5], \\
\tau_{d6} : [1, 9, 13, 5, 2, 10, 14, 6, 3, 11, 15, 7, 0, 8, 12, 4].
\]

We determine the minimal active cell counts for these shuffles and their inverses as well, which is equivalent to inverting the order of application of \( \varphi \) and \( \varphi^{-1} \). The results are reported in Table 7 on page 36. (Note that sometimes for a given series of full-cipher runs with a fixed \( \varphi \), the minimum active cell count is not a monotonic function of \( r \). The reason for this is explained in Appendix A.2 on page 35.) We then choose \( \tilde{\tau} = \tau_{d4} \). Finally, amongst the order 16 shuffles obtained by applying a cell swap to \( \tau_{d4} \), we select the one
with the best trail weights in models for the full-cipher with \( r = 5 \), then 6. This shuffle is
\[
\tau_f : \begin{bmatrix} 1, 10, 14, 6, & 2, 9, 13, 5, & 0, 8, 12, 4, & 3, 11, 15, 7 \end{bmatrix}, \tag{7}
\]
which is obtained from \( \tau_{4d} \) by swapping the topmost two cells in the first column.

We thus have a map which is probably (close to) optimal for our tweak schedule \( \varphi \), considering only one of \( T_0, T_1 \), and ignoring the other. In the design of \( \text{QARMAv2} \), we update \( T_1 \) with \( \varphi \) and \( T_0 \) with \( \varphi^{-1} \). If we wanted also \( \varphi^{-1} \) to be as optimal as possible, we would need both \( \varphi \) and \( \varphi^{-1} \) to be obtained by composing \( \tau^2 \) with a permutation of the rows. But, the only map of this type is \( \tau^2 \), which we know to be highly sub-optimal (cf. Table 7 on page 36). However, it turns out that having \( \varphi \) and its inverse each operate “optimally” on half of the schedule for each of the two tweak blocks leads to the best shuffles we know for the whole cipher.

**Two-Layer Blocks.** The chosen function is derived from \( \tau^2 \) as well. We search it among the shuffles constructed as two stacked copies of \( \tau_f \), that act on each layer in the same way, followed by some swaps of cells which are in the same positions in the two layers, provided the order of the resulting shuffle is maximal, i.e. 32. Our experiments show that swapping about a half of the cells leads to better shuffles, but we must swap only 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:
\[
\tau_F : \begin{bmatrix} 1, 10, 14, 22, & 18, 25, 29, 21, & 0, 8, 12, 4, & 19, 27, 31, 23, \\ 17, 26, 30, 6, & 2, 9, 13, 5, & 16, 24, 28, 20, & 3, 11, 15, 7 \end{bmatrix}. \tag{8}
\]
Summarising, we have \( \tau_F \) for \( \ell = 1 \), and \( \tau_F \) for \( \ell = 2 \).

**Full Diffusion Properties.**

**Theorem 1.** In \( \text{QARMAv2} \) 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 \( \text{QARMA} \). The second is proved in Appendix D on page 37.

**Two-Layer Blocks and \( \text{QARMA} \) Version 1 (i.e. The Old in the New).**

**Remark 1.** The \( m = 4, \ell = 2 \) construction is just a variant of \( \text{QARMA} \)'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 operate on these composite 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 \( \text{QARMA-128} \). However, it allows us to count the active nibbles instead of the active bytes, getting better security margin estimates. Also, many security arguments can still be carried over from the single layer version and \( \text{QARMA} \).

**Single Block Tweaks.** The single block length tweak case is handled by setting \( T_1 = \varphi(T_0) \), where \( T_0 = T \). By doing this we “tie” 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, such as forcing the two round tweaks at the sides of the central construction to be equal, or imposing that the first two round tweaks are equal, and other combinations. Looking at
the active cell counts, the chosen approach seems to give the best results, cf. Tables 9 and 10 on page 37.

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. The rationale for this is that in deployed systems only one fully unrolled or pipelined implementation will be provided (even if multiply instantiated). This said, it is a valid option to reduce the number of rounds by 2 for the single-block tweak case.

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 π and e.

The 64 most significant bits of the round constant \( RC_2 \) (the first non-zero round constant) are given by \( 0x243F6A8885A308D3ULL \), 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. 4.

```c
uint64_t PSI(uint64_t in) // Galois LFSR - ticked 23 (13+10) times.
{
    uint64_t spill, tmp;
    spill = in >> 51;
    tmp = (in << 13) ^ (spill << 50) ^ (spill << 33) ^ (spill << 19) ^ spill;
    spill = tmp >> 54;
    tmp = (tmp << 10) ^ (spill << 50) ^ (spill << 33) ^ (spill << 19) ^ spill;
    return tmp;
}
```

Figure 4: Code for the round constant generating function PSI (\( \Psi \))

3.6 The S-Box

For the general-purpose version of QARMAv2, with respect to QARMA we pick a different S-Box, namely

\[
\mathcal{S} = \begin{bmatrix} 5 & E & 9 & 1 & 8 & A & C & B & 7 & 6 & D & 4 & 2 & 0 & F & 3 \end{bmatrix},
\]

with inverse

\[
\mathcal{\overline{S}} = \begin{bmatrix} D & 3 & C & F & B & 0 & 9 & 8 & 4 & 2 & 5 & 7 & 6 & A & 1 & E \end{bmatrix}.
\]

The reason we changed S-Box with respect to QARMA is that if we use \( \sigma_1 \) with a Class I diffusion matrix, then there exists a nontrivial nonlinear invariant for the unkeyed round function of the cipher [Bey23], which is invariant under translation by a set of \( 2^{32} \) constants. This invariant does not seem to threaten the security margins of the design and is only
likely to make some reduced round integral attacks more effective. However, we do not want to finalise a design with such an invariant.

Using $\sigma_0$ would have prevented such nonlinear invariants, but the S-Box has less optimal cryptographic properties – while several designs intentionally adopt weaker, but faster or smaller S-Boxes, recall that the main point of the QARMA and QARMAv2 designs is their conservativeness. Also, $\sigma_0$ gives Boomerang attacks one extra round of reach. An alternative could have been to keep $\sigma_1$ and reverting to a Class II matrix – however the active cell counts in related-tweak differential characteristics would have been significantly reduced, more so than in QARMA, because of the longer tweaks. The S-Box $\sigma_2$ would have worked as well, but it is also larger and has a longer latency.

The S-Box $\mathcal{P}$ 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 exactly 15 such differentials with probability $1/4$; (ii) The maximal absolute bias of a linear approximation is $1/4$ and there are exactly 30 linear approximations with absolute bias $1/4$; (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.

This S-Box is not involutory and has order 4. Furthermore, it leads to slightly lighter and faster implementations than $\sigma_1$ (approximately 2% better latency and area). The search for $\mathcal{P}$ was programmed to satisfy additional conditions for both the S-Box and its inverse, namely: (v) For each output bit of the S-Box and of its inverse either the SOP (sum-of-products) or the NOT-SOP (corresponding to a product-of-sums) have expressions of weight at most 3 and degree at most 3; and (vi) The corresponding sums of weights and degrees of all four bits are bounded by 8 and 12, respectively. These constraints led to 4 S-Boxes together with their inverses, which are all affine equivalent to each other. The lightweightness of the S-Boxes is verified using PEIGEN [BGLS19] and synthesis of the full QARMAv2, whereas the cryptographic properties have been double checked using sboxU [JPB23].

The S-Box $\mathcal{P}$ is not only of order 4 but it is the product of four cycles of length 4. The choice may seem unusual, but there is a rationale behind it. First of all, we want to reduce the maximum number of invariant sets or non-linear invariants that the S-box- and linear layer have in common. Following [Bey18], the number of invariant of the S-Box is related to the number of cycles of the latter, and therefore it makes sense to minimise this number. However, with a restriction of having only 3 cycles or less in place, we found no S-Boxes that satisfied our strict constraints on the SOP and NOT-SOP: we tried for instance with cycle structures $(5, 5, 6)$, $(8, 8)$ and $(16)$. With a $(4, 4, 4, 4)$ cycle structure, however, good S-Boxes were found. Furthermore, with a computer search, we verified that $\mathcal{P}$ (as well as $\sigma_0$) has no non-linear invariants, no closed-loop invariants over two rounds, and no closed-loop quadratic invariants up to four rounds [WYWP18, WRP20].

### 3.7 Alternative Constructions

In passing, we note that we experimented also with a symmetric tweak schedule, where the backward rounds reverse the tweak schedule. 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. In all these constructions, the number of active cells turned out to be significantly lower for all the parameter choices we tested.

Regarding the two-layer version of the cipher, we looked at variant where only one row is swapped instead of two, and the swap is performed every round or every three rounds. All these variants turned out to be weaker, at least with the considered tweak schedules.
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

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. 5a. The boomerang connectivity table (BCT) \([\text{CHP}^{+}18]\) in Fig. 5b 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 behaviour is summarized in Fig. 8c.

We bound the maximum probability of differential characteristics for QARMAv2 with the help of a MILP model of the cell-wise differential behaviour. For a detailed description of the modelling approach, we refer to Appendix A.1 on page 32. 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-key setting, while the linear bounds are identical to single-key, single-tweak differential bounds.

A somewhat surprising remark is that, if we compare, say, a \( 2r + 1 \) rounds half-cipher to a \( 2r + 2 \) rounds full-cipher for “good” shuffles they seem to have a similar number of active cells (see also Table 7 on page 36). The reflector construction proves itself useful because for small \( r \) the weights seem to ramp up much faster than in a normal iterative construction, so it provides a clear advantage for the smaller cases.

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. 10 on page 41. While this distinguisher has 26 active cells, which is 2 more than the optimum of
in both directions to identify jointly active S-boxes and differentiate the key candidates. For this attack, we generate two structures of size $2^{25}$ 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. Even if we assume that for this truncated differential pattern an optimal characteristic 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 $2^{25}$ 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 $2^{218}$ operations, we instead iterate over the $2^{15}$ 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 $2^{5.9}$ candidates for the five active cells in 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 MixColumns($K_1$). By independently verifying the guess in each column, we can keep the complexity below $2^{218}$. 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 it 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 $2^6$ 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 $\leq 2^{256}$.

**Boomerang attacks.** We bound the probability of characteristics for sandwich distinguuishers with the help of a MILP model. We consider sandwich distinguishers where the cipher is decomposed into three parts $E_1 \circ E_m \circ E_0$, as illustrated in Fig. 6. 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]:

| Table 2: Minimum number of active S-boxes for related-key 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. |
|-----------------|-----------------|
| $r = 3$ | $4$ | $5$ | $6$ | $7$ | $8$ | $9$ | $10$ | $11$ | $12$ | $\ell$ | Rounds | $2$ | $3$ | $4$ | $5$ | $6$ |
| 1 | Diff. | $2$ | $4$ | $8$ | $12$ | $16$ | $22$ | $24$ | $27$ | $32$ | $36$ | $5$ | $12$ | $24$ | $32$ | $41$ |
| Linear | $16$ | $23$ | $30$ | $35$ | $38$ | $41$ | $50$ | $57$ | $62$ | $67$ | $5$ | $32$ | $50$ | $64$ | $72$ |
| 2 | Diff. | $2$ | $6$ | $11$ | $17$ | $26$ | $34$ | $44$ | $50$ | $55$ | $59$ | $5$ | $16$ | $32$ | $52$ | $61$ |
| Linear | $16$ | $25$ | $36$ | $48$ | $58$ | $68$ | $72$ | $80$ | $88$ | $100$ | $24$ | $44$ | $56$ | $80$ | $96$ |
The tweakable block cipher family QARMAv2

Figure 6: Boomerang/sandwich distinguisher.

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 ) ( 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>6</td>
<td>15</td>
<td>21</td>
<td>32</td>
<td>44</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>7</td>
<td>15</td>
<td>21</td>
<td>32</td>
<td>44</td>
<td>60</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>

S-boxes. The overall probability \( p^2 \cdot r \cdot q^2 \) 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 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. 12 to 14 on pages 43–45 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 \( p^2 \cdot q^2 \) using 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

**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. 5c. 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 $6 \frac{1}{2}$ 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 $10 \frac{3}{2}$ rounds of the full-cipher ($r = 5$).

**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 [BCFG+21, 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 QARMA against impossible-differential and zero-correlation cryptanalysis was analysed using the characteristic matrix technique from [SLG+16]. For QARMAv2 we use instead 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 Figure 11. 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 exploits low algebraic degrees of a cryptographic primitive. The attackers typically assemble a collection of $N$ chosen plaintexts and generate the corresponding ciphertexts using the target block cipher. By observing the XOR 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.

The Division property is a generalization of the integral property that was proposed by Todo at Eurocrypt 2015 [Tod15]. 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 provides a more precise analysis at the bit level.

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

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

**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 $x = (x_0, x_1, \ldots, x_{n-1})$ 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 $2^{\text{wt}(k)}$, where $\text{wt}(k)$ 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 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 6 forward rounds of the cipher. The best division properties are obtained from an input with 63 active bits and a single bit inactive, which can be in any position $4i + 1$ within the state, where $i \in \{0, 1, \ldots, 15\}$. From each such input division property, we get all 64 bits of the output state as balanced bits. We also identified another set of integral characteristic for 6 rounds, with 63 active bits and a single inactive bit positioned anywhere except at positions of the form $4i + 1$, where $i \in \{0, 1, \ldots, 15\}$. In this particular scenario, the 16 output bits at positions $4i + 1$ are balanced, and the other bits are not determined.

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 QARMA$^\text{v2}$ 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 QARMA [ZD16] can be adapted to QARMA$^\text{v2}$ 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 QARMA has data complexity of $2^{53}$ chosen plaintexts and time complexity of $2^{116}$. Since there is a large gap between the number of attacked rounds and the parameters given in Table 1 on page 16, we posit that such attacks are not a threat to the security of QARMA$^\text{v2}$.

4.6 Invariant Subspace Cryptanalysis and Non-Linear Invariants

In order to determine whether (untweaked) QARMA$^\text{v2}$-64 or QARMA$^\text{v2}$-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 the linear layer and possibly the S-Box layer. We added support for QARMA$^\text{v2}$ to the software and we could not find any small dimensional invariant subspace of the un-tweaked ciphers: for forward function of QARMA$^\text{v2}$-64 does not have non-trivial invariant subspaces as soon as we consider three full rounds, and QARMA$^\text{v2}$-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.

Finally, let us consider symmetric non-linear invariants, i.e. invariants of the form $\sum_i g_1(c_i) = \sum_i g_2((S \circ M \circ \tau)(\mathcal{X}_i))$ 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 for up to two rounds in general and up to four rounds for quadratic invariants.

4.7 Algebraic Attacks

In light of Remark 1, the exact same counts made for QARMA in [Ava17, Section 4.2] apply to QARMA$^\text{v2}$, not only for the single layer version but also for the two-layer construction. Hence, with the parameters discussed in Section 4.9 on the following page, all versions of QARMA$^\text{v2}$ should be immune from algebraic attacks in the sense of [CP02].

4.8 Security Implications of the Central Construction

The arguments in [Ava17] hold in a stronger form. For each addition of $K_i$ on the forward path, $\sigma(K_i)$ is added in various places on the backward path. Hence, any reflection-like relation encompassing the central construction $C$ and any amount of rounds on each of the two sides would be disrupted (i.e. its likelihood reduced) in the same way as described in [Ava17], where however only relations that start right at the sides of $C$ are considered. 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, \texttt{QARMAv2-64-128}, and three variants with 128 bit blocks and varying key sizes, \texttt{QARMAv2-128-128}, \texttt{QARMAv2-128-192}, \texttt{QARMAv2-128-256}. 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 summarised 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 \textit{first} attack, and if that were not possible, this would make any further attack also not possible.

4.10 Comparison to \texttt{QARMA}

\texttt{QARMAv2}'s parameter \(r\) seems, superficially, larger than \texttt{QARMA}'s at the same security level. When we will later compare the latencies directly, indeed we we lose a bit to \texttt{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 \texttt{QARMA} this was the product of data and time.

With the data limits from Table 1 on page 16, the 128-bit security level for \texttt{QARMA-64} represented a time \(\Omega(2^{72})\), and the two security levels of 192 and 256 bits for \texttt{QARMA-128} translate to time \(\Omega(2^{112})\) and \(\Omega(2^{176})\). This is due to the fact that \texttt{QARMA} i a three-round EM construction \cite{EM91} (which is, because of the simplicity of the second round that comprises only the central construction, barely more than a single round design, or FX construction \cite{Rog96}) whereas \texttt{QARMAv2} is a key alternating design.

This said, the implementor may reduce the parameter \(r\) by two for \texttt{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 \cite{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: \texttt{AES-128|192|256}, \texttt{QARMA(v1)-\{64|128\}}, \texttt{PRINCE}, \texttt{Orthros}, \texttt{SPEEDY-\{6|7\}}, \texttt{MIDORI-\{64|128\}}, \texttt{PRESENT-\{80|128\}}, and \texttt{SKINNY-\{64|128\}}. We also compare our design with the \texttt{ASCON-p\textsuperscript{12}} permutation used in a single key Even-Mansour mode. We skip \texttt{PRINCEv2} because its latency and area are almost identical to the original \texttt{PRINCE}.

In order to guarantee a fair comparison to \texttt{SKINNY} \cite{BJK+16} we use the more recent parameters that are used in Romulus for the NIST Lightweight Cryptography Competition \cite{IKMP19a, IKMP19b}. We observe that in related-tweakkey characteristics for \texttt{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 \texttt{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 \texttt{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 standardisation 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 synthesise 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. Optimising for area is computationally heavy stage outputs a circuit with near optimal area or power consumption. We use this as the datapoint for the area-optimised circuit. This step also outputs the total critical path of the circuit.
4. Timing simulations have then been performed on the synthesised 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-optimised 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 optimise overall implementations involving 4-bit S-Boxes better or at most only in a slightly suboptimal way than using ad-hoc implementations (either hand-optimised 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 authors and the table-based implementation 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 the product of Delay and Power, divided by block size, normalised 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 synthesised with a different mix of gates than one with a 4-bit S-Box, a fact that is ignored by our FoM.

Regarding the 90nm results, we note that the synthesis step traverses a wide design space before producing a circuit that optimises 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.
At the 128-bit block level, the advantage is even more striking. At all security levels, we tested. For instance, at the 64-bit block level, it is almost always faster than tweaked.

QARMAv2

<table>
<thead>
<tr>
<th>Cipher</th>
<th>Rounds</th>
<th>Area $\mu$m$^2$</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>11839</td>
<td>1728</td>
<td>26.63</td>
<td>100</td>
</tr>
<tr>
<td>AES-128</td>
<td>12</td>
<td>26098</td>
<td>137397</td>
<td>2661</td>
<td>34.96</td>
<td>157</td>
</tr>
<tr>
<td>AES-256</td>
<td>14</td>
<td>31744</td>
<td>161459</td>
<td>2319</td>
<td>47.64</td>
<td>240</td>
</tr>
<tr>
<td>PRESENT-50</td>
<td>31</td>
<td>3800</td>
<td>19370</td>
<td>1001</td>
<td>2.87</td>
<td>11.2</td>
</tr>
<tr>
<td>PRESENT-128</td>
<td>31</td>
<td>3800</td>
<td>20035</td>
<td>960</td>
<td>2.67</td>
<td>12.3</td>
</tr>
<tr>
<td>MIDori-64</td>
<td>16</td>
<td>5256</td>
<td>9844</td>
<td>629</td>
<td>9.86</td>
<td>2.61</td>
</tr>
<tr>
<td>MIDori-128</td>
<td>20</td>
<td>5102</td>
<td>25952</td>
<td>851</td>
<td>3.35</td>
<td>6.20</td>
</tr>
<tr>
<td>PRINCE</td>
<td>12</td>
<td>1664</td>
<td>8456</td>
<td>536</td>
<td>1.67</td>
<td>1.56</td>
</tr>
<tr>
<td>Ortheos</td>
<td>12</td>
<td>5766</td>
<td>29229</td>
<td>486</td>
<td>2.33</td>
<td>2.46</td>
</tr>
<tr>
<td>SPEEDY-6</td>
<td>6</td>
<td>5200</td>
<td>26551</td>
<td>545</td>
<td>2.06</td>
<td>1.63</td>
</tr>
<tr>
<td>SPEEDY-7</td>
<td>7</td>
<td>6610</td>
<td>33618</td>
<td>659</td>
<td>3.18</td>
<td>3.02</td>
</tr>
<tr>
<td>ASCMA-p$^2$</td>
<td>12</td>
<td>5562</td>
<td>34745</td>
<td>566</td>
<td>2.07</td>
<td>1.63</td>
</tr>
<tr>
<td>SKINNY-128-192</td>
<td>32</td>
<td>4069</td>
<td>20903</td>
<td>1953</td>
<td>5.03</td>
<td>42.6</td>
</tr>
<tr>
<td>SKINNY-128-256</td>
<td>40</td>
<td>18250</td>
<td>94200</td>
<td>3662</td>
<td>48.89</td>
<td>343</td>
</tr>
</tbody>
</table>

Our results show that QARMAv2 offers strong advantages with respect to all other ciphers we tested. 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 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,
### Table 6: Comparative Evaluation Metrics for the TSMC 5nm Process and Library

<table>
<thead>
<tr>
<th>Cipher</th>
<th>Rounds</th>
<th>Area µm²</th>
<th>Delay ps</th>
<th>FoM % AES</th>
<th>Area µm²</th>
<th>Delay ps</th>
<th>FoM % AES</th>
</tr>
</thead>
<tbody>
<tr>
<td>AES-128</td>
<td>2304.1</td>
<td>28873</td>
<td>3064</td>
<td>100</td>
<td>4520.6</td>
<td>56648</td>
<td>1791</td>
</tr>
<tr>
<td>AES-192</td>
<td>2635.4</td>
<td>33925</td>
<td>3686</td>
<td>138</td>
<td>5023.6</td>
<td>62952</td>
<td>2153</td>
</tr>
<tr>
<td>AES-256</td>
<td>3338.7</td>
<td>40685</td>
<td>4290</td>
<td>197</td>
<td>6191.5</td>
<td>77587</td>
<td>2513</td>
</tr>
<tr>
<td>PRESENT-80</td>
<td>812.0</td>
<td>10175</td>
<td>1836</td>
<td>42.2</td>
<td>1815.7</td>
<td>22772</td>
<td>953</td>
</tr>
<tr>
<td>MIDORI-64</td>
<td>443.5</td>
<td>5557</td>
<td>921</td>
<td>10.6</td>
<td>761.8</td>
<td>9546</td>
<td>678</td>
</tr>
<tr>
<td>MIDORI-128</td>
<td>1085.1</td>
<td>13597</td>
<td>1156</td>
<td>17.8</td>
<td>1954.5</td>
<td>24492</td>
<td>840</td>
</tr>
<tr>
<td>PRINCE</td>
<td>334.6</td>
<td>4193</td>
<td>710</td>
<td>6.74</td>
<td>672.1</td>
<td>8422</td>
<td>534</td>
</tr>
<tr>
<td>QARMAv2-64</td>
<td>12</td>
<td>394.7</td>
<td>4846</td>
<td>8.14</td>
<td>707.0</td>
<td>8860</td>
<td>525</td>
</tr>
<tr>
<td>QARMAv2/PAC</td>
<td>8</td>
<td>416.2</td>
<td>5215</td>
<td>9.16</td>
<td>735.6</td>
<td>9218</td>
<td>548</td>
</tr>
<tr>
<td>QARMA-64</td>
<td>16</td>
<td>551.7</td>
<td>6913</td>
<td>10.3</td>
<td>996.6</td>
<td>12499</td>
<td>731</td>
</tr>
<tr>
<td>QARMA-128</td>
<td>24</td>
<td>1635.6</td>
<td>20496</td>
<td>15.6</td>
<td>3878.3</td>
<td>38575</td>
<td>1091</td>
</tr>
<tr>
<td>QARMAv2-64</td>
<td>14</td>
<td>451.6</td>
<td>5659</td>
<td>9.86</td>
<td>852.2</td>
<td>10679</td>
<td>573</td>
</tr>
<tr>
<td>QARMAv2/PAC</td>
<td>8</td>
<td>384.6</td>
<td>3817</td>
<td>4.85</td>
<td>578.6</td>
<td>7251</td>
<td>496</td>
</tr>
<tr>
<td>QARMAv2-64</td>
<td>20</td>
<td>663.0</td>
<td>8308</td>
<td>12.5</td>
<td>1256.9</td>
<td>15750</td>
<td>917</td>
</tr>
<tr>
<td>QARMAv2-128</td>
<td>24</td>
<td>1591.9</td>
<td>19948</td>
<td>15.3</td>
<td>3007.6</td>
<td>37688</td>
<td>1111</td>
</tr>
<tr>
<td>QARMAv2-192</td>
<td>28</td>
<td>1858.1</td>
<td>23284</td>
<td>17.9</td>
<td>3529.3</td>
<td>44227</td>
<td>1297</td>
</tr>
<tr>
<td>QARMAv2-256</td>
<td>32</td>
<td>2120.4</td>
<td>26646</td>
<td>20.4</td>
<td>4048.1</td>
<td>50728</td>
<td>1480</td>
</tr>
</tbody>
</table>

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 and therefore to perform ideally in 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. The goal for the QARMA and QARMAv2 families is to perform optimally in fully unrolled implementations, not to provide the smallest round-based implementations. This is very different from the SKINNY approach and therefore the comparison should not be taken as an absolute assessment of SKINNY's merits.

### 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, it should raise the bar for attackers.

Like QARMA, QARMAv2 is aimed at low-latency fully unrolled implementations. Despite being suitable for general purpose use, it is optimised 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. It is also a conservative design: we could perhaps have reduced the number of rounds, or used an even lighter S-Box, but we preferred to keep healthy security margins while still addressing the needs of the intended applications. Our HW implementations suggest that our design goals have been achieved.

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 prompts 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 its complexity (intended as minimal depth and area): the longer the cycles, the more difficult it seems to find lightweight S-Boxes, provided that the usual cryptographic properties are satisfied. Also, can better tweak shuffles or 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? The last, and perhaps more important question, is how the design of lightweight cryptographic primitives can take advantage of the features of ever shrinking lithography processes.

References


<table>
<thead>
<tr>
<th>Reference</th>
<th>Title and Authors</th>
<th>Conference</th>
<th>Volume</th>
<th>Pages</th>
<th>Publisher</th>
<th>Year</th>
</tr>
</thead>
</table>

A MILP Modelling

A.1 Modelling 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 on the next page we show the admissible transitions of Class I and II matrices. We now set to turn them into MILP models.
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

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

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

\[
\begin{align*}
y_i + \sum_{j \neq i} x_j & \geq 2y_i, \quad \text{and} \quad y_i + \sum_{j \neq i} x_j \geq 2x_{j'} \quad \text{for all} \quad j' \neq i. \\
x_i + \sum_{j \neq i} y_j & \geq 2x_i, \quad \text{and} \quad x_i + \sum_{j \neq i} y_j \geq 2y_{j'} \quad \text{for all} \quad j' \neq i.
\end{align*}
\]

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:

\[
\sum_{i \neq k} y_i \geq x_k \quad \text{and} \quad \sum_{i \neq k} x_i \geq y_k \quad \text{for} \quad 0 \leq k < 4.
\]  

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:

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

A.1.2 Class I Specific Relations

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

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

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:

\[
\sum_i x_i + \sum_i y_i \leq 4 + 4d_i \quad \text{and} \quad \sum_i x_i + \sum_i y_i \geq 6d_i \quad \text{for} \quad 0 \leq i < 4.
\]
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 (14) on the preceding page). 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:

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

$$d'_{ij} \geq y_i + y_j - \sum_{k \neq i, j} y_k - 1, \quad x_i \geq d'_{ij} \quad \text{and} \quad x_j \geq d'_{ij} \quad . \quad (15)$$

### A.1.3 Class II Specific Relations

Relation (13) on the previous page 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

$$\left( \sum_{i \in I} x_i + \sum_{i \in I} (1 - x_i) \right) + \left( \sum_{j \in J} y_j + \sum_{j \in J} (1 - y_j) \right) \leq 7 \quad . \quad (16)$$

The behaviour of a diffusion matrix is normally not modelled exclusively by this approach, as the resulting model would be too large and probably too slow to optimize. Usually, the XOR model and the branch number are modelled first, then any other non-occurring 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 MANTIS and QARMA papers the matrix was defined by relations Relations (9) and (11) to (13) on the preceding page – in particular the XOR model is not used. This set of relations becomes a model for Class II matrices if in Relation (13) on the previous page we replace the condition $i \neq j$ with $|i - j| = 2$. These two models can be slightly accelerated by adding Relation (15).

Our starting point consisted of the XOR Relations (10) and the branch number Condition (9), removing inadmissible transitions as 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 (9), (10) and (14) on the preceding page. A larger, but sometimes significantly faster model when using Gurobi’s default settings, uses Relations (9), (11), (14) and (15) on pages 33–34. 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 MILP problems.

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 $\phi$ replaced by its inverse $\phi^{-1}$ (and the roles of the two tweak blocks exchanged). So if one merges the counts for a given $\phi$ with, say, even $r$ with the counts for $\phi^{-1}$ 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 7 on the next page we tabulate the weights of optimal linear and related-tweak differential trails for both half-cipher and full-cipher QARMAv2-64 for various values of $r$ and various choices of the tweak update function. Table 8 is the corresponding table for QARMAv2-128.

C A Remark on Minimal Weight Linear Characteristics

In Fig. 9 we see an example of a minimal weight iterative cell-based linear trail for single-layer QARMAv2. Several similar trails exist, all with $1 + 3 + 9 + 9 + 9 + 3 = 34$ 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
Table 7: Active cell counts for tweak-related differential characteristics and linear trails ($\ell = 1$, two independent tweak blocks)

<table>
<thead>
<tr>
<th>$r$</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
<th>12</th>
</tr>
</thead>
<tbody>
<tr>
<td>rounds</td>
<td>$3 \frac{5}{2}$</td>
<td>$4 \frac{1}{2}$</td>
<td>$5 \frac{5}{2}$</td>
<td>$6 \frac{1}{2}$</td>
<td>$7 \frac{7}{2}$</td>
<td>$8 \frac{7}{2}$</td>
<td>$9 \frac{9}{2}$</td>
<td>$10 \frac{11}{2}$</td>
<td>$11 \frac{11}{2}$</td>
<td>$12 \frac{12}{2}$</td>
</tr>
<tr>
<td>$\tau_1$</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>3</td>
<td>3</td>
<td>4</td>
<td>4</td>
<td>5</td>
<td>5</td>
<td>6</td>
</tr>
<tr>
<td>$\tau_2$</td>
<td>2</td>
<td>4</td>
<td>7</td>
<td>7</td>
<td>12</td>
<td>16</td>
<td>16</td>
<td>20</td>
<td>20</td>
<td>24</td>
</tr>
<tr>
<td>$\tau_3$</td>
<td>2</td>
<td>6</td>
<td>8</td>
<td>10</td>
<td>12</td>
<td>12</td>
<td>14</td>
<td>16</td>
<td>18</td>
<td>18</td>
</tr>
</tbody>
</table>

Table 8: Active cell counts for tweak-related differential characteristics and linear trails ($\ell = 2$, two independent tweak blocks)

<table>
<thead>
<tr>
<th>$r$</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
<th>12</th>
</tr>
</thead>
<tbody>
<tr>
<td>rounds</td>
<td>$3 \frac{5}{2}$</td>
<td>$4 \frac{1}{2}$</td>
<td>$5 \frac{5}{2}$</td>
<td>$6 \frac{1}{2}$</td>
<td>$7 \frac{7}{2}$</td>
<td>$8 \frac{7}{2}$</td>
<td>$9 \frac{9}{2}$</td>
<td>$10 \frac{11}{2}$</td>
<td>$11 \frac{11}{2}$</td>
<td>$12 \frac{12}{2}$</td>
</tr>
<tr>
<td>$\tau_1$</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>3</td>
<td>3</td>
<td>4</td>
<td>4</td>
<td>5</td>
<td>5</td>
<td>6</td>
</tr>
<tr>
<td>$\tau_2$</td>
<td>2</td>
<td>4</td>
<td>7</td>
<td>7</td>
<td>12</td>
<td>16</td>
<td>16</td>
<td>20</td>
<td>20</td>
<td>24</td>
</tr>
<tr>
<td>$\tau_3$</td>
<td>2</td>
<td>6</td>
<td>8</td>
<td>10</td>
<td>12</td>
<td>12</td>
<td>14</td>
<td>16</td>
<td>18</td>
<td>18</td>
</tr>
</tbody>
</table>

Linear 16 23 30 35 38 41 50 57 62 67 5 32 50 64 72
Table 9: Active cell counts for tweak-related differential characteristics and linear trails ($\ell = 1$, $t = 1$ with tweak blocks tied near center)

<table>
<thead>
<tr>
<th></th>
<th>Half Cipher</th>
<th>Full Cipher</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>$r =$</td>
<td>3</td>
</tr>
<tr>
<td></td>
<td>rounds</td>
<td>$3\frac{1}{2}$</td>
</tr>
<tr>
<td></td>
<td>$\tau$</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>$\tau^2$</td>
<td>5</td>
</tr>
<tr>
<td></td>
<td>$h$</td>
<td>5</td>
</tr>
<tr>
<td></td>
<td>$h_4$</td>
<td>5</td>
</tr>
</tbody>
</table>

Table 10: Active cell counts for tweak-related differential characteristics and linear trails ($\ell = 2$, $t = 1$ with tweak blocks tied near center)

<table>
<thead>
<tr>
<th></th>
<th>Half Cipher</th>
<th>Full Cipher</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>$r =$</td>
<td>3</td>
</tr>
<tr>
<td></td>
<td>rounds</td>
<td>$3\frac{1}{2}$</td>
</tr>
<tr>
<td></td>
<td>$\tau$</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>$\tau^2$</td>
<td>5</td>
</tr>
<tr>
<td></td>
<td>$h$</td>
<td>5</td>
</tr>
<tr>
<td></td>
<td>$h_4$</td>
<td>5</td>
</tr>
<tr>
<td></td>
<td>$\tau_f$</td>
<td>5</td>
</tr>
</tbody>
</table>

based on a a Class I Almost MDS matrix over a ring $R_m$ and the MIDORI StateShuffle needs $(\frac{m^2}{4})/((\frac{m}{4})^2) = 1 \approx 10\%$ 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.

D 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 32 trails displayed at the end of this Section. Now, these properties hold with all S-Boxes $\sigma_0$, $\sigma_1$, $\sigma_2$ and $\pi$ 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.
E Characteristics for Differential, Impossible Differential, and Boomerang Attacks

The following figures provide detailed characteristics for several of the cryptanalytic results of Subsection 4.1. Figure 10 illustrates how a key-recovery attack on \( r = 6 \) rounds of \texttt{QARMA}v2-64 could be designed based on a characteristics following the optimal truncated differential pattern for \( r = 4 \). Figure 11 illustrates the longest impossible differential distinguisher for \texttt{QARMA}v2-64 under a cell-based miss-in-the-middle approach. Figure 12–14 show several potential boomerang distinguishers identified by a cell-wise model for a total of \( 10^{1.2} \) rounds. The more balanced setups yield lower bounds.

F Version for Pointer Authentication and Memory Integrity

For the computation of Pointer Authentication Codes (PACs) and memory integrity tags (the latter following [JLK+23]) we instantiate \texttt{QARMA}v2-64 with \texttt{QARMA}'s S-Box \( \sigma_0 \) [Ava17]. This S-Box is given as

\[
\sigma_0 := \begin{bmatrix} 0 & E & 2 & A & 9 & F & 8 & B & 6 & 4 & 3 & 7 & D & C & 1 & 5 \end{bmatrix}.
\]
Figure 10: Sketch of key recovery attack on QARMAv2 with $r = 6$ based on differential distinguisher for $r = 4$
Figure 11: Longest impossible differential distinguisher for QARMAv2.
Figure 12: Boomerang distinguisher of weight 73 for $6\frac{1}{2} + 4\frac{1}{2}$ rounds of QARMAv2-64, corresponding to a probability bound of $2^{-10 \times 4 \cdot \left(\frac{10}{16}\right)^{13}} \approx 2^{-48.8}$.
Figure 13: Boomerang distinguisher of weight 82 for 5 rounds of QARMAv2-64, corresponding to a probability bound of $2^{-55.1}$. 

The tweakable block cipher family QARMA-v2.
Figure 14: Boomerang distinguisher weight 60 for $9\frac{1}{2} + 1\frac{1}{2}$ rounds of QARMAv2-64, corresponding to a probability bound of $2^{-5 \times 4 \cdot (\frac{10}{10})^{30}} \approx 2^{-40.3}$.
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 not derived using the procedure described in Section 3.5 on page 14, but are all consecutive digits of the hexadecimal expansions of the fractional part of the constant $\pi$.

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 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. 15 and 16. They are “tPMACs”, i.e. tweakable Parallelisable MACs. The first one is for the case where the system does not provides freshness to encryption, and the second one is to be used when freshness is available.

We recall 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 can be 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 allows to use shallower circuits where the encryption operation is not on the critical path.

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 freshness information.

Finally, since the output of the function for pointer authentication needs to be truncated to be used as a PAC, we describe the procedure to generate a $z$-bit tag with $1 \leq z \leq 32$: First, reorder the bits of the output of the tag path as follows:

$$[0, 8, 16, 24, 32, 40, 48, 56], \quad 4 + [0, 8, 16, 24, 32, 40, 48, 56],$$

$$1 + [0, 8, 16, 24, 32, 40, 48, 56], \quad 5 + [0, 8, 16, 24, 32, 40, 48, 56].$$

Then, pick the first $z$ bits in this sequence.

**G  Wider Versions**

Here we describe how to design versions of QARMAv2 with 256- and 512-blocks. 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 first S-Box layer it will have non-linearly affected four bits.
in each cell it affects, and after the second S-Box layer it will have non-linearly affected eight, resp. sixteen bits in each cell it affects.

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 $\text{QARMAv2-128}$, for the same number of rounds, even though the tweak is constrained.

\begin{figure}[h]
\centering
\includegraphics[width=\linewidth]{figure17.png}
\caption{Construction of the eight-bit S-Box (as in QARMA)}
\end{figure}

\begin{figure}[h]
\centering
\includegraphics[width=\linewidth]{figure18.png}
\caption{Construction of the sixteen-bit S-Box}
\end{figure}
H  Test Vectors

H.1  QARMAv2-64

\[ P = 0000000000000000 \]

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

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

\[ C = a49c0a683065cbbc1 \]

H.2  QARMAv2-128

\[ P = 00000000000000000000000000000000 \]

\[ K_0, K_1 = 00102030405060708090a0b0c0d0e0f000f0e0d0c0b0a09080706050403020100 \]

\[ T_0, T_1 = 7e5c3a18f6d4b290e5c3a18f6d4b2907, 1eb852fc630da741b852fc9630da741eb \]

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  \(r = 6\)

\[ C = 2b590ec5954eaa43 \]