IACR Transactions on Symmetric Cryptology
ISSN 2519-173X, Vol. 2025, No. 1, pp. 380–419. DOI:10.46586/tosc.v2025.i1.380-419

Differential Cryptanalysis of the Reduced Pointer Authentication Code Function Used in Arm's `FEAT_PACQARMA3` Feature

Roberto Avanzi[1], Orr Dunkelman[2,3] and Shibam Ghosh[2,4]

[1] Caesarea Rothschild Institute, University of Haifa, Haifa, Israel
roberto.avanzi@gmail.com
[2] Computer Science Department, University of Haifa, Haifa, Israel
orrd@cs.haifa.ac.il
[3] Faculty of Electrical Engineering and Computer Science, TU Berlin, Berlin, Germany
[4] Inria, Paris, France
shibam.ghosh@inria.fr

Abstract. The *Pointer Authentication Code* (PAC) feature in the Arm architecture is used to enforce the *Code Flow Integrity* (CFI) of running programs. It does so by generating a short MAC — called the PAC — of the return address and some additional context information upon function entry, and checking it upon exit. An attacker that wants to overwrite the stack with manipulated addresses now faces an additional hurdle, as they now have to guess, forge, or reuse PAC values. PAC is deployed on billions of devices as a first line of defense to harden system software and complex programs against software exploitation.

The original version of the feature uses a 12-round version the QARMA-64 block cipher. The output is then truncated to between 3 and 32 bits, in order to be inserted into unused bits of 64-bit pointers. A later revision of the specification allows the use of an 8-round version of QARMA-64. This reduction may introduce vulnerabilities such as high-probability distinguishers, potentially enabling key recovery attacks. The present paper explores this avenue.

A cryptanalysis of the PAC computation function entails restricting the inputs to valid virtual addresses, meaning that certain most significant bits are fixed to zero, and considering only the truncated output. Within these constraints, we present practical attacks on various PAC configurations. These attacks, while not presenting immediate threat to the PAC mechanism, show that some versions of the feature do miss the security targets made for the original function. This offers new insights into the practical security of constructing MAC from truncated block ciphers, expanding on the mostly theoretical understanding of creating PRFs from truncated PRPs.

We note that the results do not affect the security of QARMA-64 when used with the recommended number of rounds for general purpose applications.

Keywords: Tweakable Block Ciphers · Lightweight Cryptography · Pseudo-Random Functions · Pseudo-Random Permutations

1 Introduction

The 2016 Additions to the Arm V8 architecture [Arm16] introduced the *Pointer Authentication Code* PAC feature [QPS17] to counter various exploits, including *Return-Oriented Programming* (ROP) [RBSS12].

Received: 2024-09-01 Revised: 2024-11-22 Accepted: 2025-01-23 Published: 2025-03-07

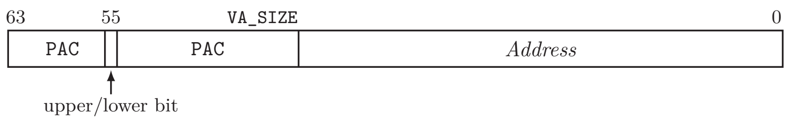


Figure 1: PAC field in an ARM64 pointer.

ROP involves smashing the stack with addresses of *gadgets*, i.e., function epilogues in the target software, together with some information that these gadgets may consume. The gadgets are typically taken from a common library like `libc` [Des97]. Each return instruction directly transfers control to the next gadget, thus creating a *chain* of gadgets, which is a form of threaded code [Bel73]. Since most non-trivial software contains a Turing-complete set of gadgets [Sha07], complex programs can be executed. Tools exist to assist the implementation of an arbitrary program as a gadget chain, such as Q [SAB11], ropc [pak13], PSHAPE [FBP+16], and ROPER [FZHJ17, Fra18].

PAC provides instructions to insert and verify short tags in reserved bits of pointers, as shown in Figure 1. On A-profile CPUs, the size of the PAC field ranges from 3 to 31 bits, depending on factors such as *Virtual Address* (VA) space size and the use of some of the reserved bits for other uses. On M-profile CPUs, a 32-bit PAC is stored separately.

These cryptographically computed tags depend on a secret key and a public value representing the pointer's context. While primarily used for enforcing CFI by protecting function return address, the PAC can also protect v-tables, procedure linkage tables, Objective-C method caches, computed gotos, and other pointers [App24]. This feature is invoked continuously on billions of deployed devices to provide a first line of defense to system software and to complex programs such as browsers. Therefore, its computation must be very fast to avoid slowing down program execution. For this purpose, a new block cipher called QARMA [Ava17] was designed, suggesting that the goal was to have a critical path not exceeding 100 GE, i.e., four to five execution pipeline stages.[1]

The tag is computed by discarding some output bits of a lightweight *Tweakable Block Cipher* (TBC) [LRW02]. A TBC is a block cipher that takes an additional public input called the tweak, which is used together with the key to select the permutation computed by the cipher. For a fixed tweak, a TBC acts as a permutation, while for a fixed plaintext, the tweak-to-ciphertext mapping aims to be a *Pseudo-Random Function* (PRF). When at least half the output bits are discarded, the result is indistinguishable from a PRF [GG15, GGM18, GG21]. This can be seen as an instance of the *Chop-MD* construction [CDMP05, Sec. 3.4]. For PAC, the pointer is the plaintext input and the context is the tweak. It remains an open question to determine the conditions under which such a construction yields a PRF over the combined plaintext-tweak space.

In this paper, we refer to the process of discarding fixed output bits as "chopping" and the resulting value as "chopped" instead of "truncating" and "truncated," in order to avoid confusion with truncation as understood in differential cryptanalysis. Additionally, we use the verb "to clamp" to describe the action of forcing certain input bits to zero.

Several lightweight cryptographic primitives already existed when the PAC feature was developed, including CLEFIA [SSA+07], KATAN [CDK09], KLEIN [GNL11], LED [GPPR11], PRESENT [BKL+07], PRINCE [BCG+12], SIMON and SPECK [BSS+13], and MIDORI [BBI+15], to name just a few. Some of these, suitably round-reduced, do have a critical path of around 100 GE. However, none of these ciphers is tweakable, and if generic methods to construct TBCs from ordinary block ciphers [Rog04, LST12, ST13] are applied to them, the resulting latency would at least double. A different approach, the TWEAKEY framework [JNP14],

[1] With typical pipeline stage delays of 6-8 FO4 inverters plus 2-3 FO4 for latching [HBK+02], and NAND to FO4 delay ratios at least 1.5 and rarely significantly exceeding 2.0 [GVG+17, PP20], critical path usually ranges from 14 to 25 GE.

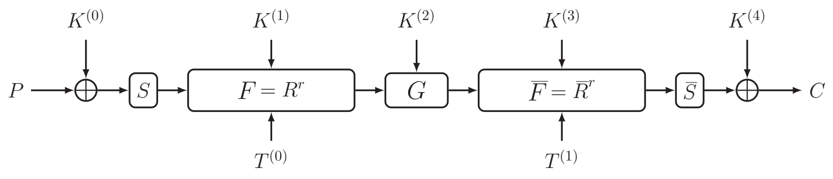


Figure 2: High-level structure of MANTIS, QARMA, and QARMAv2.

takes an existing design, mixes values derived from the tweak in each round, and suitably increases the number of the latter. The TBCs Deoxys-BC, Joltik-BC, and Kiasu-BC are examples of this design, but their latencies are still unsatisfactory, in particular because of the use of MDS diffusion matrices. Some lightweight hash functions existing at the time, such as SIPHASH [AB12], could not be used because they rely on modular additions with long critical paths. These considerations justified the design of a new primitive.

The first lightweight primitive meeting the latency requirement was MANTIS [BJK+16, Section 6]. QARMA followed shortly after, improving upon MANTIS by addressing weaknesses like "copy-and-paste" characteristics due to a suboptimal diffusion layer, and a partitioning of the state into four independent 32-bit paths through the three central rounds [DEKM16].[2] Both ciphers, along with QARMA's successor QARMAv2 [ABD+23], adopt an approach similar to the TWEAKEY framework. They are *reflection ciphers*, i.e., of the form $E = F^{-1} \circ G \circ F$ where F is an encryption function, and a G is called a *reflector*. Even in the cases where G is an involution, E itself is not self-inverting due to key schedule changes between *forward* and *backward* operations. As shown in Figure 2, these ciphers share a common structure. The external rounds of F and F^{-1} — called *half rounds* due to their simplified S-Box only structure — are separated out for clarity.

With PAC enabled, gadgets used in ROP attacks can verify tags but cannot generate them, unless they are entire procedures. An adversary must therefore either: (a) guess tags through trial and error; (b) reuse existing tagged pointers; (c) use code that generates valid tags (a "signing gadget"); or (d) forge tags through cryptanalysis.

Let us have a look at these four attack strategies. Option (a) is unfeasible once the gadget chains, or the tags themselves, are sufficiently long. Sometimes, short or single gadget ROP chain exploits are possible, and if the tag space is sufficiently small, the tags can be brute-forced by spawned processes. Therefore, there should be tests at compile time, or on the compiled binaries, to prevent such short chains. Tagged pointer reuse and the possibilities of signing gadgets are accepted risks [QPS17]. In this paper we are concerned with the fourth and last of the attack strategies listed above: cryptanalysis.

The original PAC architecture uses the 12-round QARMA$_5$-64, that achieves 100 GE (as per [Ava17]) and provided generous security margins. This version of the feature is known as FEAT_PACQARMA5 in the Arm ARM (Architecture Reference Manual) for both A-profile [Arm24] and M-profile [Arm23]. However, in small cores with short pipelines, the required four to five execute pipelines stages can create pipeline bubbles and reduce performance. For this reason, a version of the feature called FEAT_PACQARMA3 was introduced, based on the eight-round QARMA$_3$-64, with a shorter critical path of 66 GE. This can be computed in three stages, improving performance and power efficiency.

Such a heavily round-reduced version of the cipher may allow high probability distinguishers and thus expose the feature to practical cryptanalysis. Since memory read gadgets are relatively common in software environments, even in the form of complete procedures, we can assume that an adversary can read many signed pointers and mount *Known Plaintext Attack*s (KPAs) (strictly speaking, *Known Plaintext-and-Tweak Attacks*).

Furthermore, *Just-in-time* (JIT) environments, such as JavaScript engines, allow

[2] A 128-bit version for memory encryption is also defined, but it is out of scope for this paper.

attackers to generate chosen signed pointers. The latter can be collected for later reuse or for the purpose of mounting *Chosen Plaintext Attack*s (CPAs).

1.1 Security Claims, Security Model, and Goals

The security claims for `MANTIS`$_5$-64 and `QARMA`$_5$-64 are that the ciphers should offer "*security against practical attacks [...] in the sense [...] that no related-tweak attack should be applicable with less than* 2^{30} *chosen or* 2^{40} *known text pairs.*" It is debatable whether the PAC function based on `QARMA`$_3$-64 should be subjected to these criteria, but, since Arm uses it for the A-profile as an alternative to `QARMA`$_5$-64, we believe that holding to the same security claim is justified.

We assume that the attacker has access to a "signing" oracle $\mathcal{E}$ that uses an unknown key K for chosen inputs, such as a JIT environment combined with a memory read gadget.

Out goal is to minimize key recovery time while using at most 2^{30} chosen plaintext-tweak pairs. Since PAC aims to deter software exploitation, we consider attacks with time complexity up to 2^{64} to be breaks of the function. We also consider attacks requiring up to 2^{80} operations as warnings, even though they may not be practical on commodity hardware. This 2^{80} threshold has historical precedent — it was once considered a practical limit for cryptanalysis, as reflected in the use of 80-bit keys in Skipjack [NIS98] and of 160-bit SHA-1 digests [NIS95]. However, these security levels were soon deemed inufficient for long-term use [LV01, Section 5]. Today's Bitcoin network performs $\approx 2^{95.2}$ SHA-256 operations annually at a rate of $750 \cdot 10^6$ TH/s [Blo24], where one "H" involves two SHA-256 calls. This demonstrates that 2^{80} operations is achievable with significant parallel resources — such as the brute force key search in our most expensive attacks. Therefore, while attacks requiring between 2^{64} and 2^{80} operations may not lead to a quick break of the system, advancements in cryptanalytic techniques could close the gap.

1.2 Our Contributions

We apply differential (and multiple differential) cryptanalysis to the PAC function based on `QARMA`$_3$ used in `FEAT_PACQARMA3`.

Our findings reveal that the function does not meet the security bounds for the underlying cipher when pointer lengths are 48 bits or longer, with attacks using fewer than 2^{30} chosen plaintexts and 2^{64} encryptions. The 44-bit pointer version narrowly exceeds these security bounds. While the versions for shorter pointers surpass both bounds, they are susceptible to attacks using fewer than 2^{80} encryptions. The 32-bit M-profile version presents the strongest resistance to our analysis despite exposing more output bits, due to its strict input constraints.

The complete results are summarized in Table 1 together with published cryptanalysis of the unrestricted `QARMA`-64. To understand their relevance, note first that, currently, no Arm cores implement 56-bit VA spaces. 36-bit and 40-bit VA spaces are also increasingly rare, since most modern operating systems tend to favor 48-bit VAs for commodity devices and 52-bit VAs for server applications. The most relevant cases are thus those of VA spaces of 48 and 52 bits and, to a lesser extent, 44 bits.

Our results do not directly translate to practical breaks of the PAC *feature*. Attackers that rely on memory read gadgets can only mount KPAs using program-generated pointers. While JIT environments theoretically enable CPAs, practical constraints significantly limit the choice of inputs. Thus, PAC remains a significant barrier for the attackers.

Comparing to the cryptanalysis published so far, it is worth noting that we improve on the $4 + 4$ rounds attack in [LJ18] in terms of running time and memory usage, despite the restrictions on inputs and outputs.

In order to perform our cryptanalysis we had to overcome a few hurdles. First, the clamping of input bits and chopping of output bits in the PAC function limits the differential

Table 1: Our results, compared to published cryptanalysis of `QARMA`-64. Rounds are counted as $x+y$, where x and y are S-Box layers before and after the reflector, respectively. For the attacks on the PAC function we also mention the VA space size, which determines the number of clamped plaintext cells and of chopped ciphertext cells, and "A" and "M" refer to A-profile and M-profile architectures. Memory is given in cipher blocks.

	Rounds attacked	`VA` & Profile	Additions of w, $o(w)$	Attack Complexity Data	Attack Complexity Time	Attack Complexity Memory	Technique	Reference
	$4+6$		N	2^{53} CP	$2^{116}+2^{70.1}$	2^{116}	Meet-in-the-Middle	[ZD16]
	$3+8$		N	$2^{58.38}$ CP	$2^{64.92}$	$2^{63.38}$	Imp. Diff.	[LZG+20]
	$4+4$		Y	2^{16} CP	$2^{33}+2^{90}$	2^{90}	Meet-in-the-Middle	[LJ18]
This paper	$4+4$	*56 (A)*	*Y*	2^{25} CP	2^{60}	$2^{28.98}$	*Rel-Tweak (RT) Diff.*	*Section 5.1*
	$4+4$	*52 (A)*	*Y*	$2^{25.3}$ CP	2^{55}	$2^{28.94}$	*RT Diff.*	*Section 5.2*
	$4+4$	*48 (A)*	*Y*	2^{28} CP	2^{60}	$2^{40.83}$	*RT Mult. Diff.*	*Section 5.3*
	$4+4$	*44 (A)*	*Y*	$2^{31.19}$ CP	2^{68}	$2^{31.67}$	*RT Diff.*	*Section 5.4*
	$4+4$	*40 (A)*	*Y*	$2^{37.5}$ CP	2^{76}	$2^{38.82}$	*RT Diff.*	*Section 5.5*
	$4+4$	*36 (A)*	*Y*	$2^{33.7}$ CP	2^{76}	$2^{35.02}$	*RT Diff.*	*Section 5.6*
	$4+4$	*32 (A)*	*Y*	2^{42} CP	2^{76}	$2^{44.32}$	*RT Diff.*	*Section 5.7*
	$4+4$	*32 (M)*	*Y*	2^{45} CP	2^{76}	$2^{46.32}$	*RT Diff.*	*Section 5.7*
	$4+5$		Y	2^{16} CP	$2^{48}+2^{89}$	2^{89}	Meet-in-the-Middle	[LJ18]
	$4+6$		Y	2^{61} CP	2^{72}	$2^{72.2}$	Trunc. Imp. Diff.	[YQC18]
	$4+6$		Y	2^{59} KP	2^{59}	$2^{23.6}$	RT Stat. Sat.	[LHW19]
	$4+6$		Y	$2^{47.12}$ CP	$2^{75.13}$	2^{72}	RT Trunc. Diff.	[SII23]
	$5+5$		Y	$T \cdot D$ (CP) $= 2^{119.8}$		2^{31}	Rel-Tweak Imp. Diff.	[ZD19]
	$5+5$		Y	$2^{47.06}$ CP	$2^{83.53}$	2^{80}	RT Trunc. Diff.	[SII23]
	$3+8$		Y	2^{61} CP	$2^{64.4}+2^{80}$	2^{61}	Imp. Diff.	[ZD18]
	$4+7$		Y	2^{61} CP	$2^{120.4}$	2^{116}	Trunc. Imp. Diff.	[YQC18]
	$5+6$		Y	$2^{34.26}$ CP	$2^{111.16}$	2^{108}	RT Trunc. Diff.	[SII23]
	$4+8$		Y	$2^{48.4}$ CP	$2^{66.2}$	$2^{53.70}$	Zero Corr./Integral	[ADG+19]
	$6+5$		Y	2^{54} CP	$2^{66.35}$	2^{64}	RT Differential	[CXTQ23]

characteristics we can use, requiring us to perform extended searches. To address this, we leverage the cipher's symmetry around the center by forcing center differences to zero during some clustering searches and while building a multiple differential. While this only provides bounds rather than exact values, experiments shows that the omitted characteristics have negligible impact (cf. Section 5.2 and Section 5.3.1). For some attacks, this hurdle also forces us to find more than one characteristic, often with very different properties from each other, in order to recover sufficiently many key bits.

Second, the small number of visible output bits creates high noise when trying to identify the correct key guess. We adopt a statistical approach to determine minimal data requirements to overcome this noise (cf. Section 3.5). The approach is experimentally validated as well (cf. Section 4.6).

The fixed total of 64 usable input and output bits means that having more visible output bits necessarily restrict input bits, and vice versa. Despite this inherent trade-off between input flexibility and output noise, we are able to provide some successful attacks.

Because of the restrictions on the useable characteristics and the low data bounds, in our attacks the brute force step is always the most expensive part.

The security of `FEAT_PACQARMA5` and that of `QARMA`-64 with the recommended rounds for general use is unaffected by our results.

1.3 Outline

In Section 2 we recall the definitions of `QARMA` and of the PAC function. Section 3 describes Chosen-Input Key Recovery for the `FEAT_PACQARMA3` function, and describes how we determine the success probability and the data and time complexities of a key recovery step. In Section 4 we describes how we find suitable characteristics for the attacks. In

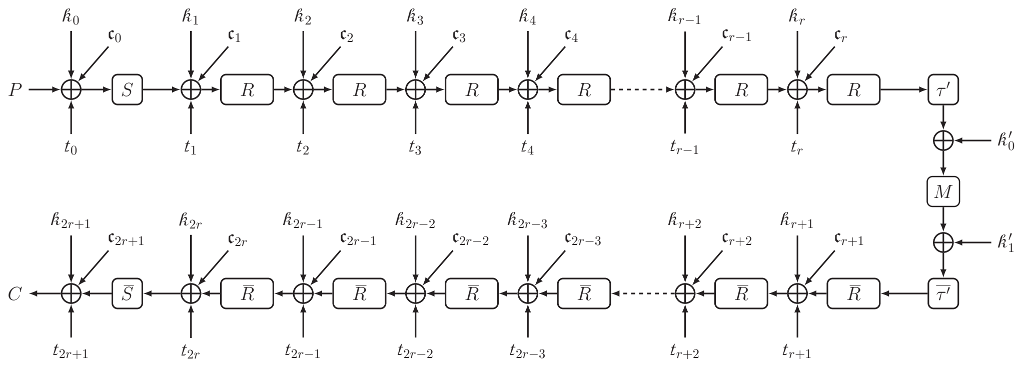


Figure 3: The detailed common structure of `MANTIS`, `QARMA`, and `QARMAv2`.

Section 5 we detail our attacks on the `FEAT_PACQARMA3` function. We conclude in Section 6.

2 Background

2.1 `QARMA`

Definition 2.1. *Let $E : \mathbb{F}_2^n \times \mathbb{F}_2^k \times \mathbb{F}_2^{n_t} \to \mathbb{F}_2^n$ be a function $(P, T, K) \mapsto C = E_{K,T}(P)$. The function E is called a TBC if, for each fixed value of the pair (K, T), it is a bijection of $\mathbb{F}_2^n$ onto itself. The n-bit input P is called the* plaintext*, the n-bit output C is called the* ciphertext*, the n_t-bit input T is called the* tweak*, and the k-bit input K is called the* key*. The key is secret, while the tweak is assumed to be public.*

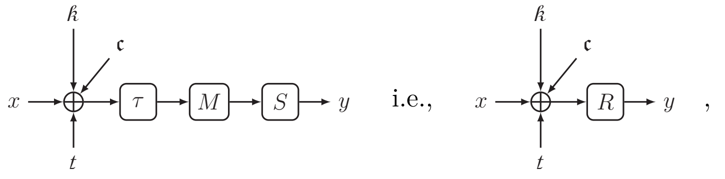
We recall here the definition of `QARMA` (v1). Its structure is represented in Figure 3, which can also be used to describe `MANTIS` and `QARMAv2`. The differences between the three ciphers, while they may appear minor, have profound implications on their security. The notations $\mathfrak{k}$, resp. t, $\mathfrak{c}$ denote a round key, resp. tweak and constant: The `XOR` of these values at the same round is the latter's *round tweakey*.

Each state of `QARMA` is represented as a 4×4 matrix of cells

$$A = \begin{pmatrix} A[0] & A[1] & A[2] & A[3] \\ A[4] & A[5] & A[6] & A[7] \\ A[8] & A[9] & A[10] & A[11] \\ A[12] & A[13] & A[14] & A[15] \end{pmatrix} ,$$

and 4×4 matrices operate column-wise on the state by left multiplication. With respect to cell numbering, the 64 bits of a state are indexed in big endian order and the bits in a cell are indexed in little endian order: bits [63 — 60] are contained in Cell 0 and bits [3 — 0] in Cell 15. The bits in a cell can also be referred to using the square bracket notation.

A full round has the following structure:



where $R = S \circ M \circ \tau$. S consists of a layer of sixteen identical S-Boxes, applied to the sixteen cells in parallel. M is an involutory Almost-MDS matrix. τ is the `MIDORI` cell permutation, which with our cell numbering is $\tau = [\,0, 11, 6, 13, 10, 1, 12, 7, 5, 14, 3, 8, 15, 4, 9, 2\,]$, acting on a state as $\tau(A)[i] = A[\tau(i)]$ for $0 \leq i \leq 15$. The 4×4 matrix M operates column-wise by left multiplication on each layer of the block. S is the parallel application of the same S-Box to all 16 cells of the state. The half-round function, only used for the outermost rounds of the cipher, consists of just a round key addition and an S-Box layer.

The matrices are involutory circulants of the form

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

over a ring with zero divisors, $R_m = \mathbb{F}_2[\rho] = \mathbb{F}_2[X]/(X^m + 1)$, where ρ is a circular left rotation of the bits.

We describe only `QARMA`'s encryption, as decryption uses the same circuit with a different configuration of round keys and tweaks, which is simple to derive and found in [Ava17].

The 128-bit key is $K = w^0 \| k^0$ where the 64-bit values w^0 and k^0 are the *whitening* and *core* keys. Put $w^1 = o(w^0) := (w^0 \ggg 1) + (w^0 \gg 63)$ and $k^1 = k^0 + \alpha$. The round keys $\mathscr{k}_0$ and $\mathscr{k}_{2r+1}$ are equal to $w^0 + k^0$ and $w^1 + k^1$. Then, $\mathscr{k}_i = k^0$ for $1 \leq i < r$, and $\mathscr{k}_i = k^1$ for $r + 2 \leq i \leq 2\,r$, where α is a constant. The round keys at the sides of the reflector are $\mathscr{k}_r = w^1$ and $\mathscr{k}_{r+1} = w^0$. The central round keys are $\mathscr{k}'_0 = 0$ and $\mathscr{k}'_1 = k^0$. Also, $\tau' = \tau$.

For PAC, `QARMA` uses the S-Box $\sigma_0 := [\,0, 14, 2, 10, 9, 15, 8, 11, 6, 4, 3, 7, 13, 12, 1, 5\,]$. The matrix M has parameters $a = 1$, $b = 2$, $c = 1$. The round constants are consecutive digits of the fractional part of π, with $\mathfrak{c}_0 = 0$. The tweak schedule is $t_{i+i} = \omega(h(t_i))$ for $0 \leq i < r$ where h is the *tweak shuffle* $h = [\,6, 5, 14, 15, 0, 1, 2, 3, 7, 12, 13, 4, 8, 9, 10, 11\,]$ and ω is a LFSR that maps cell (b_3, b_2, b_1, b_0) to $(b_0 + b_1, b_3, b_2, b_1)$, applied to Cells 0, 1, 3, 4, 8, 11, and 13. The tweak schedule is symmetric, i.e., $t_{2r+1-i} = t_i$. For the rationale behind these choices of components, we refer readers to the original design paper [Ava17].

2.2 The PAC Functions

2.2.1 A-Profile Version

We refer here to [Arm24, Section D8.8]. The admissible VA space sizes `V` in the Arm architecture(s) range from 32 to 56 bits in steps of 4 bits, as well as 42 bits, where the 56-th bit is always reserved. The system configuration bit `tbi` ("Top Byte Ignored") indicates whether the top byte of an address is used for address match for the `TTBRx_EL1` (`x` can be 0 or 1) region, or ignored and used for tagged addresses. If `tbi` is 0, the PAC is computed on $s^{64-\mathtt{V}} \| \mathtt{ptr}[\mathtt{V}\text{-}1{:}0]$, where $s \in \{0, 1\}$, and it is inserted in bits [63:56] and [54:`V`]. The effective PAC field length is thus $63 - \mathtt{V}$ bits. The bit s is called the "upper/lower" bit and it is commonly used to separate user space from system stack nemory. If `tbi` is 1, the PAC is computed on $\mathtt{ptr}[63{:}56] \| s^{56-\mathtt{V}} \| \mathtt{ptr}[\mathtt{V}\text{-}1{:}0]$, where $s \in \{0, 1\}$, and it is inserted in bits [54:`V`]. The effective PAC field length is thus $55 - \mathtt{V}$ bits.

The PAC bits are taken from the corresponding bits of the `QARMA` ciphertext output. For PAC calculation, pointers undergo sign-extension. We assume that the pointers in an attack are either all in user space or all in kernel space, ensuring zero difference in the sign extension bits. For the purpose of cryptanalysis, the first $z = (64 - \mathtt{V})/4$ cells of the plaintext input are clamped and the last $16 - z$ cells of the ciphertext are chopped.

2.2.2 M-Profile Version

We follow [Arm23, Section B6.1.1]. In M-Profile CPUs, pointers are 32 bits long, so the plaintext input is padded with 32 zeros, and the PAC consists of the 32 least significant bits of the cipher's output. Thus, in the cryptanalysis, the first eight cells of the plaintext are clamped and the first eight cells of the ciphertext are chopped.

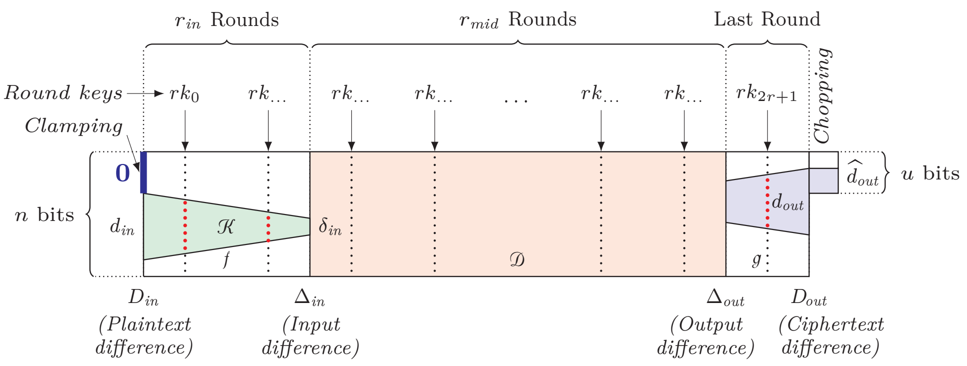


Figure 4: Key Recovery in a Differential Attack on a Clamped and Chopped Cipher.

3 Differential Key Recovery Attack on the PAC Function

Biham and Shamir's fundamental concept of a *differential characteristic* [BS90, BS91] is the starting point of our analysis.

Definition 3.1. *We call $\Delta = (\Delta_{in}, \Delta_{out}) \in \mathbb{F}_2^n \times \mathbb{F}_2^n$ a differential for a block cipher with block size n that covers a section $\mathcal{D}$ of the cipher consisting of r_{mid} rounds where Δ_{in} is the input difference and Δ_{out} is the output difference. For a TBC, this is associated with a tweak difference D_T, and we write $\Delta = (\Delta_{in}, \Delta_{out}, D_T) \in \mathbb{F}_2^n \times \mathbb{F}_2^n \times \mathbb{F}_2^{n_t}$, where n_t is the tweak's bit size. We denote the probability of a differential by 2^{-q} for some real value of q.*

To apply differential cryptanalysis to an iterated block cipher, we need to identify a *differential characteristic*, i.e., a sequence of differences that describes how a difference propagates through the successive rounds of the cipher, and with which probability.

Definition 3.2. *When $\Delta = (\Delta_{in}, \Delta_{out}) \in \mathbb{F}_2^n \times \mathbb{F}_2^n$ is fully specified in the sense that each intermediate state difference is described, we call it a differential characteristic. An r-round differential characteristic is a series of differences, denoted as $(a_0 \to a_1 \to \cdots \to a_r)$, where $a_0 = \Delta_{in}$, and $a_r = \Delta_{out}$.*

Definition 3.3. *A differential can correspond to several differential characteristics. We call a set of characteristics of the form $(a_0 \to a_1 \to \cdots \to a_r)$ with $a_0 = \Delta_{in}$ and $a_r = \Delta_{out}$, a* cluster *corresponding to the differential $\Delta = (\Delta_{in}, \Delta_{out})$. The probability of any differential from input difference Δ_{in} to output difference Δ_{out} can be computed by taking the sum over the probabilities of all possible characteristics of the form $(a_0 \to a_1 \to \cdots \to a_r)$ with $a_0 = \Delta_{in}$ and $a_i = \Delta_{out}$. We refer to Section 4.5 to see how clustering is performed.*

To run key recovery on $\mathcal{E}$ using Δ, we extend the differential by adding rounds at one or both ends until we cover the whole cipher. This is done by tracing the potential *activeness* of the cells, but not the actual *differences*, starting from Δ_{in}, resp., Δ_{out}, and going outwards through the added rounds until we reach D_{in}, resp., D_{out}. In Section 4 we describe how to trace this diffusion through the components of `QARMA`. Our goal is to recover the set of round key bits $\mathcal{K}$ affecting the differences in the rounds added in the front. We use the back of the function only as a distinguisher. Figure 4 illustrates our settings, and with reference to it, we have $\mathcal{E} = g \circ \mathcal{D} \circ f$, where f consists of the initial r_{in} rounds and g is the final (half) round of the cipher.

Following common (abuse of) terminology, we use the terms *input difference* and *output difference* to refer to Δ_{in} and Δ_{out}, respectively, even for extended characteristics, for which D_{in} and D_{out} are called the *plaintext* and *ciphertext differences.* The ciphertext output is then chopped to $u \leq n$ bits, discarding the other $n - u$ bits (in our case $n = 64$).

Definition 3.4. *The dimensions of D_{in}, D_{out}, resp., Δ_{in}, are d_{in}, d_{out}, resp., δ_{in}. We further define $\widehat{D}_{out}$ as the subset of D_{out} that can be observed in the u visible bits, and $\widehat{d}_{out}$ is its dimension, i.e. $\left|\widehat{D}_{out}\right| = 2^{\widehat{d}_{out}}$.*

Remark 3.1. *Note that Δ may be a truncated (or multiple) differential, where at least one of Δ_{in}, Δ_{out}, and D_T is a set. The non-truncated case corresponds to $|\Delta_{in}| = |\Delta_{out}| = |D_T| = 1$. For simplicity, in this section we consider only a non-truncated Δ. In Section 5.3.1 we straightorwardly adapt the treatment to the multiple differential case,*

3.1 The Key Recovery Algorithm

We give first a high-level description of the various steps of the key-recovery process. The details are given in the following subsections. Pseudo-code is given in Algorithm 1.

First, we repeatedly query the encryption oracle to generate pairs of plaintext-tweak-ciphertext triplets $\left((P_0^i, T_0^i, C_0^i), (P_1^i, T_1^i, C_1^i)\right)$ that, by construction, *satisfy* D_{in} and D_T, i.e., $P_0^i + P_1^i \in D_{in}$ and $T_0^i + T_1^i \in D_T$. Suppose also that N of these pairs satisfy Δ_{in}, i.e., their differences at the start of $\mathcal{D}$ are in Δ_{in}. We describe in Section 3.2 how we use *structures* so that we get a desired number N of pairs that satisfy Δ_{in}. For now we just assume that N is the number of pairs that satisfy Δ_{in}, and that one in 2^q of these N pairs satisfies the output difference Δ_{out} — we call the latter *right pairs.* They of course satisfy the chopped ciphertext difference $\widehat{D}_{out}$ as well. With a right pair, we can use the plaintext values and the difference at Δ_{in} to deduce the right key, as explained in Section 3.4. For an attack to be successful, we need at least one right pair, i.e., $N \geq 2^q$.

However, wrong (i.e., non-right) pairs may also satisfy $\widehat{D}_{out}$ by random chance with probability $2^{-u+\widehat{d}_{out}}$, and such pairs may suggest *any* key value. The right key will thus be suggested with probability $2^{-q} + 2^{-u+\widehat{d}_{out}}$. (More precisely, since the proportion of wrong pairs is $1 - 2^{-q}$, the right key will be suggested by a pair with input difference D_{in} with probability $2^{-q} + 2^{-u+\widehat{d}_{out}}(1 - 2^{-q})$, but we are assuming 2^{-q} too small.) The noise from wrong pairs can make it difficult for the right to stand out, especially when $2^{-q} \ll 2^{-u+\widehat{d}_{out}}$. To overcome this noise, we must choose a sufficiently large N. Choosing N is non-trivial and is described in Section 3.5.

After data collection we perform *Pair Sieving*, i.e., we keep only the L, say, pairs that satisfy $\widehat{D}_{out}$, i.e., $C_0^i + C_1^i \in \widehat{D}_{out}$. While this could be performed by building a list of length L, we instead use a hash table keyed by the values taken by the chopped ciphertext at the inactive cells of $\widehat{D}_{out}$ to enable efficient *enumeration* of the pairs that satisfy $\widehat{D}_{out}$.

Finally, we perform the actual key recovery through the front r_{in} rounds. For each sieved pair $\left((P_0^i, T_0^i, C_0^i), (P_1^i, T_1^i, C_1^i)\right)$ and each key guess $\kappa \in \mathcal{K}$ we test whether $f_\kappa(P_0^i, T_0^i) + f_\kappa(P_1^i, T_1^i) \in \Delta_{in}$. When this happens, we say we have a *positive match* that suggests κ as a key candidate, and we increase a counter associated with κ. The right key should appear among the most frequently suggested keys across all pairs.

Since the S-Box is bijective, given a non-zero output difference, an input pair with a non-zero difference produces on average one solution. Hence, the sum of all counters is expected to be L. To track key counters, we avoid full tables when most entries are zero, as these waste memory, need a lengthy initialization step, and have poor cache locality. Instead, we store non-zero counters only in self-balancing trees [Knu98, Section 6.2.3]. In our attacks, AVL trees [AVL62] give the best performance compared to Red-Black trees [Bay72], splay trees [ST85], and full tables. For cases with just a handful of non-zero counters, we could use a bubble-sorted list [Fri56] of key-counter pairs, but this does not occur in our attacks.

Algorithm 1 Basic Key Recovery Program.

Input: A tweakable encryption oracle $\mathcal{E}$, a characteristic $D_{in} \times D_T \mapsto \Delta_{in} \mapsto \Delta_{out} \mapsto D_{out}$ and a set of round key bits $\mathcal{K}$ intersecting the active cells in the additional front rounds.

Output: A dictionary associating counters with the elements in $\mathcal{K}$.

```
▷ Data Collection and Pair Sieving Stage
 1: $\mathcal{C} = \emptyset$
 2: for $s$ distinct structures $\mathcal{S}$ do
 3:     Prepare $\mathcal{L}_0 = \{(P, T_0, \mathcal{E}_{T_0,K}(P)) \mid (P, T_0) \in \mathcal{S}_0\}$
 4:     Prepare $\mathcal{L}_1 = \{(P, T_1, \mathcal{E}_{T_1,K}(P)) \mid (P, T_1) \in \mathcal{S}_1\}$
 5:     for $((P_0, T_0, C_0), (P_1, T_1, C_1)) \in \mathcal{L}_0 \times \mathcal{L}_1$ do
 6:         if $C_0 + C_1 \in D_{out}$ then
 7:             $\mathcal{C} = \mathcal{C} \cup ((P_0, T_0, C_0), (P_1, T_1, C_1))$
▷ Data Analysis Stage (Guess-and-Filter)
 8: Prepare a dictionary $\chi$ to associate counters with the elements $k \in \langle\mathcal{K}\rangle$
 9: for $((P_0, T_0, C_0), (P_1, T_1, C_1)) \in \mathcal{C}$ do
10:     for $k \in \langle\mathcal{K}\rangle$ do
11:         if $f_k(P_0^i, T_0^i) + f_k(P_1^i, T_1^i) \in \Delta_{in}$ then
12:             $\chi(k) = \chi(k) + 1$
13: return $\chi$
```

3.2 Structures and Complexity

To efficiently generate plaintext pairs, we use *structures*, a technique dating back to [BS92]. We present our adaptation of the concept tailored to our specific case.

Definition 3.5. *A* structure $\mathcal{S}$ *consists of two affine subspaces* $\mathcal{S}_0, \mathcal{S}_1 \subseteq \mathbb{F}_2^n \times \mathbb{F}_2^{n_t}$*, which we call* half-structures*, each half-structure being constant on* $\mathbb{F}_2^{n_t}$*. Here* $\mathcal{S}_1 = (\mathbf{x}_0, \mathbf{t}_0) + \mathcal{S}_0$ *where* $\mathbf{x}_0$ *and* $\mathbf{t}_0$ *are the* fixed *differences in the input pairs restricted to the plaintext and to the tweak, respectively.*

Remark 3.2. *When there are no bits with a fixed non-zero difference at the input and no tweak difference, then* $\mathcal{S}_1 = \mathcal{S}_0$*. As in this paper we use a non-zero tweak difference in the attacks, the remainder of this paper assumes two subspaces, i.e.,* $\mathcal{S}_1 \neq \mathcal{S}_0$*.*

All the elements of $D_{in} \times D_T$ are of the form $((P_0, T_0), (P_1, T_1))$ with $(P_i, T_i) \in \mathcal{S}_i$. Since the $\mathcal{S}_i$ have dimension d_{in}, each structure generates $2^{2d_{in}}$ pairs, out of which $2^{d_{in}}$ satisfy Δ_{in}. With 2^s structures we generate $2^{s+d_{in}}$ such pairs. To generate N pairs, we distinguigh two cases, depending on the relative magnitude of $2^{d_{in}}$ and N.

If $2^{d_{in}} \leq N$, then we have $s = \log_2 N - d_{in}$ and we need to query the encryption oracle $2 \cdot 2^{s+d_{in}} = 2^{\log_2 N+1} = 2\,N$ times.

If $2^{d_{in}} > N$, a complete structure generates more data than needed. Instead, we use a *partial structure*, where each half-structure $\mathcal{S}_i$ is a randomly chosen subset of $\mathbb{F}_2^n \times \mathbb{F}_2^{n_t}$, and is constant on $\mathbb{F}_2^{n_t}$. If we pick 2^x values out of $2^{d_{in}}$ from each of $\mathcal{S}_0$ and $\mathcal{S}_1$, the expected number of pairs that satisfy Δ_{in} is approximately $2^{2x-d_{in}}$. We want $2^{2x-d_{in}} = N$, that is $x = (\log_2 N + d_{in})/2$, hence we need to query the encryption oracle $2\sqrt{N \cdot 2^{d_{in}}}$ times.

In some cases (namely, where we use conditional characteristics) we have to use multiple partial structures, which requires minomal adjustments to data and time complexity (cf. Section 4.6 for the background, Section 5.3.1, and Section 5.4 for concrete applications).

For each pair that satisfies $\widehat{D}_{out}$, we perform a guess-and-filter step, whose cost we denote by T_{GF}. We determine the cost of such steps in Section 3.4. The number of non-zero counters is expected to be relatively small, hence the time for updating them is usually negligible, but its worst case is $\min(2^{|\mathcal{K}|}, L)$.

3.3 Memory Usage

Each data triplet (P^i, T^i, C^i) needs only two 64-bit *blocks* worth of storage, since the tweaks are constant per half-structure. By offloading structure data to external storage, we need to keep in RAM only at most one structure worth of collected data.

For the hash table we need at most one block (which can hold either a plaintext or a pointer depending on the implementation) per entry in the structure currently in memory, as well as one block per each of the $2^{u-\widehat{d}_{out}}$ possible hashing keys.

Regarding the dictionary structure, let us consider an AVL tree. An internal node contains two pointers, a key, and a counter, which in our attacks always fit in 3 blocks. A leaf counter does not contain the pointers, hence it fits in one block. Since at least $1/\varphi$ of the nodes are leaves (where $\varphi = (1+\sqrt{5})/2$), we have that the average size of a node does not exceed $3 \cdot (1 - 1/\varphi) + 1/\varphi \approx 1.764$ blocks. We allocate the blocks in chunks, to minimize the overhead and performance impact from the system memory allocator. The number of nodes is $\ell = \min\left(2^{|\mathcal{K}|}, L\right)$. Since the size of the AVL tree can be estimated in advance, we revert to a table of counters in case the latter is expected to be smaller.

Computing the memory storage is a routine application of the three estimates above to the data provided with the attacks. If an attack uses more characteristics, we reuse the memory and thus consider only the maximum memory usage across all characteristics. Full results are tabulated in Table 1 to provide a comparison with previous work.

3.4 Implementing Guess-and-Filter

3.4.1 Guess-and-Filter in a 0.5 R Attack

Let us start with the case where a half-round is added in front of the characteristic. This is called a 0.5 R (for half round) attack. We denote the set of the indices of the active cells at the beginning of $\mathcal{D}$ by $\Im \subseteq [0..15]$. Note that $\Im$ coincides with the set of active cells in D_{in}, except possibly for those affected by the tweak addition. Let $A[i]$ denote the i-th cell of a state A. Given a sieved pair $\big((P_0, T_0, C_0), (P_1, T_1, C_1)\big)$, we aim to obtain key suggestions $(k+w)[i]$ for $i \in \Im$. As Δ_{in} is the output difference of the first S-box layer, we recover i-th cell of $k+w$ by solving the following equations

$$\Delta_{in}[i] = S\big(P_0[i] + T_0[i] + (k+w)[i]\big) + S\big(P_1[i] + T_1[i] + (k+w)[i]\big) \quad \text{for } i \in \Im \ , \qquad (1)$$

where S is the S-box. We can rewrite Equation (1) as

$$\Delta_{in}[i] = S\big(x[i]\big) + S\big(x[i] + P_0[i] + P_1[i] + T_0[i] + T_1[i]\big) \quad \text{for } i \in \Im \ , \qquad (2)$$

where $x[i] = P_0[i] + T_0[i] + (k+w)[i]$. Solving such equations is easy since the differences $P_0[i] + P_1[i] + T_0[i] + T_1[i]$ and $\Delta_{in}[i]$ are known. Following [BS90, BS91], we store not only the number of solutions in the *Difference Distribution Table* (DDT) but also the solutions themselves. By solving each of these equations we obtain suggestions for each $(k+w)[i]$, and we denote the set of suggested keys by $\mathcal{K}_i$ for each $i \in \Im$. After that, we combine the sets $\mathcal{K}_i$ suggestions for each $i \in \Im$.

For each sieved pair, the cost of key recovery through a half-round at the front depends on the size of $\Im$. Note that while $P_0[i] + P_1[i] + T_0[i] + T_1[i]$ and $\Delta_{in}[i]$ are fixed, for any active S-box there are at most 4 solutions to Equation (2) due to the *differential uniformity* of the used S-box being 4. Out of the 255 entries (excluding the entry corresponding to $0 \mapsto 0$) in the DDT, 102 are non-zero: 84 entries have the value 2 and 18 the value 4. On average, for a random non-zero input/output pair, we expect a solution, but if there are solutions, there are either 2 or 4. In most of the characteristics used in this paper, the size of $\Im$ is at most 7. Hence, when a solution exits, there are about 2^7 solutions, the same number as the amount of S-Boxes in $\texttt{QARMA}_3$. Thus, we bound the cost of T_{GF} in a 0.5 R attack from above with the cost of a single $\texttt{QARMA}_3$ encryption, which we denote by $T_{\mathcal{E}}$.

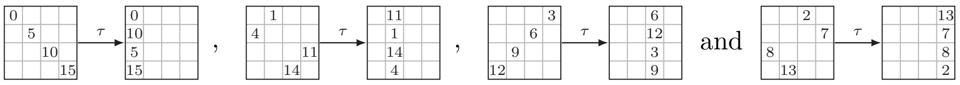


Figure 5: The four cell quartets of a state mapped by τ to columns.

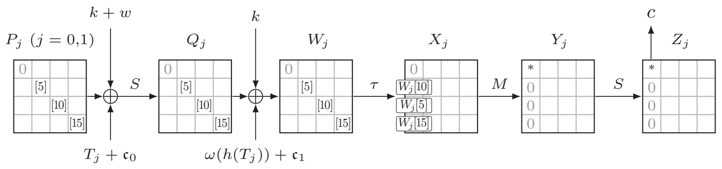


Figure 6: Attacking 1.5 rounds, naïvely.

3.4.2 Guess-and-Filter in a 1.5R Attack

Adding 1.5 rounds to the differential characteristic is also called a 1.5R attack. As before, we consider the case where the rounds are added to the front. The naïve way of performing a 1.5R attack is to try all the involved key bits in $k+w$ and k.

Due to the matrix operation in the full round, we must examine Δ_{in} column-wise. Consider the four cells of D_{in} that map to an active column c of Δ_{in} after 1.5 rounds: The four cell quartets of D_{in} mapped to the four columns of Δ_{in} by τ are given in Figure 5. For conciseness, we consider only the first quartet, corresponding to the first column, the other cases being corresponding to permutations.

If only one cell of c is active, then exactly three of the four cells in D_{in} are active unless the fourth is activated by a tweak addition or one is cancelled by the tweak difference. Given a sieved pair $\big((P_0, T_0, C_0), (P_1, T_1, C_1)\big)$, up to a permutation of the indices, the situation is equivalent to the one depicted in Figure 6, where 0 indicates a zero difference. Let $\mathcal{G}$ denote the set of the three cells activated by c in D_{in}, in our example $\mathcal{G} = \{5, 10, 15\}$. Note that here we attempt to recover the key bits of $(k+w)$ and k that are involved while we extend Δ_{in} backward to D_{in}. However, in the case of the key bits in k, it is more helpful to recover the *equivalent key* k' where $k' = M(\tau(k))$ is added before the second S-box operation as shown in Figure 7. This results in the following relations

$$\begin{cases} Q_i[j] = S\big(P_i[j] + T_i[j] + (k+w)[j] + \mathfrak{c}_0[j]\big) \text{ for } i \in \{0,1\} \text{ and } j \in \{5,10,15\} \\ W_i[j] = Q_i[j] + \omega\big(h(T_i)\big)[j] + \mathfrak{c}_1[j] \\ X_i[0] = W_i[0] \ , \ X_i[4] = W_i[10] \ , \ X_i[8] = W_i[5] \ , \ X_i[12] = W_i[15] \\ Y_i[0] = \rho(X_i[4]) + \rho^2(X_i[8]) + \rho(X_i[12]) \qquad\qquad (3a) \\ Y_i[4] = \rho(X_i[0]) + \rho(X_i[8]) + \rho^2(X_i[12]) \Rightarrow \Delta_Y[4] = \rho(\Delta_X[8]) + \rho^2(\Delta_X[12]) \\ Y_i[8] = \rho^2(X_i[0]) + \rho(X_i[4]) + \rho(X_i[12]) \Rightarrow \Delta_Y[8] = \rho(\Delta_X[4]) + \rho(\Delta_X[12]) \\ Y_i[12] = \rho(X_i[0]) + \rho^2(X_i[4]) + \rho(X_i[8]) \Rightarrow \Delta_Y[12] = \rho^2(\Delta_X[4]) + \rho(\Delta_X[8]) \\ \Delta_Z[0] = S\big(Y_0[0] + k'[0]\big) + S\big(Y_1[0] + k'[0]\big) \ , \qquad\qquad (3b) \end{cases}$$

where Δ_v denotes $v_0 + v_1$ for any variable pair $v = (v_0, v_1)$. These relations imply that in order to recover, say $k'[0]$, we also need to either know or to guess $(k+w)[\mathcal{G}]$.

We now describe the *peeling* strategy (see the 3R attack in [BS91, Section 5]) to recover one or more cells of k' — for simplicity let us stick to $k'[0]$. Assuming that we already mounted a 0.5R attack with the same D_{in} and $\widehat{D}_{out}$, we can reuse the same data set. We repeat the pair sieving iterator of the 0.5R attack, but this time instead of testing for round key candidates, we we simply check whether the current pair suggests the already recovered right key bits of $(k+w)$. For these pairs and each possible guess of the as-of-yet unknown bits of $(k+w)[\mathcal{G}]$ that contribute to $k'[0]$, we determine candidates for $k'[0]$. The

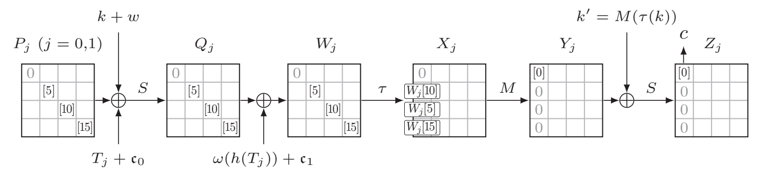


Figure 7: Attacking 1.5 rounds, using an equivalent key on the second round, landing on a column with one active cell.

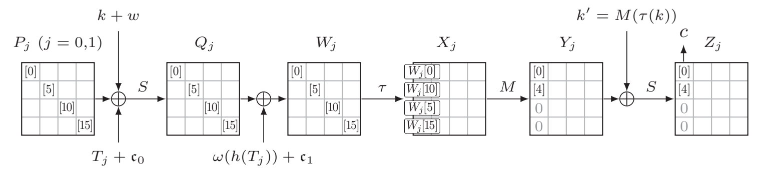


Figure 8: Attacking 1.5 rounds, using an equivalent key on the second round, landing on a column with two active cells.

counters correspond to the guessed bits $(k+w)[\mathcal{G}] \cup k'[0]$. To recover k' in our example, we compute the values $Y_0[0]$ and $Y_1[0]$ from Equation (3a). Note that we either know P_0, P_1, T_0, T_1 as well as the cells of $k+w$ that are required to compute $Y_i[0]$, or we need to fix a guess for the unknown cells of $k+w$ that contribute to $k'[0]$. Finally, we consider the following equation which is equivalent to Equation (3b), and we solve it for $k'[0]$

$$\Delta_Z[0] = S(x[0]) + S\big(Y_1[0] + Y_0[0] + x[0] + k'[0]\big)$$

where $x[0] = k'[0] + Y_1[0] + Y_0[0]$. Since the rounds of filtering are relatively rare, and the number of key bits to recover in a peeling step — including both bits to be guessed and those to be filtered — is often smaller than in the initial 0.5R attack step, making the the cost of a peeling step is bounded by that of a 0.5R attack. This is the case in all our attacks.

The case where in the column c two cells (Figure 8) or more are active is similar. As the number of active cells in c increases, the number of cells with zero differences in Δ_Y decreases, and at the same time we can recover more bits of k'. There is thus a trade-off between the number of key bits recovered and the time complexity. In this paper, we use characteristics with at most two active cells in c.

Remark 3.3. *Peeling can be done directly after a 0.5R attack even if did not start with a differential for a 1.5R attack. With reference to Figure 4, we perform key recovery also* inside $\mathcal{D}$. *The transition through the second round is not always unique, and sometimes there are multiple transitions with the same probability. However, if we fix the second round transition as in the original characteristic of the 0.5R attack, the probability of the differential obtained by removing the first round of $\mathcal{D}$ is still higher than that of $\mathcal{D}$. In particular, it allows us to recover correct key bits without the need for additional data.*

3.5 The Poisson Distribution Argument

The relation between data complexity, time complexity and success probability is quite complex. However, ignoring the latter easily leads to oversimplifications and too optimistic complexity estimates. This subsection describes the approach we follow in this paper to determine all three quantities together: the *Poisson distribution argument.*

For each integer a, let $\omega(a)$ be the probability that the counter of a fixed wrong key is equal to a. The probability that the counter of *one, fixed* wrong key is smaller than

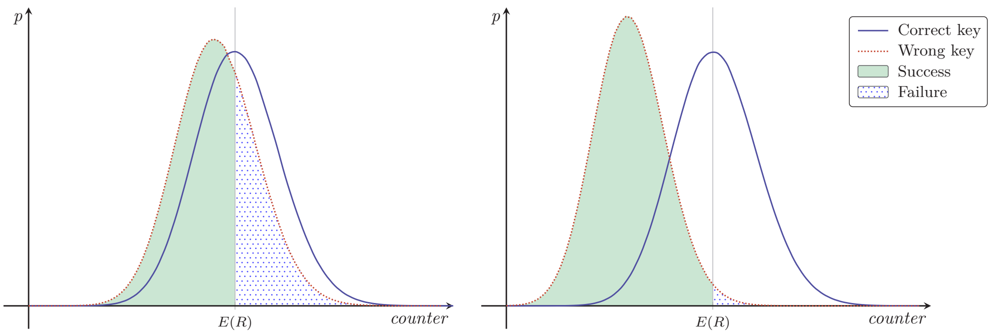


Figure 9: Modeling success probability. Let ψ be the probability that the counter for a fixed wrong key is smaller than $E(R)$. If ψ is not close to 1 (left), then its successive powers will quickly converge to zero. To ensure that the counters of many wrong keys are all smaller than $E(R)$ with a significant probability, ψ must be *very* close to 1 (right), i.e., the probability that the counter of a fixed wrong key is at least $E(R)$ must be negligible.

the expected value of the counter of the correct key $E(R)$ is the sum of the probabilities $\omega(a)$ for all $a < E(R)$. This sum is depicted as the shaded areas in Figure 9, while the dotted areas represent the failure probability. Since this must happen for all wrong keys, under the *Wrong Key Randomization Hypothesis* (WKRH) [HKM95], if the number of all possible keys is ν, the success probability is

$$\left(\sum_{a\leq E(R)} \omega(a)\right)^{\nu-1}, \tag{4}$$

i.e., the $(\nu-1)$-th power of the cumulative distribution function for the Poisson distribution at $E(R)$. To ensure a good success probability, the distributions for the counters of the correct key and of a wrong key must be clearly separated. To achieve this, we require that the difference between the expected counter values for the correct key and a wrong key be γ times the standard deviation of the counter of a wrong key. We then pick the smallest value of γ such that Equation (4) yields a 90% success probability under ideal distribution assumptions. When more than one characteristic is used in an attack, the product of their corresponding success probabilities should be at least 90%, so we have to aim at higher probabilities for the individual characteristics.

It is known that the WKRH is not always completely accurate [BT13], particularly for lightweight ciphers [ABR20]. Hence, after determining a value for N we experimentally verify the effectiveness of the characteristics — if computationally feasible (cf. Table 2).

We now describe the actual Poisson distribution argument. If we are recovering d round key bits from the front, we have $\nu = 2^d$ round key choices and, thus, counters.

Let N be the to-be-determined required number of pairs that satisfy the input difference Δ_{in}. We want to obtain a bound for N in terms of the statistical properties of the characteristic we are analysing. The counters of the wrong keys behave according to Poisson $\left(N \cdot 2^{-u+\widehat{d}_{out}}\right)$, since a random pair, even if not right, still passes the sieving test with probability $2^{-u+\widehat{d}_{out}}$. The counter of the correct key behaves like Poisson $\left(N \cdot \left(2^{-q} + 2^{-u+\widehat{d}_{out}}\right)\right)$ where $N \cdot 2^{-q}$ is the contribution from the right pairs. To ensure a significant separation between the distributions of the counter for the correct key and of the counters of the wrong keys, we require the difference between the means, $N \cdot 2^{-q}$, to be γ times the standard deviation $\sigma = \sqrt{N \cdot 2^{-u+\widehat{d}_{out}} \cdot \left(1 - 2^{-u+\widehat{d}_{out}}\right)}$ of the counter associated with a wrong key for some positive real number γ. Solving the inequality

$N \cdot 2^{-q} \geq \gamma \cdot \sigma$ for N, we obtain that N must be at least

$$N(\gamma) = \gamma^2 \cdot 2^{2q} \cdot 2^{-u+\widehat{d}_{out}} \cdot \left(1 - 2^{-u+\widehat{d}_{out}}\right) \quad .$$

Then, we apply Equation (4) to the distribution Poisson $\left(N(\gamma) \cdot 2^{-u+\widehat{d}_{out}}\right)$ and the expected value of the counter of the correct key, $E(R) = N(\gamma) \cdot \left(2^{-q} + 2^{-u+\widehat{d}_{out}}\right)$, with $\nu = 2^d$ to determine the success probability for the given γ.

If $\log_2 N \geq d_{in}$ we use multiple structures, otherwise we use partial structures. Out of the structures of Section 3.2 one can construct $2^{d_{in}} \cdot N$ pairs and therefore there are

$$L \approx 2^{d_{in}} \cdot N \cdot \left(2^{-u+\widehat{d}_{out}} + 2^{-(q+d_{in})}\right) = N \cdot \left(2^{d_{in}-u+\widehat{d}_{out}} + 2^{-q}\right) \quad . \tag{5}$$

sieved pairs. Note that if we fix some of the variable bits, the effective value of d_{in} needs to be reduced accordingly. The time and data complexities of Algorithm 1 are thus

$$\mathbb{D} = 2^{m+1} \quad \text{and} \quad \mathbb{T} = 2^{m+1} \cdot (T_{\copyright} + T_{MW} + T_{MR}) + L \cdot T_{GF}$$
$$\text{where} \quad m = \begin{cases} \log_2 N & \text{if } \log_2 N \geq d_{in} \text{ and} \\ \dfrac{\log_2 N + d_{in}}{2} & \text{if } \log_2 N < d_{in} \ . \end{cases} \tag{6}$$

Here, T_{MW}, resp., T_{MR} is the time required for a memory write, resp., read. $T_{\copyright}$ is the time required to query the SW oracle to obtain a chosen input encryption. The additional cost of $T_{\copyright}$ over that of an encryption, $T_{\mathcal{E}}$, is impossible to estimate in general, as it depends on the details of the SW attack. For simplicity, we shall assume that $T_{\copyright} \approx T_{\mathcal{E}}$.

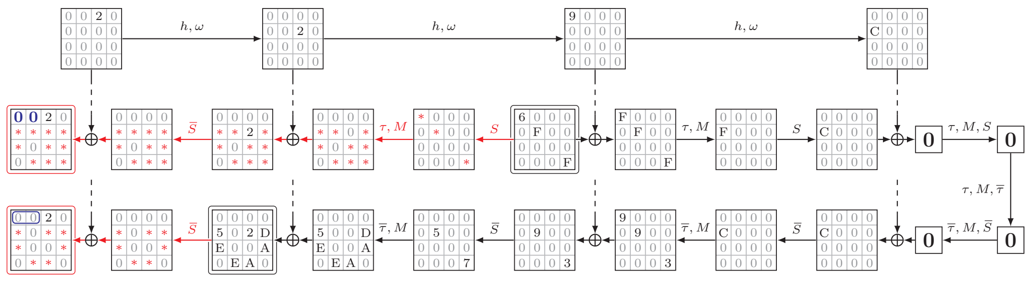
4 Finding Good Key Recovery Characteristics

In this section, we describe how we search for key recovery friendly differential characteristics for QARMA. This is a cipher-specific and challenging problem. For a different scenario, namely the search for key recovery friendly characteristics for GIFT, see [ZDC+21].

4.1 Choice of the Cost Function

It is well established that the time complexity of key recovery inversely correlates with the probability 2^{-p} of the characteristic, making it desirable to minimize p. However, a characteristic with minimal p but involving too many key bits may lead to a poor attack complexity [BS93]. Indeed, when the guess-and-filter phase dominates the cost of key recovery, time complexity is roughly proportional to L encryptions. From Equation (5) we have $L \approx \left(\gamma \cdot 2^{q-u+\widehat{d}_{out}}\right)^2 \cdot 2^{d_{in}}$, with a dependency on d_{in}, specifically a factor of $2^{d_{in}}$. To heuristically account for this effect, we also consider characteristics minimizing $p + d_{in}$.

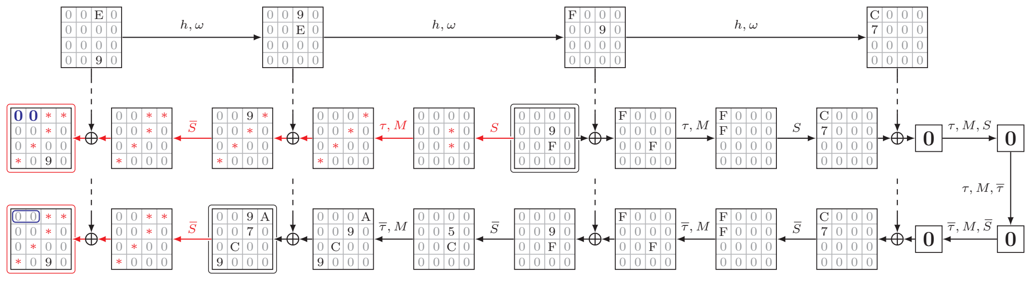
To better illustrate our approach, we discuss two characteristics. Each characteristic is constructed from a differential with two forward and three backward rounds, extended by 1.5 rounds at the front and 0.5 at the back. Only the two most significant nibbles of the output are visible and can thus be used as a distinguisher. The first characteristic has been found by maximizing its probability, which is $2^{-p} = 2^{-9}$.



Here, and in the following, we denote the undetermined differences by $*$, *the clamped plaintext cells by a boldface zero, and the visible ciphertext cells are grouped by a "lasso." A single large, boldface zero in a smaller square denotes an all-zero state in the center. The states with an additional border are* D_{in}, Δ_{in}, Δ_{out} *and* D_{out}.

A random pair passes the sieving step with probability 2^{-8}. By the Poisson distribution argument (Section 3.5) we need 2^{16} pairs satisfying Δ_{in}, with $\gamma = 8$, to get a success probability of 95 %. A partial structure with 2^{28} values per each half-structure (out of 2^{40}) produces the required 2^{16} pairs, with a data complexity of 2^{29}. After pair sieving, an average of 2^{48} sieved pairs remain. The time complexity is approximately $2^{49}\, T_{\mathcal{E}}$.

The second characteristic has been found by minimizing $p + d_{in}$. It has probability 2^{-14}, $d_{in} = 20$ and thus $p + d_{in} = 34$.



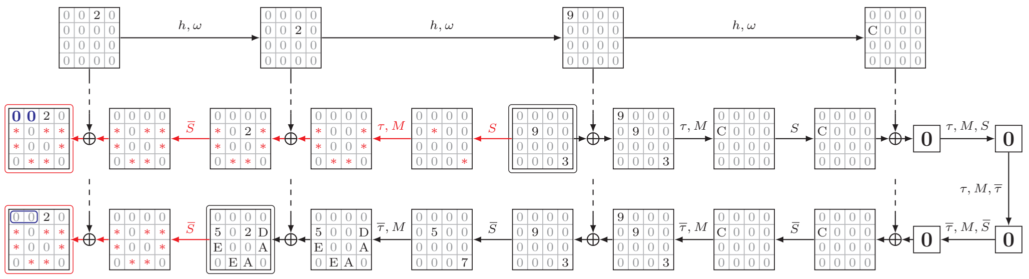
The Poisson distribution argument shows we need 2^{25} pairs satisfying Δ_{in}, with $\gamma = 5.35$, to attain a success probability of 95 %. We use 2^5 complete structures, each producing 2^{20} pairs satisfying Δ_{in}, for a data complexity of 2^{26}. The time complexity is $2^{37}\, T_{\mathcal{E}}$. Therefore, the second characteristic leads to a faster key recovery that also needs less data.

We now re-evaluate the two examples using the probability of the *differential*. For the first example, the probability of the differential $2^{-q} \approx 2^{-2}$ is significantly higher than the probability of the characteristic.[3] Thus, there is an attack with a data complexity of $2^{27.5}$, a time complexity of $2^{46}\, T_{\mathcal{E}}$, and a success probability of 95 %. For the second example, $q \leq 5$, for a data complexity of 2^{17}, a time complexity of $2^{25}\, T_{\mathcal{E}}$, and a success probability of 95 %. Notably, the second characteristic remains the more effective one. While this is usually to be expected, it is not necessarily always the case. Nonetheless, these examples indicate that minimizing $p + d_{in}$ is a source of useful characteristics.

Though the expression $L \approx \left(\gamma \cdot 2^{q-u+\widehat{d}_{out}}\right)^2 \cdot 2^{d_{in}}$ suggests the use of the cost function $p + \widehat{d}_{out} + d_{in}/2$ (note that d_{in} is always a multiple of 4), the latter did not lead to improved attacks, while considerably increasing search time.

[3] This bound, as well as the bounds for the other examples in this subsection, can be obtained by clustering (cf. Section 4.5) or by an argument similar to the one we use for Char. (13).

Expanding on the examples above, minimizing $p + \widehat{d}_{out} + d_{in}/2$ we find a characteristic with probability 2^{-10}, $d_{in} = 28$ and $\widehat{d}_{out} = 0$, for $p + \widehat{d}_{out} + d_{in}/2 = 24$.



We need $2^{17.60}$ pairs satisfying Δ_{in}, with $\gamma = 7$, to attain a success probability of 95 %. A partial structure with $2^{22.80}$ values in each half-structure produces the required $2^{17.60}$ pairs. With respect to the second characteristic we need less data — $2^{23.80}$ blocks instead of 2^{26} — but the running time is slightly higher — $2^{37.61}\,T_{\mathcal{E}}$ as opposed to $2^{37}\,T_{\mathcal{E}}$.

Using the probability of the differential 2^{-3}, i.e., $q = 3$, results in an attack with a data requirement of $2^{21.25}$ blocks and a time complexity of $2^{32.50}, T_{\mathcal{E}}$. This is notably less effective than the second characteristic. This observation holds generally and can be attributed to the presence of the term $\widehat{d}_{out}$ in the cost function. Reducing $\widehat{d}_{out}$ imposes a fixed difference on more output cells, limiting their ability to take any value; as a result, it restricts the number of characteristics within a cluster.

We observe that constraining $\widehat{d}_{out}$ leads to an increase in data requirements while the time complexity reduction does not significantly improve the time complexity of complete attacks, since they are primarily determined by the final brute-force steps.

Therefore, a more favorable cost function could be $p + d_{in}/2$. It quickly provides the second example, and also Char. (16) was found by minimizing this function.

4.2 Modelling the Search

In order to find good characteristics for key recovery, we model QARMA as a program for STP, the *Simple Theorem Prover* [GD07]. We extend Stefan Kölbl's cryptosmt [Köl14], which produces a *Satisfiability Modulo Theories* (SMT) program written in *Cooperating Validity Checker* (CVC) format (a language whose roots can be traced back to [SBD02]). STP uses then cryptominisat [SNC09] as the SAT solver. The steps are:

1. We program a bit-wise model of the r_{mid} middle rounds (always including also the central construction). This allows tighter bounds than the cell-wise MILP (Mixed-Integer Linear Programming) models used in the development of QARMA.

2. To determine D_{in} and $\widehat{D}_{out}$, and thus d_{in} and $\widehat{d}_{out}$, we add cell-wise models to trace the involved key cells through the first r_{in} rounds and the last round.

3. We add constraints on D_{in}, Δ_{in}, Δ_{out}, D_{out} and, if necessary, also relations involving arbitrary cell differences to the model. Such constraints are used to restrict a search, or to split a single search into more searches that can run in parallel.

4. Finally, we add one of the cost functions discussed in Section 4.1 to the model.

STP can find all characteristics with a given cost. We also use it to enumerate representatives of all subsets of a cluster defined by a common difference at a given state (e.g., the state after the first S-Box layer that follows Δ_{in}). The code is publicly available on GitHub[4].

[4]https://github.com/ShibamCrS/DifferentialAttackPACQARMA3.git

4.3 Impact of Input Clamping

The input to PAC functions is clamped. Without clamping, for each characteristic, usually additional ones exist with similar probabilities and related by various symmetries. Such symmetries can often be expressed as permutations of the cells, which can be iterated. In papers on unconstrained ciphers, authors often just need to mention that more key cells can be recovered at the same complexity by finding additional similar characteristics. However, input clamping eliminates most of these symmetries, necessitating an explicit search for additional characteristics.

4.4 Impact of Output Chopping

It is tempting to invoke some form of signal-to-noise-ratio argument to conclude that differential attacks are impossible when the probability of the differential is smaller than 2^{-u}, where u is the number of output bits. However, our analysis provides counterexamples, proving such an argument incorrect. Similar findings have been reported in the differential attacks on Feistel structures [Pat08] used to attack format preserving encryption schemes [HTT18, HMT19] and in linear cryptanalysis [ABD17].

4.5 Clustering

After a potentially good differential characteristic has been found, we can estimate its actual probability by identifying *clusters* for the given Δ_{in} and Δ_{out}.

In our SMT model we fix the input difference Δ_{in} and only the zero positions of the output difference Δ_{out}. This reflects the fact that pair sieving is based on the activeness of the output cells, not on their values, and suffices for key recovery only in the front rounds. The values of D_T are fixed. The number of solutions is determined using `cryptominisat`.

Suppose we have a characteristic $\Delta_{in} \mapsto \Delta_{out}$ with probability 2^{-p}. We count the characteristics $\Delta_{in} \mapsto \Delta_{out}$ with probability $2^{-p}, 2^{-(p+1)}, \ldots$ until a certain fixed limit is reached. The sum of the probabilities of all the found characteristics, denoted by 2^{-q}, is an estimate of the probability of the *differential* $\Delta_{in} \mapsto \Delta_{out}$. In Tables 2 and 3 we collect data on the effect of clustering analysis on the differentials used in our attacks.

4.6 Experimental Verification of Clustering Probabilities, and Conditional Characteristics

We experimentally verified the probabilities 2^{-q} computed by clustering by averaging results from 100 trials using random plaintexts, tweaks, and keys, obtaining a value $2^{-\bar{q}}$. A highly optimized implementation of `QARMA` is used in these experiments. The code for experimental verification is available on GitHub[5].

We now explain our experiments. If we use N pairs, we observe random noise of magnitude $\eta = N \cdot 2^{-u+\widehat{d}_{out}}$. The goal is to distinguish between two distributions, one with probability $\mu_1 = 2^{-q} + 2^{-u+\widehat{d}_{out}}$ and another with probability $\mu_2 = 2^{-u+\widehat{d}_{out}}$ and want to estimate q. Let $\overline{h}_i$ be the number of pairs that satisfy the truncated output difference $\widehat{D}_{out}$ in the i-th experiment. The statistics of $\overline{h}_i - \eta$ give us an estimate of q as $\bar{q} = -\log_2(\text{Mean}(\overline{h}_i - \eta)/N)$. These values are given in Table 2 for each characteristic.

We consider that a differential has sufficient signal if $\overline{h}_i - \eta$ is at least $\gamma/2$ units of the standard deviation of random noise. Thus, if $\overline{h}_i - \eta > (N \cdot 2^{-q})/2$, we consider the experiment to have been successful. We record the number of successful experiments in Table 2 together with $\bar{q}$ and the value of q obtained from clustering analysis.

[5]`https://github.com/ShibamCrS/DifferentialAttackPACQARMA3.git`

Table 2: Statistical properties of the characteristic used in our attacks.
The number of pairs N used in each experiment is the same as in the corresponding attack in Section 5. * marks the experiments restricted to tweaks fulfilling Relations (7) (resp. (8)) for Char. (13) (resp., Char. (15)). The experimentally obtained value of q, namely $\bar{q}$, is compared to the value obtained by clustering.

			Estimated				*Computed*				
							$\bar{h}_i - \eta$				Successes
z	Char.	$u - \hat{d}_{out}$	N	q	Chars. in Cluster	η S.D.	Mean ± S.D	Min	Max	$\bar{q}$	out of 100 trials
2	(9)	8	2^{14}	7.98	1543	7.98	64.11 ± 11.69	40	107	8.00	100
	(10)	8	2^{18}	10.01	474186	31.94	256.19 ± 32.40	186	346	10.00	100
	(11)	8	2^{20}	12.00	385024	63.87	257.50 ± 58.90	140	406	11.99	100
3	(11′)	12	2^{18}	12.01	384880	7.99	63.80 ± 11.50	34	96	12.00	100
	(13)	8	2^{22}	12.00	> 100 K	127.75	1022.33 ± 1832.27	−396	4739	12.00	27
	(13)*	8	2^{18}	10.00	—	31.93	258.32 ± 39.27	164	366	9.98	100
4	(13′)	8	2^{22}	12.00	> 100 K	127.75	1022.33 ± 1832.27	−396	4739	12.00	27
	(13′)*	8	2^{18}	10.00	—	31.93	258.32 ± 39.27	164	366	9.98	100
5	(15)	12	2^{29}	18.00	1512	361.99	2128.71 ± 2075.29	−614	5345	17.94	53
	(15)*	12	2^{27}	17.00	—	181.00	1026.25 ± 174.94	568	1527	17.00	100
	(16)	16	2^{30}	21.00	22440	128.00	505.58 ± 141.39	126	790	21.02	94
	(16)	16	2^{31}	21.00	22440	181.02	1028.42 ± 176.56	659	1472	20.99	100
6	(18)	4	2^{33}	18.00	> 350 K	5781.29	33324.88 ± 6337.25	20921	47369	17.97	100
7	(18′)	8	2^{33}	18.00	> 350 K	5781.29	31647.44 ± 5694.98	19383	43991	18.05	100

Looking at Table 2 we see that two characteristics — (13) (also (13′)) and (15) — have significantly lower success rates. We observe that, for these two characteristics, the probability is highly dependent on the tweak and key values — being higher than average for some combinations and zero for others. This also explains the high standard deviation in $\bar{h}_i - \eta$. Such characteristics are called *conditional characteristic* [BB93],

To understand this behavior, we investigate the dependency on the tweak and the key for both characteristics. For Char. (13), the dependency occurs in the third round. This part of the characteristic is depicted in Figure 10, where T'_i for $i = 0, 1$ is the value of the tweak in the third round in the first and the second element of a differential pair. The sets $\mathcal{A}$ and $\mathcal{B}$ of the valid S-box transitions in $\Delta_P \to \Delta_Q$ and $\Delta_Y \to \Delta_Z$, respectively, are:

$$\begin{aligned} \mathcal{A} &= \{x : S(x + \mathtt{4}) + S(x) = \mathtt{1}\} = \{\mathtt{1}, \mathtt{3}, \mathtt{5}, \mathtt{7}\} \\ \mathcal{B} &= \{x : S(x + \mathtt{4}) + S(x) = \mathtt{2}\} = \{\mathtt{10}, \mathtt{11}, \mathtt{14}, \mathtt{15}\} \ . \end{aligned}$$

Now we have the following relations

$$\begin{cases} X_i[0] = S(P_i[0]) + (T'_i + k + \mathfrak{c}_2)[0] \ , \qquad X_i[4] = S(P_i[10]) + (T'_i + k + \mathfrak{c}_2)[10] \ , \\ X_i[8] = S(P_i[5]) + (T'_i + k + \mathfrak{c}_2)[5] \ , \qquad X_i[12] = S(P_i[15]) + (T'_i + k + \mathfrak{c}_2)[15] \ , \\ Y_i[0] = \rho(X_i[4]) + \rho^2(X_i[8]) + \rho(X_i[12]) \\ \qquad = \rho^2\big((T'_i + k + \mathfrak{c}_2)[5]\big) + \rho\big((T'_i + k + \mathfrak{c}_2)[10]\big) + \rho\big((T'_i + k + \mathfrak{c}_2)[15]\big) + \\ \qquad\quad + \rho^2\big(S(P_i[5])\big) + \rho\big(S(P_i[10])\big) + \rho\big(S(P_i[15])\big) \ , \\ Y_i[8] = \rho^2(X_i[0]) + \rho(X_i[4]) + \rho(X_i[12]) \\ \qquad = \rho^2\big((T'_i + k + \mathfrak{c}_2)[0]\big) + \rho\big((T'_i + k + \mathfrak{c}_2)[10]\big) + \rho\big((T'_i + k + \mathfrak{c}_2)[15]\big) + \\ \qquad\quad + \rho^2\big(S(P_i[0])\big) + \rho\big(S(P_i[10])\big) + \rho\big(S(P_i[15])\big) \ . \end{cases}$$

To get a valid transition through the part of the characteristic depicted in Figure 10,

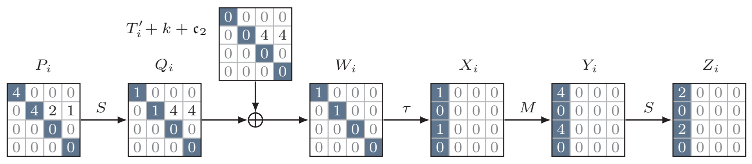


Figure 10: Example of tweak dependency (from (13)).

the following four conditions must be satisfied

$$P_i[0],\ P_i[5] \in \mathcal{A} \ \text{ and } \ Y_i[0],\ Y_i[8] \in \mathcal{B} \ .$$

This implies $\rho^2\big((T'_i + k + \mathfrak{c}_2)[5]\big) + \rho^2\big((T'_i + k + \mathfrak{c}_2)[0]\big) \in \{\mathtt{0}, \mathtt{1}, \mathtt{4}, \mathtt{5}\}$, or, equivalently $(T'_i + k + \mathfrak{c}_2)[5] + (T'_i + k + \mathfrak{c}_2)[0] \in \{\mathtt{0}, \mathtt{1}, \mathtt{4}, \mathtt{5}\}$. We get two linearly independent binary relations on a few bits of the tweak and the key:

$$\begin{cases} (T'_i + k + \mathfrak{c}_2)[5][1] + (T'_i + k + \mathfrak{c}_2)[0][1] = 0 \\ (T'_i + k + \mathfrak{c}_2)[5][3] + (T'_i + k + \mathfrak{c}_2)[0][3] = 0 \ . \end{cases}$$

However, the relations given above are on the third round tweaks T'_i. Inverting the tweak schedule, we obtain these two relations over the input tweaks T_i:

$$\begin{cases} B_0 := T_i[2][2] \qquad\qquad\qquad + T_i[5][2] \qquad\qquad\qquad = (k + \mathfrak{c}_2)[0][1] + (k + \mathfrak{c}_2)[5][1] \\ B_1 := T_i[2][0] + T_i[2][1] + T_i[5][0] + T_i[5][1] = (k + \mathfrak{c}_2)[0][3] + (k + \mathfrak{c}_2)[5][3] \ . \end{cases} \tag{7}$$

Since Cells 2 and 5 of the tweak are inactive, Relations (7) do not depend on i. For an attack to be feasible, the l.h.s. expressions in the tweak bits in Relations (7) must take specific values B_0 and B_1, which are determined by the key. This means that only 25 % of tweaks can be used to recover a given key, and a given tweak pair can aid in recovering only 25 % of all keys. However, when the conditions are met, the distinguisher probability increases fourfold to approximately 2^{-10}. Our experimental results in Table 2 confirm this.

Remark 4.1. (i) *The structures can be easily constructed such that the pairs are evenly split into four subsets according to the four values of* (B_0, B_1)*. Since only one subsets allows to counter of the correct key to stand out, testing the subset in succession reduces the average complexity by 37.5% versus the worst case. If multiple structures are used, then a quarter each of the structures will have a different value of* (B_0, B_1)*. If a single partial structure was meant to be used had the characteristic not been conditional, then four partial structures should be used instead, each one having half-structures of half-size.*

(ii) *Furthermore, this construction of the structures still allows the counter of the correct key to stand out also when all pairs are considered and all their suggestions counted.*

Remark 4.1.(ii) is crucial for the multiple differential attack for $z = 4$ (Section 5.3.1). In addition to Char. (13′), 43 more characteristics are used, displayed in Figures 12 and 13. Their statistical properties are collected in Appendix A. They are all conditional characteristics, with the conditions all using the same expressions B_i as defined in (7), but with different parities $(B_0, B_1) = (d_0, d_1) \in \mathbb{F}_2 \times \mathbb{F}_2$, as in

$$\begin{cases} B_0 := T_i[2][2] \qquad\qquad\qquad + T_i[5][2] \qquad\qquad\qquad = (k + \mathfrak{c}_2)[0][1] + (k + \mathfrak{c}_2)[5][1] + d_0 \\ B_1 := T_i[2][0] + T_i[2][1] + T_i[5][0] + T_i[5][1] = (k + \mathfrak{c}_2)[0][3] + (k + \mathfrak{c}_2)[5][3] + d_1 \ . \end{cases} \tag{7′}$$

Each characteristic is effective for only one of the four possible values (d_0, d_1) can take and, for a fixed key, all four possible values are taken by the characteristics. This implies that, for the attack to succeed, the pairs must be equally distributed among the four subsets characterised by (d_0, d_1).

Similarly, we establish that Char. (15) is also conditional, the dependency

$$\sum_{j\in\{0,2,3\}} \big(T_i[3]+T_i[15]\big)[j] = 1 + \sum_{j=0}^{3} \big((k+\mathfrak{c}_2)[7]+(k+\mathfrak{c}_2)[8]\big)[j] \tag{8}$$

being independent from i since $\Delta_T[3] = \Delta_T[15] = 0$. When this condition is met, the probability is 2^{-17}, otherwise it is 0. This has been experimentally verified, cf. Table 2.

Finally, we observe that some experiments for Equation (16) do not show a significantly strong signal. We could not identify any tweak or key dependency, and the absence of such a dependency is supported by the low standard deviation of $\overline{h}_i - \eta$. While some experiments had $\overline{h}_i - \eta$ lower than $(N \cdot 2^{-q})/2$ due to our choice of N, increasing N to 2^{31} from 2^{30} yields a success rate of 100. In contrast, for Chars. (13) and (15), increasing the data amount does not improve success rates because of the tweak dependencies discussed above.

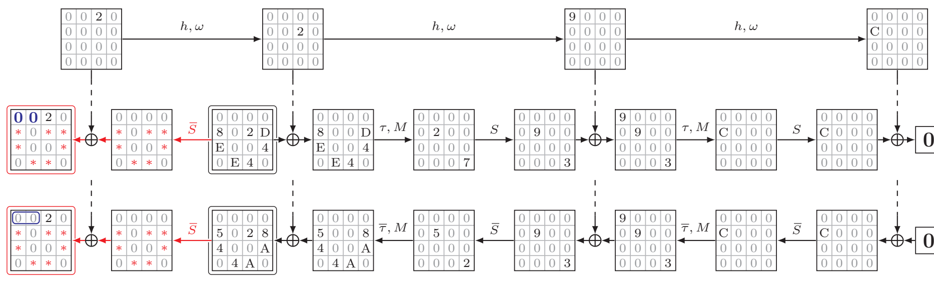
5 Cryptanalysis of the PAC Function in FEAT_PACQARMA3

We now describe the cryptanalysis of the PAC function for $z = 2$ through 8, with $z = v$, in full detail. Table 3 summarizes all attacks. For most characteristics we perform a 0.5R key recovery, followed by peeling to recover second round equivalent key cells $k' = M(\tau(k))$, guessing additional $k + w$ bits as needed. This is repeated for every characteristic in an attack, and any remaining bits are recovered through brute force. For $z = 2$ and 3, we also perform a 1.5R recovery, while for $z = 4$ we also mount a multiple differential attack.

5.1 Attack for $z = 2$ (56-bit Pointers)

Let us consider first the case of $z = 2$, i.e., the two most significant cells of the input are clamped, and only the two most significant cells of the output are revealed.

We first use Char. (9), with $p = 16$ and $d_{in} = 28$, in a 0.5R key recovery step, to recover the 28 bits $(k + w)[\{4, 6, 7, 8, 11, 13, 14\}]$. Clustering analysis (cf., Section 4.5 and experimental verification results in Table 2) show that the probability of the differential distinguisher is $2^{-7.98}$, i.e., $q = 7.98$.



(9)

We follow the Poisson distribution argument from Section 3.5 to determine how many pairs satisfying the input difference Δ_{in} are required in a key recovery step on Char. (9). As we can observe exactly two output cells, a random pair passes the sieving test with probability 2^{-8}. We have 2^{28} possible round key. With N pairs, the counters of the wrong keys follow Poisson$\big(N \cdot 2^{-8}\big)$ distribution. The counter for the correct key follows Poisson$(N \cdot (2^{-8} + 2^{-7.98}))$ distribution as $N \cdot 2^{-8}$ of the pairs pass the filter by random chance, and $N \cdot 2^{-7.98}$ are right pairs satisfying the characteristics. We find that the

Table 3: Summary of key recovery attacks on the PAC function for $2 \leq z \leq 8$.
For each characteristic, initial key cell recovery and peeling steps are separated by semi-colons. "M" denotes the only M-profile attack, which is for $z = 8$.
The success probability of each complete attack is at least 90 %.

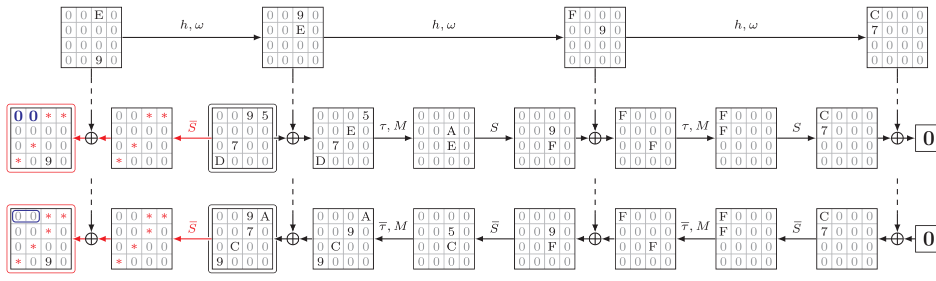
z	Char.	d_{in}	p	q	$\log_2 \mathbb{D}$	$\log_2 \mathbb{T}$	Recovered Round Key Cells
2	(9)	28	16	7.98	22	35	$(k+w)[\{4,6,7,8,11,13,14\}] \cup$ $\cup\, (k+w)[\{1,2\}] \cup k'[\{5,15\}]$.
	(10)	16	19	10.01	18	26	$(k+w)[\{3,9,12\}]$; $k'[\{6,10\}]$.
	(11)	28	22	12.00	24.5	39	$k'[2]$.
	Exhaustive Search				—	60	$(k+w)[\{0,5,10,15\}] \cup$ $\cup\, k'[\{0,1,3,4,7,8,9,11 — 14\}]$.
	Overall Complexity				25	60	
	Without using (11)				22.2	64	
3	(11′)	28	22	12.01	24	35	$(k+w)[\{3,4,7,9,11,12,14\}]$, $k'[\{2,5\}]$.
	(13)	40	24	12.00	24.5	38	$(k+w)[\{5,6,8,10,13,15\}]$; $k'[\{0,6\}]$; $(k+w)[2] \cup k'[7]$.
	Exhaustive Search				—	52	$(k+w)[\{0,1\}] \cup k'[\{1,3,4,8 — 15\}]$.
	Overall Complexity				25.3	52	
4	(13′)			12.00			$(k+w)[\{4,5,6,8 — 11,13,14,15\}]$;
	Fig. 12	40	24	13.00	28	54.5	$(k+w)[0] \cup k'[0,5]$;
	Fig. 13			14.00			$(k+w)[3] \cup k'[6]$; $(k+w)[2] \cup k'[7]$.
	Exhaustive Search				—	60	$(k+w)[\{1,7,12\}] \cup k'[\{1,2,3,4,8 — 15\}]$.
	Overall Complexity				28	60	
5	(15)	24	26	18.00	30	41	$(k+w)[\{5,7,8,10,13,15\}]$; $(k+w)[0] \cup k'[\{4,8\}]$; $(k+w)[2] \cup k'[\{3,7\}]$.
	(16)	16	25	20.80	30.35	31	$(k+w)[\{9,14\}]$; $k'[12]$.
	Exhaustive Search				—	68	$(k+w)[\{1,3,4,6,11,12\}] \cup$ $\cup\, k'[\{0,1,2,5,6,9,10,11,13,14,15\}]$.
	Overall Complexity				31.19	68	
6	(18)	20	30	18.00	37.50	54	$(k+w)[\{10,12,13,15\}]$; $(k+w)[\{0,5\}] \cup k'[\{4,12\}]$; $(k+w)[\{2,7,8\}] \cup k'[\{3,7\}]$.
	Exhaustive Search				—	76	$(k+w)[\{1,3,4,6,9,11,14\}] \cup$ $\cup\, k'[\{0,1,2,5 — 10,13,14,15\}]$.
	Overall Complexity				37.50	76	
7	(18′)	20	30	18.00	33.70	46	*Same as for* $z = 6$, *Char.* (18).
	Exhaustive Search				—	76	*Same as for* $z = 6$.
	Overall Complexity				33.70	76	
8	(19)	12	38	29.00	42	43	$(k+w)[\{9,11,13\}]$; $(k+w)[\{2,7,8\}] \cup k'[3]$; $(k+w)[\{1,4,14\}] \cup k'[\{5,9,13\}]$.
	Exhaustive Search				—	76	$(k+w)[\{0,3,5,6,10,12,15\}] \cup$ $\cup\, k'[\{0,1,2,4,6,7,8,10,11,12,14,15\}]$.
	Overall Complexity				42	76	
M	(19)	12	38	26.00	45	46.33	*Same as for* $z = 8$ *(A-profile), Char.* (19).
	Exhaustive Search				—	76	*Same as for* $z = 8$ *(A-profile), Char.* (19).
	Overall Complexity				45	76	

difference between the means must be at least $\gamma = 7$ times the standard deviation of the counter values associated with wrong keys, and that we need (at least) $N = 2^{14}$ pairs satisfying the input difference Δ_{in}, to achieve a success rate of 99 %. A partial structure of 2^{21} values out of 2^{28} for each of the two half-structures generates an expected 2^{14} such pairs, and the total data complexity is 2^{22} chosen plaintexts.

So far we have recovered the 28 key bits $(k + w)[\{4, 6, 7, 8, 11, 13, 14\}]$. We can recover more cells using the same sieved pairs from the previous step by peeling the first half round. To recover $k'[\{5, 15\}]$, we additionally need to guess $(k + w)[\{1, 2\}]$. Thus, we iterate again through all sieved pairs from the first step. This time, for each sieved pair that suggested the correct value of the 28 key bits, we guess $(k + w)[\{1, 2\}]$ and compute the values of $(Y_0[5], Y_1[5])$ and $(Y_0[15], Y_1[15])$ (i.e., the cell values before the second S-box layer according to the notation given in Figure 7). By enforcing the transitions $2 \mapsto 9$ and $7 \mapsto 3$, we obtain candidates for $k'[\{5, 15\}]$, and for each of them we tick the counter corresponding to $(k + w)[\{1, 2\}] \cup k'[\{5, 15\}]$.

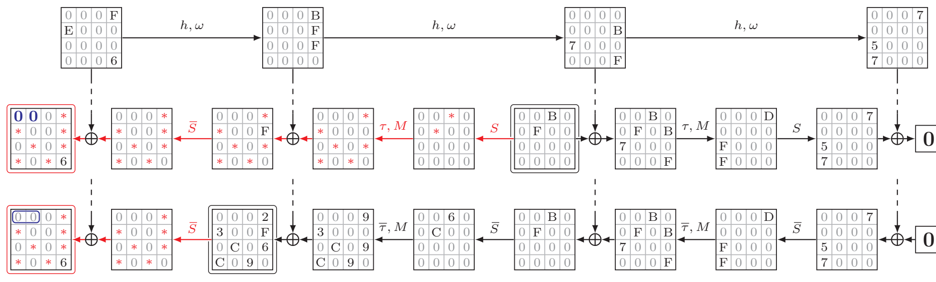
After pair sieving, we are left with $L \approx 2^{14} \cdot \left(2^{28-8} + 2^{-7.98}\right) = 2^{34}$ candidate pairs on average, as per Equation (5). For each such pair we perform a guess-and-filter step and of the peeling step, whose costs are bounded by the cost of a single QARMA encryption. So, the time complexity of this key recovery step is $\approx 2 \cdot 2^{34} = 2^{35}\, T_{\mathcal{E}}$.

We then use Char. (10) below, which has $p = 19$, $d_{in} = 16$ to recover the 12 key bits $(k + w)[\{3, 9, 12\}]$. For this characteristic, clustering analysis yields $q = 10.01$.


(10)

Reasoning as before, we need 2^{17} pairs with input difference Δ_{in}, and $\gamma = 5$, for a 95 % success rate. We use two structures, each providing 2^{16} such pairs out of 2^{32} total pairs, for a data complexity of 2^{18}. We then peel another round to recover $k'[\{6, 10\}]$, using the known $(k + w)[6]$ from earlier. This key recovery step has time complexity $2^{26}\, T_{\mathcal{E}}$.

Next, we use Char. (11) below, with $p = 22$, $d_{in} = 28$ and $q = 12.01$, to recover $k'[2]$ in a 1.5 R attack, with the knowledge of previously recovered key bits $(k + w)[\{3, 9, 12\}]$:


(11)

Since there are only 16 possible key values, we have $\nu = 4$ in (4). A Poisson distribution argument shows that $\gamma = 2.8$ is sufficient achieve a success rate of 96 %, using 2^{19} pairs with input difference Δ_{in}. To generate these 2^{19} pairs we use a partial structure of $2^{23.5}$ values out of 2^{28} per each half-structure, resulting in a data complexity of $2^{24.5}$. The time complexity of this key recovery step is $2^{39}\, T_{\mathcal{E}}$.

Lastly, we brute force the 60 remaining bits. The entire attack is graphically represented in Equation (12), where each cell of $(k+w)$ and k' is labeled with the number of the characteristic used to recover it or with b to indicate that its value is brute forced:

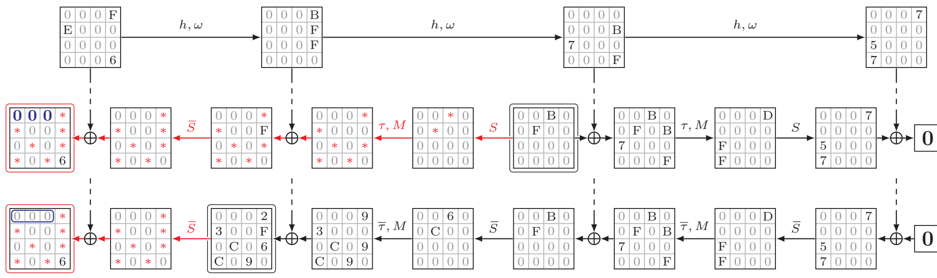
$$(k+w) \leftarrow \begin{bmatrix} b & (9) & (9) & (10) \\ (9) & b & (9) & (9) \\ (9) & (10) & b & (9) \\ (10) & (9) & (9) & b \end{bmatrix} \quad \text{and} \quad k' \leftarrow \begin{bmatrix} b & b & (11) & b \\ b & (9) & (10) & b \\ b & b & (10) & b \\ b & b & b & (9) \end{bmatrix}. \qquad (12)$$

The total data complexity of the attack is $\approx 2^{25}$ and the time complexity is dominated by the brute force step, i.e., $2^{60}\,T_{\mathcal{E}}$. The success probability is $99\,\% \cdot 95\,\% \cdot 96\,\% = 90\,\%$.

Remark 5.1. *Without using Char. (11), one can still obtain a valid attack with data complexity of $2^{22.2}$ and time complexity of $\mathbb{T} = 2^{64}$ encryptions.*

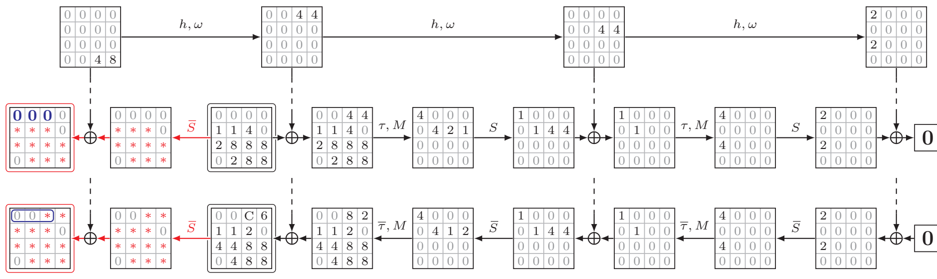
5.2 Attack for $z = 3$ (52-bit Pointers)

We adapt Char. (11) by clamping one additional plaintext cell and making one more ciphertext cell visible. This is possible since the first three plaintext cells of Char. (11) are zero differences. The resulting Char. (11′) has parameters $p = 22$, $d_{in} = 28$ and $q = 12$.


(11′)

We use it to recover $(k+w)[\{3,4,7,9,11,12,14\}]$ and $k'[\{2,5\}]$. By means of a Poisson distribution argument, we find that 2^{18} pairs satisfying the input difference Δ_{in} are required with $\gamma = 8$ to obtain a success rate of $99\,\%$. These pairs can be generated using a partial structure of 2^{23} values out of 2^{28} for each half-structure, for a data complexity of 2^{24}. The overall time complexity of this key recovery step is $2^{35}\,T_{\mathcal{E}}$.

To recover $(k+w)[\{5,6,8,10,13,15\}]$ we consider Char. (13) with $p = 24$.


(13)

We can establish that $q \le 12$ and that in fact q is very close to 12 as follows: First, we note that Char. (13) has a zero difference at its center. It is easy to prove that a zero difference at the center and Δ_T being active only in Cells 14 and 15 force the first two cells of D_{out} to be zero (cf. Figure 11). In other words, the transition probability from center to $\widehat{D}_{out}$ is 1. The path from Δ_{in} to the center is uniquely determined and has probability 2^{-12}. Hence, all characteristics matching the forward path up to the center have a cumulative

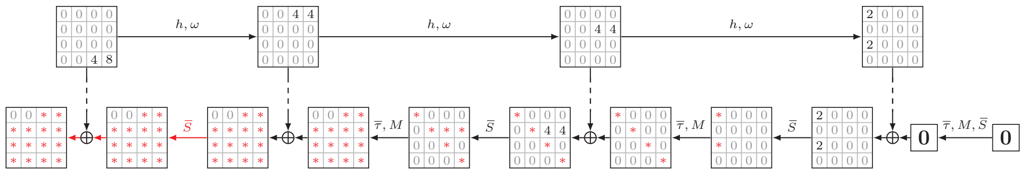


Figure 11: Diffusion from the center to the output in Char. (13) and similar ones.

probability of 2^{-12}. They belong to the cluster for Char. (13) and, while characteristics with non-zero center exist, we expect their contribution to be negligible. This is confirmed in the experiments reported in Section 4.6 (see Table 2), where we obtained $\bar{q} = -12.00$. For comparison, a clustering search only reached $q = 12.78$ in a week (during which it found 613768 characteristics), before we interrupted it.

By the Poisson distribution argument, we find that about $N = 2^{21}$ pairs satisfying Δ_{in} are needed, with $\gamma = 6$, for a success rate over 97 %. However, since this characteristic has 10 active cells in D_{in}, using it naïvely results in a data complexity of at least 2^{31}. To reduce the latter, we leverage the fact that the 16 key bits $(k+w)[\{4,9,11,14\}]$ have already been recovered using Char. (11′). With a negligible computation overhead, we can determine fixed differences at these positions in Δ_{in} such that, by varying the remaining 24 bits within a single structure, we generate plaintext pairs where one in 2^{24} satisfies the input difference Δ_{in}, instead of one in 2^{40}. (A similar approach was used in [BDD+23].)

Now, Char. (13) is conditional (cf. Section 4.6). To use it, we partition the $N = 2^{21}$ pairs satisfying Δ_{in} into four equal sets with the tweak bit expressions in Relations (7) taking a different value for each set. Instead of using a single partial structure with $2^{22.5}$ values out of 2^{24} for each half-structure, we build four partial structures with $2^{21.5}$ values for each half-structure, each structure generating 2^{19} pairs. The data complexity is $2^{24.5}$.

After the recovery of the 24 first round key bits, we perform peeling again to recover $k'[\{0,6\}]$ using the transitions $4 \mapsto 1$ and $2 \mapsto 4$ through the S-Boxes on Cells 0 and 6. Last, we recover $k'[7]$ by guessing $(k+w)[2]$ and using the transition $1 \mapsto 4$. The overall time complexity of this key recovery step is $2^{39.5}\, T_{\mathcal{E}}$.

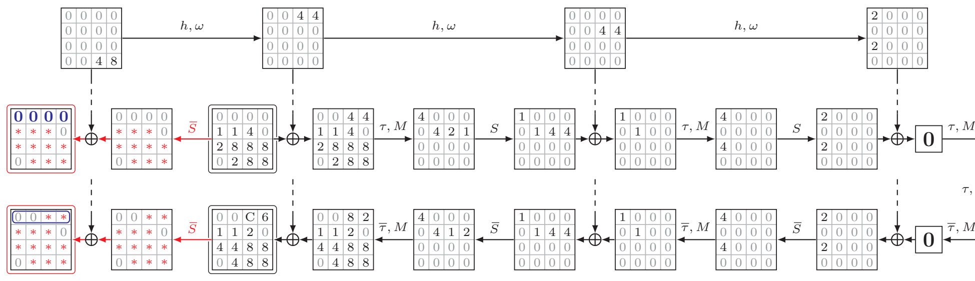
We brute force the remaining 13 cells. The complexity of the brute force step, $2^{52}\, T_{\mathcal{E}}$, dominates the time complexity. The total data complexity is $2^{25.3}$ chosen plaintexts, and the success rate is over 96 %. The complete attack is represented in Equation (14):

$$(k+w) \leftarrow \begin{bmatrix} b & b & (13) & (11') \\ (11') & (13) & (13) & (11') \\ (13) & (11') & (13) & (11') \\ (11') & (13) & (11') & (13) \end{bmatrix} \quad \text{and} \quad k' \leftarrow \begin{bmatrix} (13) & b & (11') & b \\ b & (11') & (13) & (13) \\ b & b & b & b \\ b & b & b & b \end{bmatrix} . \tag{14}$$

5.3 Attack for $z = 4$ (48-bit Pointers)

From this point onwards, if the recovery process is similar to previous cases, we provide only the used characteristics, key bits, and complexities.

We reuse Char. (13) with an additional clamped cell, as Char. (13′) below.


(13′)

It has $p = 24$ and $q \leq 12$ (by the same argument as in the previous attack). Here, we aim to recover first the 40 key bits $(w + k)[\{4, 5, 6, 8, 9, 10, 11, 13, 14, 15\}]$, and then $(w + k)[0] \cup k'[0, 5]$, $(w + k)[3] \cup k'[6]$, and $(w + k)[2] \cup k'[7]$ by peeling. We need $\gamma = 5.77$ to achieve a success rate of at least 90 %, using 2^{21} pairs with input difference Δ_{in}. Similarly to what we have done in the case $z = 3$, we generate these pairs by using four partial structures, each with $2^{29.5}$ values out of 2^{40} of each half-structure, for a total data complexity of $2^{32.5}$. The time complexity is $2^{54} T_{\mathcal{E}}$. We brute force the remaining 60 bits.

5.3.1 A Multiple Differential for $z = 4$

While the attack for $z = 4$ described above works, it exceeds our target limit of 2^{30} for the data complexity. To improve the latter, we propose a multiple differential attack [Knu94] for $z = 4$ using Char. (13′) together with similar ones. We start by searching for additional characteristics of lowest weight such that:

1. The have the same 10 cells active in Δ_{in} as (13′);
2. The have the same tweak difference;
3. The differences after the first S-Box layer of all characteristics are pairwise distinct;
4. The difference after the second S-Box layer is always equal to that in (13′); and
5. The first two cells at the ciphertext output have zero differences.

The third requirement prevents counting some characteristics multiple times when clustering each characteristic, and then building the multiple differential.

We can find 12 such characteristics with $p = 25$ and 31 with $p = 26$. The characteristics with $p = 25$, resp., $p = 26$, are given in Figure 12, resp., Figure 13. It suffices to show the beginning of their non-truncated section, as the rest is similar to Char. (13′).

For all the characteristics with $p = 25$, resp., $p = 26$, the probability of the differential from the input to the center (with a zero difference) is at least 2^{-13}, resp., 2^{-14}. This holds because the argument used for Char. (13) regarding the backward path having probability 1 clearly applies to all 44 characteristics. Furthermore, we have experimentally verified all the characteristics using the approach described in Section 4.6. For all the characteristics with $p = 25$, resp., $p = 26$, the experimental probability of the differential is approximately $2^{-12.9}$, resp., $2^{-13.7}$, which confirms the claim.

We combine these 44 characteristics in a multiple differential attack to recover the 40 key bits $(w + k)[\{4, 5, 6, 8, 9, 10, 11, 13, 14, 15\}]$. The counter associated with a key guess is a 44-dimensional vector with the coordinates corresponding to the 44 differentials. We assume that the coordinates are independent. Here we are interested in the distribution of the sum of these 44 values in each counter. Let us consider N pairs, each one satisfying one of the 44 input differences in Δ_{in}. Then, for each wrong key, the distribution of the counters

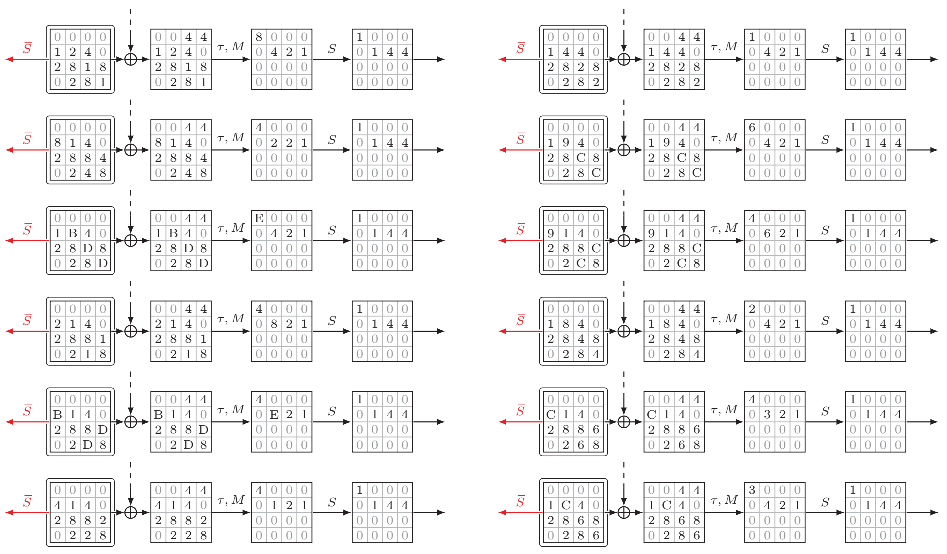


Figure 12: Start of the the 12 characteristics with $p = 25$.

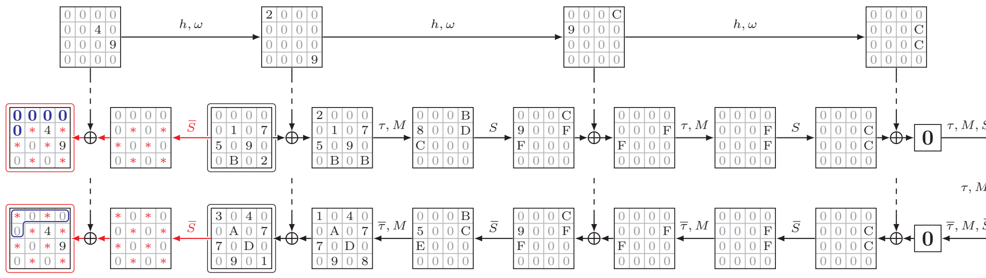
follows $\text{Poisson}(N \cdot 44 \cdot 2^{-8})$, and for the correct key, it follows $\text{Poisson}\big(N \cdot (2^{-8.12} + 44 \cdot 2^{-8})\big)$ where $2^{-8.12} = 2^{-12} + 12 \cdot 2^{-13} + 31 \cdot 2^{-14}$. By the Poisson distribution argument, we need $\gamma = 8.12$ to achieve a success probability higher than 90 %, corresponding to $N = 2^{19.5}$.

To take into account the fact that all 44 characteristics are conditional, we use four partial structures. In each structure, the tweak bits involved in Relations (7) satisfy a different value of $(B_0, B_1) \in F_2 \times \mathbb{F}_2$, and we pick 2^{26} values from each half-structure (out of 2^{40}), which allows it to generate $2^{17.5}$ pairs satisfying Δ_{in}. The data complexity is 2^{29}.

After that, we perform three peeling steps to recover, in order, $(w + k)[0] \cup k'[0, 5]$, $(w + k)[3] \cup k'[6]$, and $(w + k)[2] \cup k'[7]$. Here we make use of the fact that the difference after the second S-box layer is the same in all 44 characteristics. Thus, we can use the same data we have collected previously. The time complexity of this key recovery step is $4 \cdot 44 \cdot 2^{47}\, T_{\mathcal{E}} \approx 2^{54.5}\, T_{\mathcal{E}}$. We recover the remaining 60 bits by brute force.

5.4 Attack for $z = 5$ (44-bit Pointers)

First, we use Char. (15) with $p = 26$ and $d_{in} = 24$. Clustering analysis gives here $q = 18$. Using it, we recover $(k+w)[\{5, 7, 8, 10, 13, 15\}]$. We need 2^{29} pairs with input difference Δ_{in} and $\gamma = 5.8$ to achieve a 94 % success rate. The attack requires data complexity 2^{30} (using 32 structures) and time complexity $2^{41}\, T_{\mathcal{E}}$. As shown in Section 4.6, this characteristic is conditional with a single bit condition. We therefore split the 32 structures equally between tweaks where the expression in Equation (8) equals 0 and 1.



(15)

Then, using Char. (16), with $p = 25$, $d_{in} = 16$ and $q = 21.00$, we recover $(k+w)[\{9, 14\}]$,

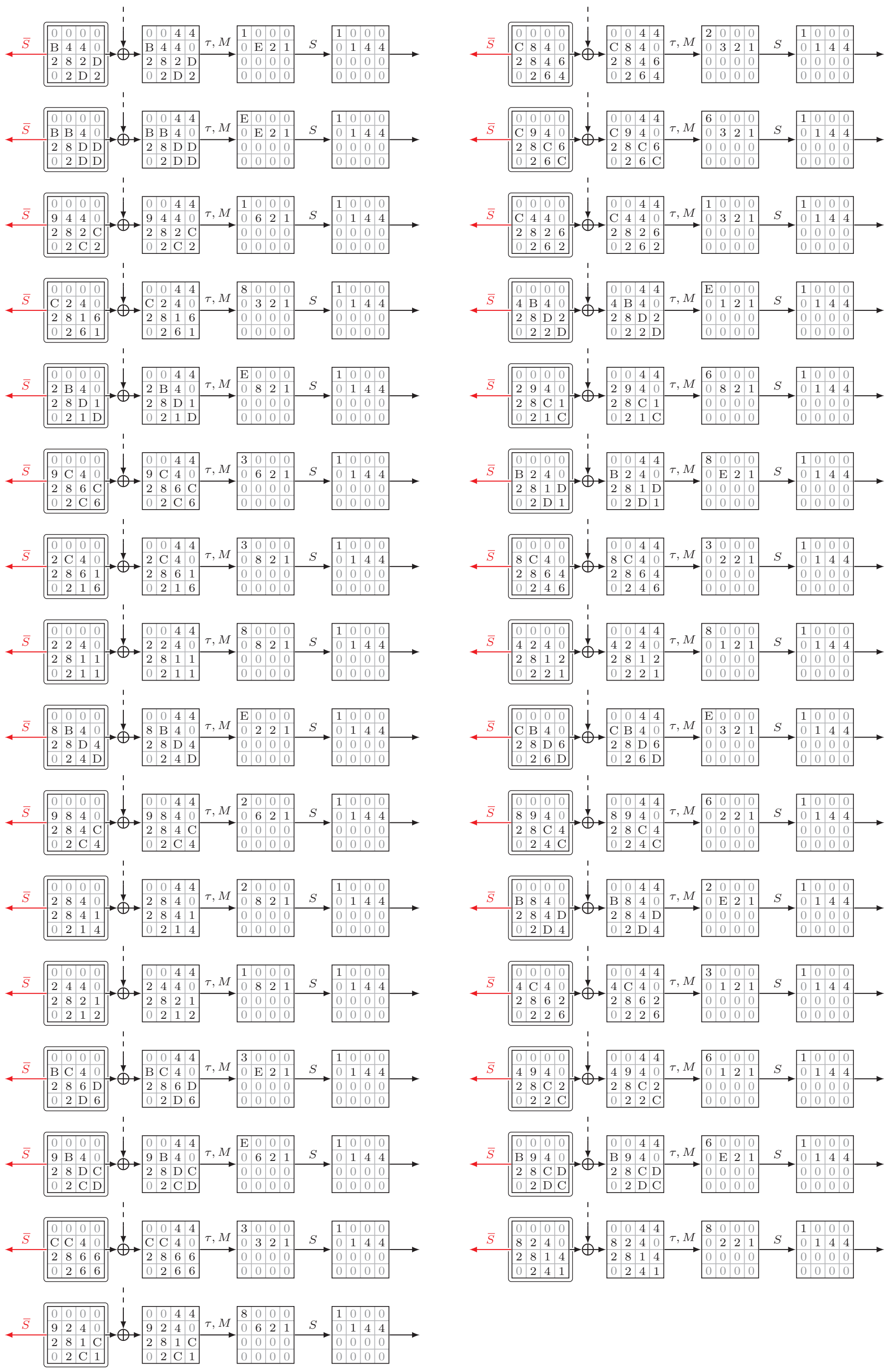
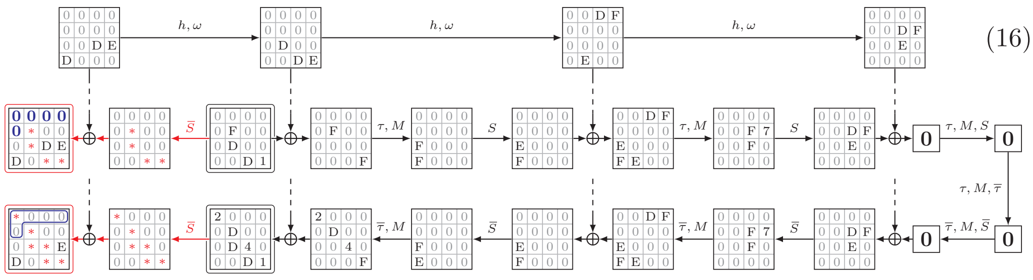


Figure 13: Start of the the 31 characteristics with $p = 26$.

and $k'[12]$. The data complexity of this key recovery step, with $\gamma = 3.4$, is $2^{30.53}$ and the time complexity is $2^{31.11}\, T_{\mathcal{E}}$ for a success rate of 91 %.


(16)

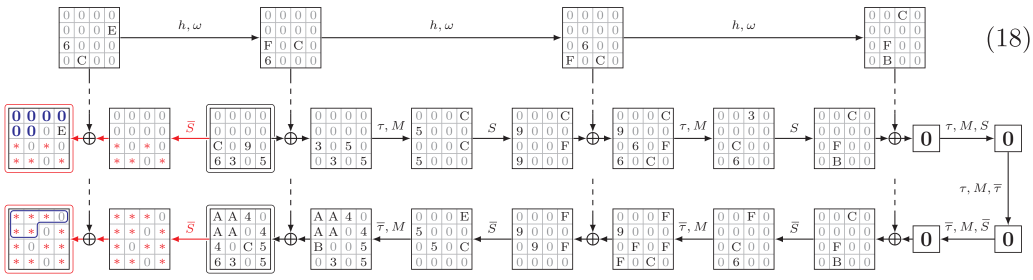
Incidentally, Char. (16) is found quickly by minimizing the cost function $p + d_{in}/2$.

Brute forcing the remaining 68 bits dominates the time complexity. The total data complexity is $2^{31.19}$. The entire attack is summarized as follows:

$$(k+w) \leftarrow \begin{bmatrix} (15) & b & (15) & b \\ b & (15) & b & (15) \\ (15) & (16) & (15) & b \\ b & (15) & (16) & (15) \end{bmatrix} \quad \text{and} \quad k' \leftarrow \begin{bmatrix} b & b & b & (15) \\ (15) & b & b & (15) \\ (15) & b & b & b \\ (16) & b & b & b \end{bmatrix} . \tag{17}$$

5.5 Attack for $z = 6$ (40-bit Pointers)

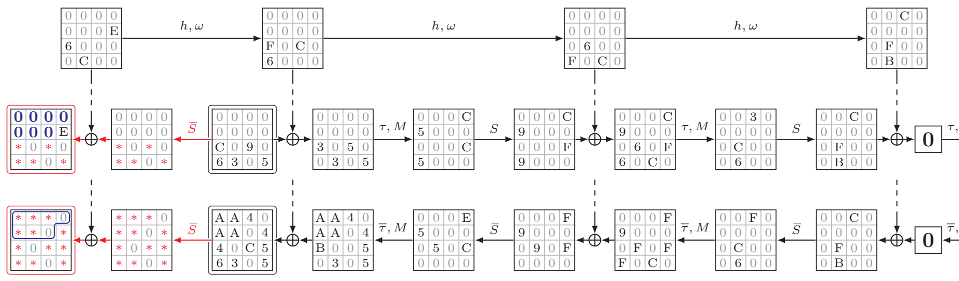
Consider Char. (18), with $p = 30$, $d_{in} = 20$ and clustering yields $q = 18$. We use it to recover $(k+w)[\{8, 10, 12, 13, 15\}]$, then $(k+w)[\{0, 5\}] \cup k'[\{4, 12\}]$, and finally $(k+w)[\{2, 7\}] \cup k'[\{3, 7\}]$ by successive peeling. The data complexity of this step is $2^{37.50}$, since $2^{36.50}$ pairs that satisfy the input difference Δ_{in} are required, with $\gamma = 5$, to achieve a success rate of 92 %. The time complexity of this step is approximately $2^{54}\, T_{\mathcal{E}}$.


(18)

We brute force the remaining 76 bits.

5.6 Attack for $z = 7$ (36-bit Pointers)

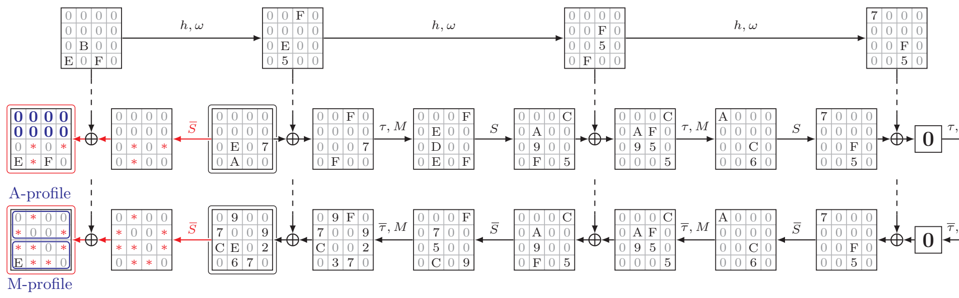
This attack is similar to the previous one. Clustering on Char. (18′) yields $q = 18$, i.e., the same as Char. (18). This can be explained by the easily proved fact that Cell 3 of the ciphertext output has a zero difference if and only if Cell 6 has a zero difference. We recover first $(k+w)[\{8, 10, 12, 13, 15\}]$ and then, by peeling, $(k+w)[\{0, 5\}] \cup k'[\{4, 12\}]$, and $(k+w)[\{2, 7\}] \cup k'[\{3, 7\}]$. Approximately $2^{32.7}$ pairs are required that satisfy the input difference Δ_{in}, considering $\gamma = 5.2$, to achieve a success rate of 90 %. The data complexity of this step is $2^{33.7}$, and its time complexity is approximately $2^{46}\,T_{\mathcal{E}}$.


(18′)

We brute force the remaining 76 bits.

5.7 The case of $z = 8$ (32-bit Pointers)

The best attack we found uses the following characteristic, with $p = 38$:


(19)

This characteristic has been found by searching for characteristics with the smallest p, for various values of the number of active cells of D_{in}.

For the A-profile variant, we reach $q = 29$ by clustering, using 1512 characteristics. We reach a success rate of 90 % in an attack requiring 2^{42} pairs satisfying Δ_{in}, with $\gamma = 4.2$. The data complexity is 2^{43}. We first recover $(k + w)[\{9, 11, 13\}]$. By peeling, we recover $k'[3]$ guessing $(k + w)[\{2, 7, 8\}]$, and recover $k'[\{5, 9, 13\}]$ guessing $(k + w)[\{1, 4, 14\}]$. The time complexity is $2^{43}\,T_{\mathcal{E}}$, mostly data collection (guess-and-filter costs about $2^{34}\,T_{\mathcal{E}}$).

For the M-profile variant, clustering returns exactly $q = 26$ with a total of 286272 characteristics. Taking $\gamma = 4$, we obtain an attack requiring 2^{44} pairs satisfying the input differential, for a data complexity of 2^{45} and a success rate of 90 %. The time complexity of guess-and-filter is $2^{44}\,T_{\mathcal{E}}$ for each of initial key recovery, two peeling steps, i.e. $2^{45.6}$, and including data collection we reach $2^{45} + 2^{45.6} = 2^{46.33}$.

We recover the same 13 round key cells for both versions and brute force 76 bits.

Because of the data requirements, both versions can be considered secure. However, the running time is lower than $2^{80}\,T_{\mathcal{E}}$ and the time/data product smaller than 2^{128}.

6 Conclusions

We have analysed the security of the clamped and chopped version of $\texttt{QARMA}_3$ used in `FEAT_PACQARMA3`. We found that while the versions with $z = 8$ are secure, and those with $z = 6$ and 7 provide adequate security, the *cryptographic strength* of variants more typical in commodity devices ($z = 3, 4,$ and 5) falls short of expectations.

In all cases, computational and memory requirements are non-trivial, and ongoing attacks could be detected by system-level anti-malware monitoring. Additionally, certain practical constraints complicate real-world attacks:

1. Code pointers on architectures with fixed size, 32-bit long instructions always have their two least significant bits equal to zero.
2. The activeness patterns of plaintext and tweak are very different in our characteristics, in contrast with the fact that they usually are pointers from close memory regions.
3. Bit 56 of the output is omitted and, if the *Memory Tagging Extension* (MTE) [Arm24] is enabled, the reserved bits of the pointer are shared between MTE and PAC.

Despite these constraints, security margins remain thinner than ideal. To hedge against potential advancements in cryptanalysis, we recommend to deploy $\texttt{QARMA}_5$ or $\texttt{QARMAv2}_4$ on high-performance, out-of-order cores, restricting the use of $\texttt{QARMA}_3$ to small 32-bit cores.

Acknowledgments

Part of this research was conducted while the first author was employed at Arm. The second and third authors express their gratitude to Arm for their support. The recommendations in this paper are the opinion of the authors only and do not necessarily represent the view of any former or present employer.

References

[AB12] Jean-Philippe Aumasson and Daniel J. Bernstein. Siphash: A fast short-input PRF. In Steven D. Galbraith and Mridul Nandi, editors, *Progress in Cryptology — INDOCRYPT 2012, 13th International Conference on Cryptology in India, Kolkata, India, December 9-12, 2012. Proceedings*, volume 7668 of *LNCS*, pages 489–508. Springer, 2012.

[ABD17] Tomer Ashur, Daniël Bodden, and Orr Dunkelman. Linear cryptanalysis using low-bias linear approximations. *IACR Cryptol. ePrint Arch.*, page 204, 2017.

[ABD+23] Roberto Avanzi, Subhadeep Banik, Orr Dunkelman, Maria Eichlseder, Shibam Ghosh, Marcel Nageler, and Francesco Regazzoni. The QARMAv2 family of tweakable block ciphers. *IACR Transactions on Symmetric Cryptology*, 3:25–73, 9 2023.

[ABR20] Tomer Ashur, Tim Beyne, and Vincent Rijmen. Revisiting the wrong-key-randomization hypothesis. *J. Cryptol.*, 33(2):567–594, 2020.

[ADG+19] Ralph Ankele, Christoph Dobraunig, Jian Guo, Eran Lambooij, Gregor Leander, and Yosuke Todo. Zero-Correlation attacks on tweakable block ciphers with linear tweakey expansion. *IACR Trans. Symmetric Cryptol.*, 2019(1):192–235, 2019.

[App24] Apple Inc. Operating system integrity: Pointer Authentication Codes, 2024.

[Arm16] Arm. ARM Connected blog. ARMv8-A architecture — 2016 additions. https://community.arm.com/arm-community-blogs/b/architectures-and-processors-blog/posts/armv8-a-architecture-2016-additions, 10 2016.

[Arm23] Arm. Arm Architecture Reference Manual for M-profile architecture, 2023.

[Arm24] Arm. Arm Architecture Reference Manual for A-profile architecture, 2024.

[Ava17] Roberto Avanzi. The QARMA block cipher family. almost MDS matrices over rings with zero divisors, nearly symmetric even-mansour constructions with non-involutory central rounds, and search heuristics for low-latency s-boxes. *IACR Trans. Symmetric Cryptol.*, 2017(1):4–44, 2017.

[AVL62] Georgy Maximovich Adelson-Velsky and Evgeny Mikhailovich Landis. An algorithm for the organization of information. *Soviet Mathematics Doklady*, 3:1259–1263, 1962.

[Bay72] Rudolf Bayer. Symmetric binary b-trees: Data structure and maintenance algorithms. *Acta Informatica*, 1:290–306, 1972.

[BB93] Ishai Ben-Aroya and Eli Biham. Differential cryptanalysis of lucifer. In Douglas R. Stinson, editor, *Advances in Cryptology - CRYPTO '93, 13th Annual International Cryptology Conference, Santa Barbara, California, USA, August 22-26, 1993, Proceedings*, volume 773 of *Lecture Notes in Computer Science*, pages 187–199. Springer, 1993.

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

[BCG+12] Julia Borghoff, Anne Canteaut, Tim Güneysu, Elif Bilge Kavun, Miroslav Knezevic, Lars R. Knudsen, Gregor Leander, Ventzislav Nikov, Christof Paar, Christian Rechberger, Peter Rombouts, Søren S. Thomsen, and Tolga Yalçin. PRINCE — A low-latency block cipher for pervasive computing applications — extended abstract. In Xiaoyun Wang and Kazue Sako, editors, *Advances in Cryptology — ASIACRYPT 2012 — 18th International Conference on the Theory and Application of Cryptology and Information Security, Beijing, China, December 2-6, 2012. Proceedings*, volume 7658 of *Lecture Notes in Computer Science*, pages 208–225. Springer, 2012.

[BDD+23] Christina Boura, Nicolas David, Patrick Derbez, Gregor Leander, and María Naya-Plasencia. Differential meet-in-the-middle cryptanalysis. In Helena Handschuh and Anna Lysyanskaya, editors, *Advances in Cryptology - CRYPTO 2023 - 43rd Annual International Cryptology Conference, CRYPTO 2023, Santa Barbara, CA, USA, August 20-24, 2023, Proceedings, Part III*, volume 14083 of *Lecture Notes in Computer Science*, pages 240–272. Springer, 2023.

[Bel73] James R. Bell. Threaded code. *Commun. ACM*, 16(6):370–372, 6 1973.

[BJK+16] Christof Beierle, Jérémy Jean, Stefan Kölbl, Gregor Leander, Amir Moradi, Thomas Peyrin, Yu Sasaki, Pascal Sasdrich, and Siang Meng Sim. The SKINNY family of block ciphers and its low-latency variant MANTIS. In Matthew Robshaw and Jonathan Katz, editors, *Advances in Cryptology — CRYPTO 2016 — 36th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 14-18, 2016, Proceedings, Part II*, volume 9815 of *LNCS*, pages 123–153. Springer, 2016.

[BKL+07] Andrey Bogdanov, Lars R. Knudsen, Gregor Leander, Christof Paar, Axel Poschmann, Matthew J. B. Robshaw, Yannick Seurin, and C. Vikkelsoe.

PRESENT: An ultra-lightweight block cipher. In Pascal Paillier and Ingrid Verbauwhede, editors, *Cryptographic Hardware and Embedded Systems — CHES 2007, 9th International Workshop, Vienna, Austria, September 10-13, 2007, Proceedings*, volume 4727 of *Lecture Notes in Computer Science*, pages 450–466. Springer, 2007.

[Blo24] Blockchain.com. Total hash rate, 2024.

[BS90] Eli Biham and Adi Shamir. Differential cryptanalysis of des-like cryptosystems. In Alfred Menezes and Scott A. Vanstone, editors, *Advances in Cryptology - CRYPTO '90, 10th Annual International Cryptology Conference, Santa Barbara, California, USA, August 11-15, 1990, Proceedings*, volume 537 of *Lecture Notes in Computer Science*, pages 2–21. Springer, 1990.

[BS91] Eli Biham and Adi Shamir. Differential cryptanalysis of des-like cryptosystems. *J. Cryptol.*, 4(1):3–72, 1991.

[BS92] Eli Biham and Adi Shamir. Differential cryptanalysis of the full 16-round DES. In Ernest F. Brickell, editor, *Advances in Cryptology - CRYPTO '92, 12th Annual International Cryptology Conference, Santa Barbara, California, USA, August 16-20, 1992, Proceedings*, volume 740 of *Lecture Notes in Computer Science*, pages 487–496. Springer, 1992.

[BS93] Eli Biham and Adi Shamir. *Differential Cryptanalysis of the Data Encryption Standard.* Springer, 1993.

[BSS+13] Ray Beaulieu, Douglas Shors, Jason Smith, Stefan Treatman-Clark, Bryan Weeks, and Louis Wingers. The simon and speck families of lightweight block ciphers. Cryptology ePrint Archive, Report 2013/404, 2013.

[BT13] Andrey Bogdanov and Elmar Tischhauser. On the wrong key randomisation and key equivalence hypotheses in matsui's algorithm 2. In Shiho Moriai, editor, *Fast Software Encryption - 20th International Workshop, FSE 2013, Singapore, March 11-13, 2013. Revised Selected Papers*, volume 8424 of *Lecture Notes in Computer Science*, pages 19–38. Springer, 2013.

[CDK09] Christophe De Cannière, Orr Dunkelman, and Miroslav Knezevic. KATAN and KTANTAN - A Family of Small and Efficient Hardware-Oriented Block Ciphers. In Christophe Clavier and Kris Gaj, editors, *CHES 2009*, volume 5747 of *Lecture Notes in Computer Science*, pages 272–288. Springer, 2009.

[CDMP05] Jean-Sébastien Coron, Yevgeniy Dodis, Cécile Malinaud, and Prashant Puniya. Merkle-damgård revisited: How to construct a hash function. In Victor Shoup, editor, *Advances in Cryptology - CRYPTO 2005: 25th Annual International Cryptology Conference, Santa Barbara, California, USA, August 14-18, 2005, Proceedings*, volume 3621 of *Lecture Notes in Computer Science*, pages 430–448. Springer, 2005.

[CXTQ23] Yaxin Cui, Hong Xu, Lin Tan, and Wenfeng Qi. Sat-aided differential cryptanalysis of lightweight block ciphers midori, MANTIS and QARMA. In Ding Wang, Moti Yung, Zheli Liu, and Xiaofeng Chen, editors, *Information and Communications Security - 25th International Conference, ICICS 2023, Tianjin, China, November 18-20, 2023, Proceedings*, volume 14252 of *Lecture Notes in Computer Science*, pages 3–18. Springer, 2023.

[DEKM16] Christoph Dobraunig, Maria Eichlseder, Daniel Kales, and Florian Mendel. Practical key-recovery attack on MANTIS5. *IACR Trans. Symmetric Cryptol.*, 2016(2):248–260, 2016.

[Des97] Solar Designer. Bugtraq: Getting around non-executable stack (and fix), 8 1997.

[FBP$^+$16] Andreas Follner, Alexandre Bartel, Hui Peng, Yu-Chen Chang, Kyriakos K. Ispoglou, Mathias Payer, and Eric Bodden. PSHAPE: automatically combining gadgets for arbitrary method execution. In Gilles Barthe, Evangelos P. Markatos, and Pierangela Samarati, editors, *Security and Trust Management - 12th International Workshop, STM 2016, Heraklion, Crete, Greece, September 26-27, 2016, Proceedings*, volume 9871 of *Lecture Notes in Computer Science*, pages 212–228. Springer, 2016.

[Fra18] Olivia Lucca Fraser. Urschleim in Silicon: Return Oriented Programming Evolution with ROPER. Technical report, Dalhousie University, Halifax, Nova Scotia, 12 2018.

[Fri56] Edward Harry Friend. Sorting on electronic computer systems. *Journal of the Association for Computing Machinery*, 3(3):134–168, 7 1956.

[FZHJ17] Olivia Lucca Fraser, Nur Zincir-Heywood, Malcolm I. Heywood, and John T. Jacobs. Return-oriented programme evolution with ROPER: a proof of concept. In Peter A. N. Bosman, editor, *Genetic and Evolutionary Computation Conference, Berlin, Germany, July 15-19, 2017, Companion Material Proceedings*, pages 1447–1454. ACM, 2017.

[GD07] Vijay Ganesh and David L. Dill. A decision procedure for bit-vectors and arrays. In Werner Damm and Holger Hermanns, editors, *Computer Aided Verification, 19th International Conference, CAV 2007, Berlin, Germany, July 3-7, 2007, Proceedings*, volume 4590 of *Lecture Notes in Computer Science*, pages 519–531. Springer, 2007.

[GG15] Shoni Gilboa and Shay Gueron. Distinguishing a truncated random permutation from a random function. *IACR Cryptol. ePrint Arch.*, page 773, 2015.

[GG21] Shoni Gilboa and Shay Gueron. The advantage of truncated permutations. *Discret. Appl. Math.*, 294:214–223, 2021.

[GGM18] Shoni Gilboa, Shay Gueron, and Ben Morris. How many queries are needed to distinguish a truncated random permutation from a random function? *J. Cryptol.*, 31(1):162–171, 2018.

[GNL11] Zheng Gong, Svetla Nikova, and Yee Wei Law. KLEIN: A new family of lightweight block ciphers. In Ari Juels and Christof Paar, editors, *RFID. Security and Privacy - 7th International Workshop, RFIDSec 2011, Amherst, USA, June 26-28, 2011, Revised Selected Papers*, volume 7055 of *Lecture Notes in Computer Science*, pages 1–18. Springer, 2011.

[GPPR11] Jian Guo, Thomas Peyrin, Axel Poschmann, and Matthew J. B. Robshaw. The LED block cipher. In Bart Preneel and Tsuyoshi Takagi, editors, *Cryptographic Hardware and Embedded Systems - CHES 2011 - 13th International Workshop, Nara, Japan, September 28 - October 1, 2011. Proceedings*, volume 6917 of *Lecture Notes in Computer Science*, pages 326–341. Springer, 2011.

[GVG$^+$17] Xinfei Guo, Vaibhav Verma, Patricia Gonzalez-Guerrero, Sergiu Mosanu, and Mircea R. Stan. Back to the future: Digital circuit design in the finfet era. *J. Low Power Electron.*, 13(3):338–355, 2017.

[HBK$^+$02] M. S. Hrishikesh, Doug Burger, Stephen W. Keckler, Premkishore Shivakumar, Norman P. Jouppi, and Keith I. Farkas. The optimal logic depth per pipeline stage is 6 to 8 FO4 inverter delays. In Yale N. Patt, Dirk Grunwald, and Kevin Skadron, editors, *29th International Symposium on Computer Architecture (ISCA 2002), 25-29 May 2002, Anchorage, AK, USA*, pages 14–24. IEEE Computer Society, 2002.

[HKM95] Carlo Harpes, Gerhard Kramer, and James L. Massey. A generalization of linear cryptanalysis and the applicability of matsui's piling-up lemma. In Louis C. Guillou and Jean-Jacques Quisquater, editors, *Advances in Cryptology - EUROCRYPT '95, International Conference on the Theory and Application of Cryptographic Techniques, Saint-Malo, France, May 21-25, 1995, Proceeding*, volume 921 of *Lecture Notes in Computer Science*, pages 24–38. Springer, 1995.

[HMT19] Viet Tung Hoang, David Miller, and Ni Trieu. Attacks only get better: How to break FF3 on large domains. In Yuval Ishai and Vincent Rijmen, editors, *Advances in Cryptology - EUROCRYPT 2019 - 38th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Darmstadt, Germany, May 19-23, 2019, Proceedings, Part II*, volume 11477 of *Lecture Notes in Computer Science*, pages 85–116. Springer, 2019.

[HTT18] Viet Tung Hoang, Stefano Tessaro, and Ni Trieu. The curse of small domains: New attacks on format-preserving encryption. In Hovav Shacham and Alexandra Boldyreva, editors, *Advances in Cryptology - CRYPTO 2018 - 38th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 19-23, 2018, Proceedings, Part I*, volume 10991 of *Lecture Notes in Computer Science*, pages 221–251. Springer, 2018.

[JNP14] Jérémy Jean, Ivica Nikolic, and Thomas Peyrin. Tweaks and Keys for Block Ciphers: The TWEAKEY Framework. In Palash Sarkar and Tetsu Iwata, editors, *Advances in Cryptology — ASIACRYPT 2014 — 20th International Conference on the Theory and Application of Cryptology and Information Security, Kaoshiung, Taiwan, R.O.C., December 7-11, 2014, Proceedings, Part II*, volume 8874 of *LNCS*, pages 274–288. Springer, 2014.

[Knu94] Lars R. Knudsen. Truncated and higher order differentials. In Bart Preneel, editor, *Fast Software Encryption: Second International Workshop. Leuven, Belgium, 14-16 December 1994, Proceedings*, volume 1008 of *Lecture Notes in Computer Science*, pages 196–211. Springer, 1994.

[Knu98] Donald Ervin Knuth. *The art of computer programming, Volume III: Sorting and Searching, 2nd Edition.* Addison-Wesley, 1998.

[Köl14] Stefan Kölbl. CryptoSMT: An easy to use tool for cryptanalysis of symmetric primitives, 2014. https://github.com/kste/cryptosmt.

[LHW19] Muzhou Li, Kai Hu, and Meiqin Wang. Related-tweak statistical saturation cryptanalysis and its application on QARMA. *IACR Trans. Symmetric Cryptol.*, 2019(1):236–263, 2019.

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

[LRW02] Moses Liskov, Ronald L. Rivest, and David Wagner. Tweakable block ciphers. In Moti Yung, editor, *Advances in Cryptology — CRYPTO 2002, 22nd Annual International Cryptology Conference, Santa Barbara, California, USA, August 18-22, 2002, Proceedings*, volume 2442 of *LNCS*, pages 31–46. Springer, 2002.

[LST12] Will Landecker, Thomas Shrimpton, and R. Seth Terashima. Tweakable Blockciphers with Beyond Birthday-Bound Security. In Reihaneh Safavi-Naini and Ran Canetti, editors, *Advances in Cryptology — CRYPTO 2012 — 32nd Annual Cryptology Conference, Santa Barbara, CA, USA, August 19-23, 2012. Proceedings*, volume 7417 of *LNCS*, pages 14–30. Springer, 2012.

[LV01] Arjen K. Lenstra and Eric R. Verheul. Selecting cryptographic key sizes. *J. Cryptol.*, 14(4):255–293, 2001.

[LZG$^+$20] Ya Liu, Tiande Zang, Dawu Gu, Fengyu Zhao, Wei Li, and Zhiqiang Liu. Improved cryptanalysis of reduced-version QARMA-64/128. *IEEE Access*, 8:8361–8370, 2020.

[NIS95] NIST. FIPS PUB 180-1 — Secure Hash Standard. Technical report, National Institute of Standards and Technology, Gaithersburg, MD, United States, April 1995.

[NIS98] NIST, May 1998.

[pak13] pakt. ropc — a Turing-complete ROP compiler, 2013.

[Pat08] Jacques Patarin. Generic Attacks on Feistel Schemes. *IACR Cryptol. ePrint Arch.*, page 36, 2008.

[PP20] Rajeev Kumar Pandey and Sanjeev Kumar Pandey. Analyzing the performance of 7nm finfet based logic circuit for the signal processing in neural network. In *2020 IEEE Recent Advances in Intelligent Computational Systems (RAICS)*, pages 136–140, 2020.

[QPS17] QPSI. Qualcomm Product Security. Pointer Authentication on ARMv8.3 — Design and Analysis of the New Software Security Instructions. https://www.qualcomm.com/documents/whitepaper-pointer-authentication-armv83, 1 2017.

[RBSS12] Ryan Roemer, Erik Buchanan, Hovav Shacham, and Stefan Savage. Return-oriented programming: Systems, languages, and applications. *ACM Trans. Inf. Syst. Secur.*, 15(1):2:1–2:34, 2012.

[Rog04] Phillip Rogaway. Efficient Instantiations of Tweakable Blockciphers and Refinements to Modes OCB and PMAC. In *Advances in Cryptology — ASIACRYPT 2004, 10th International Conference on the Theory and Application of Cryptology and Information Security, Jeju Island, Korea, December 5-9, 2004, Proceedings*, pages 16–31, 2004.

[SAB11] Edward J. Schwartz, Thanassis Avgerinos, and David Brumley. Q: exploit hardening made easy. In *20th USENIX Security Symposium, San Francisco, CA, USA, August 8-12, 2011, Proceedings*. USENIX Association, 2011.

[SBD02] Aaron Stump, Clark W. Barrett, and David L. Dill. CVC: A cooperating validity checker. In Ed Brinksma and Kim Guldstrand Larsen, editors, *Computer Aided Verification, 14th International Conference, CAV 2002,Copenhagen, Denmark, July 27-31, 2002, Proceedings*, volume 2404 of *Lecture Notes in Computer Science*, pages 500–504. Springer, 2002.

[Sha07] Hovav Shacham. The geometry of innocent flesh on the bone: return-into-libc without function calls (on the x86). In Peng Ning, Sabrina De Capitani di Vimercati, and Paul F. Syverson, editors, *Proceedings of the 2007 ACM Conference on Computer and Communications Security, CCS 2007, Alexandria, Virginia, USA, October 28-31, 2007*, pages 552–561. ACM, 2007.

[SII23] Kosei Sakamoto, Ryoma Ito, and Takanori Isobe. Parallel SAT framework to find clustering of differential characteristics and its applications. Cryptology ePrint Archive, Paper 2023/1227, 2023.

[SNC09] Mate Soos, Karsten Nohl, and Claude Castelluccia. Extending SAT solvers to cryptographic problems. In Oliver Kullmann, editor, *Theory and Applications of Satisfiability Testing - SAT 2009, 12th International Conference, SAT 2009, Swansea, UK, June 30 - July 3, 2009. Proceedings*, volume 5584 of *Lecture Notes in Computer Science*, pages 244–257. Springer, 2009.

[SSA$^+$07] Taizo Shirai, Kyoji Shibutani, Toru Akishita, Shiho Moriai, and Tetsu Iwata. The 128-Bit blockcipher CLEFIA. In Alex Biryukov, editor, *FSE 2007*, volume 4593 of *LNCS*, pages 181–195. Springer, 2007.

[ST85] Daniel Dominic Sleator and Robert Endre Tarjan. Self-adjusting binary search trees. *J. ACM*, 32(3):652–686, 1985.

[ST13] Thomas Shrimpton and R. Seth Terashima. A Modular Framework for Building Variable-Input-Length Tweakable Ciphers. In Kazue Sako and Palash Sarkar, editors, *Advances in Cryptology — ASIACRYPT 2013 — 19th International Conference on the Theory and Application of Cryptology and Information Security, Bengaluru, India, December 1-5, 2013, Proceedings, Part I*, volume 8269 of *LNCS*, pages 405–423. Springer, 2013.

[YQC18] Dong Yang, Wen-Feng Qi, and Hua-Jin Chen. Impossible differential attack on QARMA family of block ciphers. Cryptology ePrint Archive, Paper 2018/334, 2018. https://eprint.iacr.org/2018/334.

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

[ZD18] Rui Zong and Xiaoyang Dong. MILP-aided related-tweak/key impossible differential attack and its applications to QARMA, Joltik-BC. Cryptology ePrint Archive, Paper 2018/142, 2018. https://eprint.iacr.org/2018/142.

[ZD19] Rui Zong and Xiaoyang Dong. Milp-aided related-tweak/key impossible differential attack and its applications to qarma, joltik-bc. *IEEE Access*, 7:153683–153693, 2019.

[ZDC$^+$21] Rui Zong, Xiaoyang Dong, Huaifeng Chen, Yiyuan Luo, Si Wang, and Zheng Li. Towards key-recovery-attack friendly distinguishers: Application to GIFT-128. *IACR Trans. Symmetric Cryptol.*, 2021(1):156–184, 2021.

A Statistical Properties of the Multiple Differential used in the Attack on $z = 4$

In Tables 4 and 5 we show the statistical properties for the characteristics shown in Figures 12 and 13. In the first table we display the results with no constraints on the involved tweak bits, and in the second table the results with the constraints set to the indicated values of (d_0, d_1), i.e., the parities in Relations ($7'$).

Table 4: Statistical properties of the 43 characteristics from Figures 12 and 13 without restrictions on the tweak values.

The 12 characteristics from Figure 12: $N = 2^{24}$ pairs were used in each experiment.

Char.	*Estimated* η S.D.	*Computed* $\bar{h}_i - \eta$ Mean ± S.D	Min	Max	$\bar{q}$	Successes out of 100 trials
1	255.50	2085.58 ± 3615.61	−670	8751	12.97	26
2	255.50	2266.01 ± 3650.12	−451	8512	12.85	28
3	255.50	2369.74 ± 3820.03	−655	10826	12.79	33
4	255.50	1993.47 ± 3511.78	−700	8576	13.04	25
5	255.50	2165.11 ± 3724.98	−645	10359	12.92	28
6	255.50	2463.98 ± 3933.09	−676	10652	12.73	30
7	255.50	2434.28 ± 3911.55	−415	10487	12.75	29
8	255.50	1920.57 ± 3284.62	−503	8903	13.09	32
9	255.50	1994.94 ± 3398.59	−531	9023	13.04	32
10	255.50	2293.29 ± 3684.26	−473	8570	12.84	28
11	255.50	2440.46 ± 3836.79	−414	10527	12.75	32
12	255.50	2288.07 ± 3783.04	−673	10673	12.84	32

The 31 characteristics from Figure 13: $N = 2^{26}$ pairs were used in each experiment.

Char.	*Estimated* η S.D.	*Computed* $\bar{h}_i - \eta$ Mean ± S.D	Min	Max	$\bar{q}$	Successes out of 100 trials
1	511.00	5798.21 ± 8069.88	−1299	24959	13.50	33
2	511.00	5640.47 ± 9929.57	−1085	33788	13.54	31
3	511.00	4209.51 ± 6506.38	−1296	17398	13.96	32
4	511.00	4094.33 ± 7048.86	−1252	17159	14.00	25
5	511.00	4408.30 ± 6818.51	−777	17841	13.89	31
6	511.00	5069.72 ± 7001.92	−935	17287	13.69	36
7	511.00	3903.80 ± 6924.97	−1515	17376	14.07	24
8	511.00	6774.15 ± 12369.40	−1581	42109	13.27	31
9	511.00	4927.80 ± 7474.82	−871	17614	13.73	30
10	511.00	6633.73 ± 10880.50	−1201	25595	13.30	27
11	511.00	7330.09 ± 7933.66	−1294	25146	13.16	42
12	511.00	5502.13 ± 8254.59	−1307	24922	13.57	33
13	511.00	6316.38 ± 9064.93	−1201	25065	13.38	35
14	511.00	4066.96 ± 6295.16	−1230	16806	14.01	34
15	511.00	6032.48 ± 9243.16	−1186	33229	13.44	37
16	511.00	4092.05 ± 7116.01	−1225	17532	14.00	25
17	511.00	4475.24 ± 7876.58	−1203	24894	13.87	27
18	511.00	4534.33 ± 6575.20	−1079	17148	13.85	35
19	511.00	4129.26 ± 7557.05	−1347	25604	13.99	25
20	511.00	4732.90 ± 7566.95	−1166	24574	13.79	32
21	511.00	5011.33 ± 7583.58	−1446	17219	13.71	31
22	511.00	4941.77 ± 6975.90	−930	16978	13.73	37
23	511.00	6122.91 ± 10670.29	−1168	25513	13.42	25
24	511.00	5618.01 ± 8562.89	−1066	33500	13.54	36
25	511.00	4012.03 ± 7452.16	−1749	25390	14.03	25
26	511.00	6583.67 ± 10868.99	−1115	26046	13.32	27
27	511.00	4414.86 ± 7269.64	−1220	17261	13.89	27
28	511.00	5322.38 ± 8400.51	−1485	25166	13.62	31
29	511.00	4432.92 ± 6609.33	−1053	17107	13.89	34
30	511.00	3959.22 ± 6344.22	−812	17069	14.05	31
31	511.00	6913.14 ± 7769.96	−1602	25567	13.24	54

Table 5: Statistical properties of the 43 characteristics from Figures 12 and 13 when the tweak Conditions (7′) are met with the given values of (d_0, d_1).

The 12 characteristics from Figure 12: $N = 2^{20}$ pairs were used in each experiment.

Char.	*Estimated* η S.D.	*Computed* $\bar{h}_i - \eta$ Mean ± S.D	Min	Max	$\bar{q}$	Successes out of 100 trials	(d_0, d_1)
1	63.87	515.52 ± 62.94	369	679	10.99	100	$(0, 1)$
2	63.87	509.91 ± 66.30	383	665	11.01	100	$(1, 0)$
3	63.87	543.28 ± 84.07	342	719	10.91	100	$(1, 1)$
4	63.87	512.59 ± 78.59	352	783	11.00	100	$(0, 1)$
5	63.87	539.08 ± 78.20	358	760	10.93	100	$(1, 1)$
6	63.87	540.32 ± 92.68	309	838	10.92	100	$(1, 0)$
7	63.87	548.32 ± 90.16	393	760	10.90	100	$(1, 0)$
8	63.87	508.02 ± 69.37	316	677	11.01	100	$(1, 1)$
9	63.87	509.21 ± 64.34	370	718	11.01	100	$(1, 1)$
10	63.87	507.80 ± 65.95	374	676	11.01	100	$(1, 0)$
11	63.87	538.37 ± 76.19	376	702	10.93	100	$(0, 1)$
12	63.87	551.91 ± 82.18	378	805	10.89	100	$(0, 1)$

The 31 characteristics from Figure 13: $N = 2^{22}$ pairs were used in each experiment.

Char.	*Estimated* η S.D.	*Computed* $\bar{h}_i - \eta$ Mean ± S.D	Min	Max	$\bar{q}$	Successes out of 100 trials	(d_0, d_1)
1	127.75	1154.17 ± 262.88	753	1741	11.83	100	$(0, 1)$
2	127.75	1261.24 ± 442.28	745	2296	11.70	100	$(0, 0)$
3	127.75	1037.21 ± 141.04	690	1482	11.98	100	$(0, 1)$
4	127.75	1026.67 ± 139.99	715	1398	12.00	100	$(0, 0)$
5	127.75	1028.04 ± 135.39	654	1314	11.99	100	$(1, 0)$
6	127.75	1010.56 ± 138.67	610	1396	12.02	100	$(1, 0)$
7	127.75	1024.98 ± 130.43	805	1385	12.00	100	$(0, 0)$
8	127.75	1475.22 ± 613.13	754	2765	11.47	100	$(0, 0)$
9	127.75	1025.01 ± 123.17	706	1340	12.00	100	$(0, 1)$
10	127.75	1526.31 ± 143.34	1124	1845	11.42	100	$(0, 1)$
11	127.75	1148.12 ± 254.34	744	1768	11.83	100	$(1, 1)$
12	127.75	1370.89 ± 264.60	730	1812	11.58	100	$(1, 1)$
13	127.75	1138.22 ± 228.49	769	1672	11.85	100	$(1, 0)$
14	127.75	1028.94 ± 129.78	797	1443	11.99	100	$(0, 0)$
15	127.75	1307.01 ± 486.10	758	2342	11.65	100	$(0, 0)$
16	127.75	1014.07 ± 140.57	704	1394	12.01	100	$(1, 0)$
17	127.75	1122.94 ± 236.16	725	1780	11.87	100	$(1, 1)$
18	127.75	1021.63 ± 117.58	735	1402	12.00	100	$(1, 0)$
19	127.75	1138.60 ± 246.36	679	1876	11.85	100	$(1, 1)$
20	127.75	1165.59 ± 258.04	659	1891	11.81	100	$(0, 1)$
21	127.75	1018.88 ± 136.51	759	1452	12.01	100	$(1, 0)$
22	127.75	1022.58 ± 133.59	611	1387	12.00	100	$(1, 0)$
23	127.75	1140.75 ± 259.66	701	1689	11.84	100	$(1, 1)$
24	127.75	1452.98 ± 248.09	790	1824	11.50	100	$(1, 1)$
25	127.75	1197.76 ± 275.43	754	1812	11.77	100	$(1, 0)$
26	127.75	1533.00 ± 118.06	1295	1887	11.42	100	$(0, 1)$
27	127.75	1034.45 ± 114.33	708	1300	11.99	100	$(0, 1)$
28	127.75	1165.27 ± 246.96	671	1701	11.81	100	$(1, 1)$
29	127.75	1036.41 ± 121.61	721	1369	11.98	100	$(0, 1)$
30	127.75	1013.12 ± 106.29	742	1263	12.02	100	$(0, 0)$
31	127.75	1129.12 ± 240.12	699	1737	11.86	100	$(1, 1)$