BEANIE – A 32-bit Cipher for Cryptographic Mitigations against Software Attacks

Simon Gerhalter[1], Samir Hodžić[2], Marcel Medwed[2], Marcel Nageler[1], Artur Folwarczny[3], Ventzi Nikov[3], Jan Hoogerbrugge[4], Tobias Schneider[2], Gary McConville[2] and Maria Eichlseder[1]

[1] Graz University of Technology, Graz, Austria
firstname.lastname@tugraz.at
[2] NXP Semiconductors, Gratkorn, Austria
[3] NXP Semiconductors, Leuven, Belgium
[4] NXP Semiconductors, Eindhoven, Netherlands
firstname.lastname@nxp.com

Abstract. In modern CPU architectures, various security features to mitigate software attacks can be found. Examples of such features are logical isolation, memory tagging or shadow stacks. Basing such features on cryptographic isolation instead of logical checks can have many advantages such as lower memory overhead and more robustness against misconfiguration or low-cost physical attacks. The disadvantage of such an approach is however that the cipher that has to be introduced has a severe impact on the system performance, either in terms of additional cycles or a decrease of the maximum achievable frequency. Finally, as of today, there is no suitable low-latency cipher design available for encrypting 32-bit words as is common in microcontrollers. In this paper, we propose a 32-bit tweakable block cipher tailored to memory encryption for microcontroller units. We optimize this cipher for low latency, which we achieve by a careful selection of components for the round function and leveraging an attack scenario similar to the one used to analyze the cipher SCARF. To mitigate some attack vectors introduced by this attack scenario, we deploy a complex tweak-key schedule. Due to the shortage of suitable 32-bit designs, we compare our design to various low-latency ciphers with different block sizes. Our hardware implementation shows competitive latency numbers.

Keywords: Low-latency block cipher · Memory encryption

1 Introduction

While the Advanced Encryption Standard (AES) has been thoroughly analyzed throughout the years, there are many applications where its comparatively heavy implementation hinders its adoption. To address this issue, various lightweight ciphers with low latency, small area, or low power consumption have been proposed over the last decade. Prominent examples of low-latency ciphers are PRINCE [BCG+12], PRINCEv2 [BEK+20], and the tweakable block cipher QARMAv2 [ABD+23]. One thing these ciphers have in common is the rather large state size of 64 or 128 bits. Recently, new low-latency designs for specific applications with uncommon block sizes have been introduced. The 24-bit block cipher BipBip [BDD+23a] aims to prevent types of software vulnerabilities by encrypting part of the pointers. An even smaller 10-bit state size is used by the cipher SCARF [CGL+23a] to achieve cryptographically secure cache index randomization. Another interesting detail of this cipher and its use case is the unique attacker model. Here, the attacker can not directly access the ciphertext. Instead, it is only possible to detect if two plaintexts encrypted

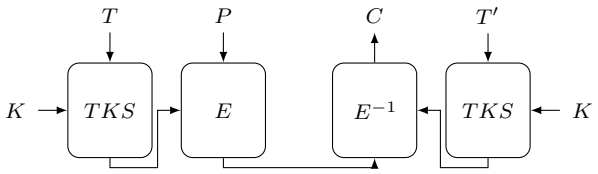


Figure 1: The U-shape attack setting.

with different tweaks lead to the same ciphertext. For the designers, that means that they only need to consider the latency of the encryption, while an attack needs to break an encryption followed by a decryption. While simply halving the number of rounds of a secure cipher may have serious security implications, we can still drastically reduce the number of rounds by conducting careful cryptanalysis.

Encrypting data is not only important for internet communication or encrypting files on personal computers, but can also play a vital part in securing programs running on a microcontroller unit (MCU). These MCUs (e.g. the Cortex-M family by ARM or the RISC-V RV32-based architectures) typically have 32-bit data words and addresses used for their internal memories such as ROM, non-volatile memory, and RAM. When designing hardware-based encryption for these memory types, we have to account for different types of attacks. In case the attacker can mount physical attacks using specialized equipment, it may be possible to read out the encrypted ciphertext stored in the memory. However, in case the attacker can only mount software attacks (potentially in combination with low-cost phyiscal attacks to overcome isolation [NSUH22]), the attacker is only able to decrypt the ciphertext with a different tweak and use this information to mount an attack. More precisely (cf. Section 5 for more details), for a plaintext P and its corresponding ciphertext $E_{K,T}(P) = \overline{C}$ (with a correct tweak T and main key K), the attacker may query the decryption of $\overline{C}$ (not visible to the attacker) with a different tweak $T' \neq T$ in order to obtain an input-output pair (P, C) given by $P \to E_{K,T}(P) = \overline{C} \to E^{-1}_{K,T'}(\overline{C}) = C$. Similarly as in SCARF, we attack the mapping $U = E^{-1}_{K,T'} \circ E_{K,T}$, which produces (P, C) pairs $U(P) = E^{-1}_{K,T'}(E_{K,T}(P)) = C$. We call this setting *U-shape attack* (Figure 1).

Contribution. In this paper, we present a new 32-bit low-latency cipher designed for hardware-based memory encryption on MCUs. By leveraging the limited power of the attacker and a careful selection of components, we are able to achieve compelling latency numbers. In more detail, our contributions include the following.

- *New cipher design with strong non-linear tweak-key schedule.* We first motivate the need for a new cipher design (called BEANIE) having a unique interface and facing an uncommon attack setting(s). In order to prevent easily inducible collisions in the round keys, we employ a complex non-linear tweak-key schedule. This is necessary, since colliding subkeys can lead to round cancellations in the U-shape attacker model. Compared to the other small blocksize ciphers SCARF and BipBip, we do not employ an iterative tweak-key schedule, where each round a function updates the tweak. Instead, we use a block cipher based on the round function of PRINCE that is optimized to generate a high number of differentially active S-Boxes. While this tweak-key schedule only slightly increases the complexity of the key recovery in the U-shape attacker model, we argue that it has substantial security benefits in the normal attack setting. Thus, the tweak-key should mainly prevent inducing collisions in the subkeys with a below-generic complexity and provide some additional resistance against certain attacks on the cipher.

- *In-depth analysis.* Due to the tight security margin, we conducted an in-depth

cryptanalysis. Using more traditional building blocks for the data path compared to SCARF provides confidence when using this cipher in a normal attack setting. However, in conjunction with the tweak-key schedule and the U-shape attack setting, a variety of new attack vectors have to be considered. We provide arguments against generic round-key collision attacks. Furthermore, for various types of cryptanalysis, we build tools aiding us in finding the best distinguishers. This was necessary since the transition from encryption to decryption in the U-shape attack setting allows for a multitude of different distinguishers.

- *Performance benchmarking.* We provide hardware performance benchmarks showing the latency and area usage of the data path as well as the tweak-key schedule. An important feature of the whole design is the possibility to implement the cipher and tweak-key schedule (TKS) in two different cycles and thus splitting the latency, while introducing stall cycles only in case of writes but not reads (see Section 2.2). Note that reads are more frequent memory operations. In this context, running the TKS and data path of the cipher "in parallel", allows us to obtain competitive latency numbers. Additionally, in the same section, we give an evaluation of the design space covering cipher variants with different components. Namely, we display the performance impact of different classes of S-Boxes and other MDS matrices, as well as non-MDS lightweight linear layers. This allows us to make an informed decision when weighing the performance impact of a component against its security benefit.

We provide reference implementations and code for cryptanalysis in https://github.com/isec-tugraz/beanie_cipher and the hardware implementation in https://github.com/NXP-Research/beanie_cipher.

Related work. In recent years, low-latency ciphers have become a very active field of research in symmetric cryptography. A first wave of designs focused on low-latency block ciphers as a replacement for standard or lightweight block ciphers, with classical block sizes of 64 to 128 bits. A main envisioned area of application is the encryption of memory with plain (or sometimes authenticated) encryption schemes, but also general applications that profit from low-latency response times. The pioneering low-latency block cipher PRINCEv1 [BCG+12] inspired several followup designs, including PRINCEv2 [BEK+20], QARMA [Ava17], and QARMAv2 [ABD+23], with improvements such as a more standard security claim (instead of the time-data-tradeoff of PRINCE), a tweak, and a larger 128-bit block size. Other designs in this category include SPEEDY [LMMR21] (with a larger block size and gate-level-optimized building blocks), MATTER [ADM24], ARADI [GMW24], and uKNIT [HKPT24] (with a round-by-round-optimized unrolled design). While several of these designs target memory encryption, none focus on the 32-bit word sizes common in microcontrollers. An exception is the very recent ultra-low-latency ChiLow family [BDG+25] focusing on code encryption. This cipher is mainly optimizing the decryption latency, while for our design, a low encryption latency is also vital. Further, its security is limited to a tiny number (≤ 256) of queries per tweak due to the specific target setting.

A second interesting direction in low-latency cryptography is the design of non-invertible block functions, i.e., fixed-size pseudorandom functions (PRFs) instead of pseudorandom permutations (PRPs). Besides allowing improved security bounds in some modes of operation, recent research suggests that it may be easier to design secure non-invertible functions with low latency rather than invertible ones. Examples include ORTHROS [BIL+21], GLEEOK [ABC+24], TWINKLE [WHWL24], and ZIP-AES [FGL+24] (or zip-ciphers as a general approach). KOALA [AEBD+24] is an example of a variable-input-length PRF based on KOALA-p, a low-latency permutation.

Finally, several authors have discussed the use of low-latency ciphers for protecting memory or cache mapping functions, to support countermeasures against side-channel

attacks and other attacks exploiting the addressing scheme. Since typically only small parts of the address should be remapped in a pseudorandom way, this requires primitives with very small block lengths. Examples include the 24-bit block cipher BipBip [BDD+23a], the 10-bit SCARF [CGL+23a], or the 16-bit QARTA used to instantiate a PRF with configurable output distribution in the SassCache cache design [GSP+23].

Outline. We outline the attack setting and design considerations in Section 2. In Section 3 we specify the data path and tweak-key schedule. Section 4 provides the design rationale and Section 5 the security analysis of the cipher. Section 6 analyzes its latency in hardware.

2 Background and Attacker Model

In this section we introduce two attack scenarios on memory encryption ciphers and further explain the U-shape attack setting. Furthermore, we argue what the requirements for a low latency cipher are from an architectural point of view, when we want to protect the internal memories of a 32-bit CPU.

2.1 Attack Scenarios

Stemming from the specific target application of the cipher, we have to account for attackers with different types of capabilities. In general, the target application for the cipher is the enablement of logical attack protection, such as SERVAS [SSW+21] or CrypTag [NSW+21], but tailored to MCUs. In particular, these countermeasures use a tweakable encryption in combination with context specific tweaks, to isolate these contexts from each other. This can be in the form of domain specific tweaks for isolation as in the case of SERVAS, where the tweaks can be secret and composed of 64 bits or more. Additionally, tweak bits can be used for cryptographic memory tagging, as in the case of CrypTag. Finally, instructions themselves could impact the tweak, for instance to allow using different tweaks when handling return addresses and when handling data on the stack and therefore preventing an attacker from overwriting the return address in a controlled manner. Together with the address, which is 32 bits large for MCUs, deploying several such protections at once already exceeds 100 tweak bits. Hence, we assume a tweak which is 128 bits large.

For this application of the cipher we consider two attack scenarios: First, an attacker who can overcome logical isolation[1] and therefore read internal memory which they should not be able to read. In such an attack, we assume that the attacker accesses the memory content with the wrong tweak (i.e., $C = E^{-1}_{K,T'}(E_{K,T}(P))$). This constitutes a scenario similar to SCARF [CGL+23a] where a software attacker cannot access the ciphertext itself, but only a wrong decryption of the very same. Second, an adversary could try to read out internal memories by means of physical attacks and therefore access $\overline{C}$ directly.

For the first attack scenario, which we will refer to as attacking the U-shape, we will dedicate Section 5. For the second scenario of physical attacks on internal memories, we will argue that they are mostly impractical for chips manufactured in modern technologies or can be vastly aggravated by additional software measures.

In case of attacks on internal memories we have to argue about ROM, non-volatile memory and RAM. For simplicity, we assume all memories to have a size of 1 MB, which translates to 2^{18} words. Given that we have address dependent encryption, for ROM, an attacker would have either 2^{18} pairs (in the context of U-shape) for the entire memory or a single pair per address. For non-volatile memory we can make the same assumptions, as read-out methods [CSW16, ZLG24] are destructive, and the content cannot be updated

[1] Typically, a configuration in a CPU which would define context specific rules for granting access to different memory regions

after a first read-out. As such, it is sufficient to argue that an attack on one half of the U-shape with this data complexity is infeasible.

For RAM the situation is more involved. Whether we can draw similar conclusions for RAM as for the other memories above, depends on the boundary conditions. On the one side, one has to consider the used technology node and on the other side, whether custom boot software can be executed on the device.

On devices, which perform memory encryption, we can typically also assume that a user cannot perform attacks like [MH22], where the SRAM state is retained and read after reboot by custom software. This is mainly because we can assume that authenticated boot is used, or the cryptographic initialization of RAM is already done in ROM code.

What then remains is the impact of the technology node. If we look at technologies larger 40nm, thermal laser stimulation probing can be used to scan a device from the backside (after appropriate sample preparation). In such a scenario, we can assume that it takes a minute to read a single word and about an hour for an entire 16kB RAM readout on a 180nm technology with a 100k$ setup [KLS+20]. This would lead to $24 \times 365 \times 2^{12} \approx 2^{25}$ chosen or known plaintext/ciphertext pairs per year for an entire 1MB RAM. If a single address is attacked, up to $60 \times 24 \times 365 \approx 2^{19}$ plaintext/ciphertext pairs can be retrieved in a year. For laser voltage probing, we did not find a reference from which we could deduce the time it would take to extract an SRAM, only the feasibility to extract a register from a 60nm FPGA [LTBS16].

If we look at smaller technology nodes, that is 28nm and below, the resolution of optical means becomes the limiting factor and one would refer to e-beam techniques [AJK+24]. However, there the samples have to be so heavily prepared that this can only be done for a limited area. Trenching an entire RAM will most likely lead to a thermal or electrical issues. First, identifying the area of interest by optical means and afterward preparing the chip does not work either. This is because after a reboot, the address scrambling[2] changes and therefore also the area of interest. Thus, for smaller technologies, we can rule out SRAM readout attacks. Some publications [ABB+21] (and references therein) state that one could go down to even 10nm, but there is no evidence and if successful, this data will come with a certain bit-error-rate.

Summarizing, when deploying such memory encryption scheme for a security subsystem in a modern System on Chip (SoC), probing attacks to read out internal RAM are considered out of scope and this is also the area of application that we suggest. If the desire is to use it for a secure element with older technology, one should consider adjusting the number of rounds. Additionally, if the attacker has only API access, which is usually the case for secure elements, additional measures can be taken (e.g., changing keys/tweaks/addresses or cleaning up memories per API invocation).

Finally, if we only consider remote attacks, we can rule out ciphertext accesses entirely as long as the encryption of memories is enforced permanently.

2.2 Architectural Motivation

Given the attack scenarios for the U-shape, we require that the round keys are very different when decrypting with a wrong tweak. Hence, we propose to use a complex key schedule.

From a latency point of view it is also beneficial to outsource some complexity into the key schedule and in exchange to keep the encryption/decryption itself leaner. To explain why this presents an advantage, we need to consider three types of memory accesses: word-sized reads (32-bit), word-sized writes, and sub-word-sized writes (in our case, 16- or 8-bit). For reads, the address is present on the bus in a first cycle and the data is returned from the memory only in a second cycle. This is of course the optimal case and

[2] An obfuscation technique against reverse engineering which randomizes the logical-to-physical address mapping

the response from the memory might take longer, but for tightly coupled internal memories, this typically holds. As a consequence, a complex key schedule that only depends on the address and some tweak material but not on the data can be calculated while waiting for the data to arrive from memory. The key observation is that we can spend significant computation time without blocking any other operation. Then, when the data arrives, only the leaner decryption itself has to be carried out. For writes, on the other hand, the situation is different as the address and the data can be present on the bus in the same cycle. Thus, in this case, the cycle that has to be spent to calculate the key schedule before carrying out the encryption introduces a stall cycle where the rest of the system needs to wait. The least beneficial case are sub-word-sized writes, because in this case, a read-modify-write has to be performed. That is, the data is read, then either a half-word or a single byte is replaced within the read word and afterward the data has to be written back to memory. The operations that need to be performed for this are the key schedule calculation, a decryption, and an encryption. Compared to a word-write, an additional cycle has to be spent, and hence two stall cycles are introduced.

The second important observation is that the most frequent memory accesses are reads, followed by word writes, followed by sub-word writes. In fact, in our benchmarks, the overall impact of the extra cycles during writes (and sub-word writes) on the entire cycles count was only 2.5% for the CoreMark CPU benchmark [cor]. Thanks to the 32-bit block size, sub-word writes become negligible, the benchmarks show that their impact is below 1%; on the other hand, using a 64-bit cipher, every single write would have introduced a read-modify-write without a cache.

Finally, while we do see a cycle count increase of around 2.5%, the design allows us to almost sustain the maximum clock frequency of a four-stage pipeline CPU as the CV32E40P [CV3]. In fact, we only saw a 16% drop of the maximum clock frequency and could synthesize the design at 500 MHz in a 16nm technology. We did not put too much effort into optimizing the integration and believe that the maximum frequency drop can be decreased. In addition, the absence of stall cycles during read operations is even more beneficial when the scheme is also deployed in the instruction path, as there the frequency of read operations is much higher than in the data path.

Hence, as we would always perform the tweak-key schedule and the data path in separate cycles, the latency numbers for them are given separately in Sections 4.2 and 6.

3 Specification

The cipher consists of a datapath and a tweak-key schedule. We display the relation between these two building blocks in Figure 2.

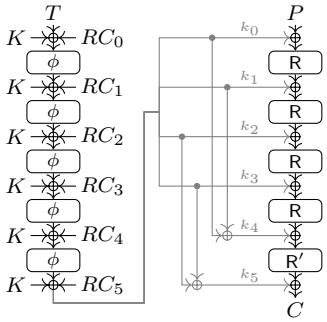


Figure 2: Tweak-key schedule and data path of 5-round BEANIE. ϕ, R, R′ are the round functions φ, R, R' omitting the key and constant addition.

3.1 Data Path

As shown in Figure 2, the data path consists of $r = 5$ rounds given as $R' \circ R^4$. The round R_i, with $i = 0, \ldots, r-2$, consists of a $\mathsf{AddRoundTweakey}_i$ operation ART_i, a SubCells layer SC, a ShiftRows transformation SR, and MixColumns operation MC. In the final round R' the MC layer is omitted due to the weak impact against many attacks and an additional whitening key is appended with an ART_r operation. In short:

$$R_i = \mathsf{MC} \circ \mathsf{SR} \circ \mathsf{SC} \circ \mathsf{ART}_i, \quad R' = \mathsf{ART}_r \circ \mathsf{SR} \circ \mathsf{SC} \circ \mathsf{ART}_{r-1}.$$

Each operation is performed on a 32 bit state. We view this state as a 4×2 array of cells, each cell having the size of a nibble. The indexing of the cells (starting at 0) goes from top to bottom and from left to right and is shown at state X_i in Figure 3. We refer to $X[i]$ as the nibble located in the $i \mod 4$ row and the $\lfloor \frac{i}{4} \rfloor$ column. Note that if the 32-bit state is given as a concatenation of bits $X = (x_0, x_1, \ldots, x_{32})$, then the nibble with the index $i \in \{0, 1, \ldots, 7\}$ contains the bits $(x_{4i+0}, x_{4i+1}, x_{4i+2}, x_{4i+3})$.

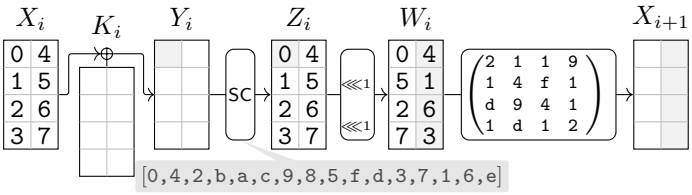


Figure 3: One round of BEANIE.

$\mathsf{AddRoundTweakey}_i$. A 32-bit round tweak-key k_i is XORed to the state. This round tweak-key is derived from the tweak and the key by the tweak-key schedule.

SubCells. On each cell of the state we apply the S-Box S. The mapping performed by S is shown in Figure 3 and in Table 12a in the appendix.

ShiftRows. The cells of the second and fourth row are rotated by one position to the left. We can describe this rotation SR in terms of cell indices by

$$(0, 1, 2, \ldots, 7) \xrightarrow{\mathsf{SR}} (0, 5, 2, 7, 4, 1, 6, 3).$$

MixColumns. For the MixColumns operation we view the nibble of each cell as an element in $\mathrm{GF}(2^4)/0x13$ (the polynomial used is $p(x) = x^4 + x + 1$). Each column of the state can therefore be viewed as a polynomial with a term for each row. These polynomials get multiplied with an involuntary 4×4 MDS matrix M [JPST17a], which in hex representation is shown in Figure 3. When multiplying a column of our state with M from the left side, we therefore need to take the result modulo $p(x)$.

3.2 Tweak-key Schedule (TKS)

A suitable key schedule of the cipher has to accommodate a tweak with 128 bits and a 128-bit main key. The tweak itself includes 32 address bits, as well as other memory related parameters mentioned in Section 2.1, while the size of the main key comes from the cryptanalytic perspective. We realize the TKS by an alternating block cipher. This cipher takes a 128-bit main key K and maps the 128-bit tweak T to a set of intermediate round keys $S = k_0||k_1||k_2||k_3$, where k_i are 32-bit blocks.

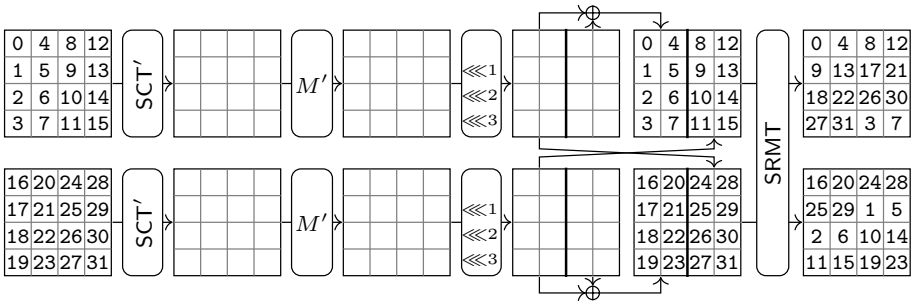


Figure 4: One round of the tweak-key schedule of BEANIE without AddKey.

Table 1: The round constants of BEANIE in hex.

RC_0	00000000000000000000000000000000	RC_5	0000000000000000be5466cf34e90c6c
RC_1	00000000000000013198a2e03707344	RC_6	0000000000000007ef84f78fd955cb1
RC_2	0000000000000000a4093822299f31d0	RC_7	000000000000000085840851f1ac43aa
RC_3	0000000000000000082efa98ec4e6c89	RC_8	0000000000000000c882d32f25323c54
RC_4	00000000000000000452821e638d01377	RC_9	0000000000000000064a51195e0e3610d

Our approach for the TKS construction is based on the idea used in QARMA128 [ABD+23], where we use two copies of the 64-bit PRINCE [BCG+12] round function and then mix the states after every round. The mixing is done by a Feistel Type-2 and AES rotation over the 128-bit state (shifts by 2 cells row-wise). To reflect the composition of two PRINCE rounds, we view the state as two 64-bit blocks. Each of these blocks can be segmented into a 4 by 4 matrix of cells. The indices for each cell over the whole 128-bit state are shown in the first state of Figure 4. Note that if the 128-bit state is given as a concatenation of bits $X = (x_0, x_1, \ldots, x_{127})$, then the nibble with the index $i \in \{0, 1, \ldots, 31\}$ contains the bits $(x_{4i+0}, x_{4i+1}, x_{4i+2}, x_{4i+3})$.

As shown in Figure 2, the tweak-key schedule consists of $r = 5$ applications of the round function φ. The round φ_i, with $i = 0, \ldots, r-1$, consists of a AddKey_i operation AK_i, a SubCellsT layer SCT, a MixColumnsT operation MCT, a ShiftRowsT transformation SRT and the mixing operations FeistelMixT (FMT) and ShiftRowsMixT (SRMT).

In short (see also Figure 4):

$$\varphi_i = \mathsf{SRMT} \circ \mathsf{FMT} \circ \mathsf{SRT} \circ \mathsf{MCT} \circ \mathsf{SCT} \circ \mathsf{AK}_i.$$

After the last round, an additional AK_r operations is applied. Here, $\mathsf{SRT} \circ \mathsf{MCT} \circ \mathsf{SCT}$ implements two copies of PRINCE round function (with non-PRINCE 4-bit S-Box), while $\mathsf{SRMT} \circ \mathsf{FMT}$ mixes the states.

AddKey_i. The main key K is XORed to the state. To prevent weak key attacks, an additional round constant RC_i is added together with the key. The round constants are the same used for PRINCE and displayed in Table 1.

SubCellsT. To both blocks of the state, a non-linear layer SCT' is applied. Concretely, we apply to each nibble the 4-bit S-Box used in the data path (given in Table 12a).

MixColumnsT. To the two blocks of the state, the M' matrix of PRINCE [BCG+12] is applied. When working with the whole state, we can view this as an application of the extended block diagonal matrix of PRINCE, given as

$$M_{128} = \mathrm{diag}(M', M') = \mathrm{diag}(\hat{M}^{(0)}, \hat{M}^{(1)}, \hat{M}^{(1)}, \hat{M}^{(0)}, \quad \hat{M}^{(0)}, \hat{M}^{(1)}, \hat{M}^{(1)}, \hat{M}^{(0)}).$$

Here, we use the matrices

$$M_0 = \begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}, \quad M_1 = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}, \quad M_2 = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}, \quad M_3 = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix}$$

to compose the matrices $\hat{M}^{(i)}$, $i = 0, 1$, by

$$\hat{M}^{(0)} = \begin{pmatrix} M_0 & M_1 & M_2 & M_3 \\ M_1 & M_2 & M_3 & M_0 \\ M_2 & M_3 & M_0 & M_1 \\ M_3 & M_0 & M_1 & M_2 \end{pmatrix}, \quad \hat{M}^{(1)} = \begin{pmatrix} M_1 & M_2 & M_3 & M_0 \\ M_2 & M_3 & M_0 & M_1 \\ M_3 & M_0 & M_1 & M_2 \\ M_0 & M_1 & M_2 & M_3 \end{pmatrix}.$$

For an input state X (viewed as a 128-bit vector), the application of M_{128} can be done either as $X \times M_{128}$ or $M_{128} \times X^T$, since it is a symmetric matrix.

ShiftRowsT. This operation applies two PRINCE rotation layers in parallel (operating within the two blocks of the state). Each operation is actually the AES rotation SRT over the 4×4 block which is described as

$$(0, 1, \ldots, 15) \xrightarrow{\mathsf{SRT}} (0, 5, 10, 15, 4, 9, 14, 3, 8, 13, 2, 7, 12, 1, 6, 11).$$

FeistelMixT. We apply a Type-2 round function with the *identity* inner functions defined on 32-bit branches, i.e.

$$\text{Feistel Type-2} : (X_0, X_1, X_2, X_3) \rightarrow (X_1 \oplus X_0, X_2, X_3 \oplus X_2, X_0),$$

where $X_k = (x_{32 \cdot k+0}, \ldots, x_{32 \cdot k+31})$, $k = 0, 1, 2, 3$. Note that X_0 represents the first two columns (cells $0, \ldots, 7$), X_1 represents the next two columns, etc.

ShiftRowsMixT. When viewing the state as an 8×4 matrix of cells, we apply the AES rotation by grouping two cells. More specifically, by applying $\lll [0, 2, 4, 6]$, we rotate the cells of the four rows by 0, 2, 4, and 6 cells, respectively.

Relation to the datapath. The 128-bit output $TKS_K(T) = (k_0, k_1, k_2, k_3)$ supplies the 5-round datapath with the round keys k_i, $i = 0, \ldots, 5$. We derive round keys k_4 and k_5, as well as k_6, k_7, k_8, k_9 for extended versions of the cipher, with (see Figure 2)

$$k_4 = k_0 \oplus k_1,\ k_5 = k_2 \oplus k_3,\ k_6 = k_0 \oplus k_2,\ k_7 = k_1 \oplus k_3,\ k_8 = k_0 \oplus k_3,\ k_9 = k_1 \oplus k_2.$$

We provide test vectors in Appendix C.

3.3 Security Claims

Stemming from the real-world application described in Subsection 2.1, we define a practical security claim limiting the attacker to 2^{80} time and 2^{40} data in the U-shape attack setting. For this security claim, the cipher has a 5-round data path and a 5-round tweak-key schedule. In case the latency requirements are less stringent, we recommend a 7-round data path and a 7-round tweak-key schedule, leading to 128-bit security. As outlined in Subsection 5.6, the cipher provides insufficient security guarantees in the non–U-shape attack setting for both the 5 round and the 7 round version of the cipher. Therefore, we do not claim any security in this attack setting.

4 Design Rationale

4.1 Considerations for the Data Path

In this subsection we describe design decisions behind the structure of the round function $R = \mathsf{MC} \circ \mathsf{SR} \circ \mathsf{SC}$. We provide some relevant properties of the layers, as well as certain justification arguments for their choices in general.

The 4-bit S-Box. The S-Box in Table 12a belongs to the $G7$ class [Saa11, Table 1], which has optimal differential and linear properties (with branch number 2). In addition, this S-Box has maximal algebraic degree (3), which helps against algebraic attacks.

More precisely, viewing S as a vectorial Boolean mapping over $\mathbb{F}_2^4$ (binary vector space of dimension 4), i.e. $S(x) = (f_0(x), \ldots, f_3(x))$, we have the Boolean functions f_i given as:

$$
\begin{aligned}
f_0(x_0, \ldots, x_3) &= x_0x_1x_2 + x_0x_1x_3 + x_0x_1 + x_0x_2x_3 + x_0x_3 + x_1x_2x_3 + x_1x_3 + x_2x_3 + x_2 \\
f_1(x_0, \ldots, x_3) &= x_0x_1 + x_0x_2x_3 + x_0x_3 + x_0 + x_3 \\
f_2(x_0, \ldots, x_3) &= x_0x_1x_2 + x_0x_2x_3 + x_0x_2 + x_0x_3 + x_1x_3 + x_1 + x_2 \\
f_3(x_0, \ldots, x_3) &= x_0x_1x_3 + x_0x_1 + x_1x_2 + x_3.
\end{aligned}
$$

Note that algebraic degree, i.e., $\deg(S) = \min\{\deg(f_i) : i = 0, 1, 2, 3\} = 3$, equals the degree of the component functions as well: $\deg(\bigoplus_{i=0}^{3} c_i f_i(x)) = 3$ for all $(c_0, \ldots, c_3) \neq (0, 0, 0, 0)$.

Clearly, there exist other 4-bit S-boxes which are more lightweight while maintaining good algebraic properties (e.g. MIDORI S-Box Sb0 [BBI+15], PRINCE [BCG+12]), which could improve the overall performance. In our search for the S-box, we not only considered algebraic properties but also implementation and differential properties. Generally, our search was driven by the following considerations: (1) Good differential and linear properties (2) Good resistance against algebraic attacks based on degree of component functions (3) Latency within the whole round function R (i.e. not individually against other lightweight S-Boxes).

We did not exhaustively test all 4-bit S-Boxes in terms of the latency/area in the context of full implementation of the cipher. Instead, we selected a set of well-known S-boxes that have good statistical and algebraic properties and measured their implementation characteristics in terms of area and latency. According to our somewhat limited search it appears that S achieves better performance within the round function R (considered over more rounds, see Section 6 and Appendix B) compared to other well-known S-Boxes.

ShiftRows. The rotation layer ShiftRows $\ggg [0, 1, 0, 1]$ corresponds to the AES ShiftRows layer when viewed on the state of the format 4×2. On the other hand, let us assume that the cipher state is represented in the format 4×4, where each cell contains precisely 2 bits. By exhausive search over all row-wise rotations of $0, 1, 2$ or 3 cells, we have found that the rotation corresponding to AES ShiftRows ($\lll [0, 1, 2, 3]$) gives the same (maximal) contribution to differential resistance (in terms of active S-Boxes) as $\ggg [0, 1, 0, 1]$. Note that there exist other rotations which maintain the same differential resistance as $\ggg [0, 1, 0, 1]$. We choose to include the ShiftRows operation in the last round of encryption to permit a lower-area implementation of a combined encryption/decryption circuit (see Section 6).

MixColumns. The MixColumn matrix given in Section 3 has been provided in [JPST17b, Subsection 5.4], denoted by $M_{4,i,4}$. It is an involutive MDS matrix over $GF(2^4)$, which greatly improves the resistance to differential-like attacks (due to its maximal branch number), as well as the full diffusion property of $R = \mathsf{MC} \circ \mathsf{SR} \circ \mathsf{SC}$ being 2 rounds.

The choice of a MixColumn matrix is mainly based on the following aspects: impact on the resistance to statistical attacks, impact on the diffusion of the round function R, and overall performance when implemented in a cipher.

Regarding the performance, in [JPST17b, Table 3] it was shown that $M_{4,i,4}$ behaves optimally in terms of s-XOR metric in comparison to other common types of MDS matrices (cf. Definition 2 in [JPST17b]). Considering the performance of $M_{4,i,4}$ within the whole round R, we have also compared it to another well-known type of lightweight matrices known as Hadamard, that is an involutive type matrix $Had(1, 4, 9, d)$ over $GF(2^4)/(0x13)$ (see [SKOP15, Table 1]). Note that these matrices can be entirely described via the first row only. According to our implementation, $M_{4,i,4}$ achieves slightly better performance in terms of time and quite similar in area (see Section B for more details). On the other hand, since BEANIE has smaller number of rounds, it is important that it quickly achieves full diffusion, which prevents usage of lightweight MixColumn matrices like those in SKINNY or MIDORI. We compare the diffusion of these matrices when used in BEANIE in Table 2. Similarly, we compare implementation performance in Table 14.

Table 2: Diffusion comparison between $M_{4,i,4}$ and other MixColumn matrices when used in BEANIE. "$*$" denotes *almost* full diffusion, i.e., certain bits do not affect all output bits.

MixColums Matrix	SKINNY	MIDORI	$M_{4,i,4}$
# rounds for forward diffusion	3^*	3^*	2
# rounds for backward diffusion	3	3^*	2

In the context of security, we are compare the matrices in the context of the U-shape, as discussed in Subsection 4.2. Based on a cell-wise MILP model, we find that if BEANIE would use the SKINNY or MIDORI matrices, we would need about 2 more rounds (in each branch of the U-shape) to achieve the same level of security against differential distinguishers. Additionally, the better diffusion of the MDS matrix also makes key-recovery harder. We give the detailed results for the differential analysis in Table 4. Similarly, we compare implementation performance in terms of area and latency in Table 3.

Table 3: Performance comparison (16nm cell library) for BEANIE with MIDORI/SKINNY MixColumns over 7 rounds and $M_{4,i,4}$ over 5 rounds. The S-Box is always from Table 12a.

MixColumn	Time (ps) Encryption	Area (μ^2) Encryption	Time (ps) Decryption	Area (μ^2) Decryption
MIDORI (7R)	1184	1068.3	1250	1104.2
SKINNY (7R)	1068	885.5	1075	898.5
$M_{4,i,4}$ (5R)	987	949.6	1035	926.7

Table 4: Number of differentially active S-Boxes (based on a cell-based MILP model) of the cipher with MIDORI/SKINNY and $M_{4,i,4}$ MixColumns when considered in U-shape.

Matrix \ #Rounds	4	5	6	7	8	9	10
MIDORI			12	14	16	18	20
SKINNY			16	18	20	22	24
$M_{4,i,4}$	15	16	20	23	30		

Remark 1. Considering the fact that ShiftRows and MixColumns are involutive, raises the question whether BEANIE should use an involutive S-box as it allows reusing the same circuit for encryption and decryption while only changing the round keys. We decided against this optimization due to potential security issues. In [Bir03], a slide-with-a-twist attack has been demonstrated on 5-round on the AES-like cipher Khazad which exploits its cyclic property of its round function under weak keys. In particular, [Bir03, Subsection 4.2] describes a general attack approach which can be applied to any cipher that can be decomposed as $E_K(x) = P_K \circ F \circ Q_K(x)$, for an involutive F possibly assuming weak

keys. As the encryption function of BEANIE would admit this decomposition if involutive S-boxes are used, we opt to use non-involutive S-boxes.

4.2 Considerations for the TKS Construction

Recall that the key schedule has to accomodate a large tweak of length 128 bits, and a main key of length 128 bits. Due to the small state size of the data path, we need a full tweak-key schedule rather than a generic construction such as XEX to avoid birthday-bound attacks [Dae91, KR11] and accomodate the full tweak. The construction of the TKS is based on the following analysis, for which we assume that:

1. The secret main key K can not be affected by the attacker (tamper resistant). We note that the application of K after every round may impose a weakness against different attacks. While the TKS does not drastically boost the security of the data path, as explained later on in Section 4.2, it still prevents additional wekanesses. If one aims for a higher security of the TKS itself, we suggest the use of more rounds and a 256-bit main key $K = K_0||K_1$ where the key parts K_i would be used alternatively in TKS (as in PRINCEv2). On the other hand, regarding the tweak T, we assume to be known and the attacker is allowed to choose its value.

2. The underlying tweak-key schedule is constructed such that it maps the pair (K, T) into a set of round keys, i.e. $TKS : (K, T) \rightarrow \{k_i : i \in I\}$, for some set of indices I. In addition, we assume that the TKS achieves *full diffusion*, i.e. all bits of k_i depend on all input bits of the pair (K, T). This requirement is motivated by the fact that any attack which manipulates the tweak may eventually aim to achieve collisions in $\{k_i : i \in I\}$, that is, for suitable tweaks T one can try to obtain sets of round keys among which some keys are the same (especially those at the bottom of the U-shape). Consequently, this would allow easier application of certain attacks in general, as it effectively reduces the number of rounds in the data path in the U-shape setting [CGL+23b]. For this purpose, the cost of obtaining collisions in the set of round keys $\{k_i : i \in I\}$ should not be too low in general, and has to be aligned with the best attack on the cipher (more details will be given later on).

3. We assume that the attacker is considering the U-shape scenario, i.e. some attack(s) is applied at fixed address A, where input-output pairs (P, C) are obtained via the U-shape as $P \rightarrow E_{K_1,T_1} \rightarrow E^{-1}_{K_2,T_2} \rightarrow C$. Here, both tweaks T_1, T_2 correspond to the address A, and C is essentially the decryption of $E_{(K_1,T_1)}(P) = C'$ using different tweak T_2 (which imposes a different set of round keys K_2 via TKS), i.e., $C = E^{-1}_{K_2,T_2}(C')$. Note that obtaining C from C' is seen as querying the memory oracle to decrypt C' using given wrong tweak T_2, which results in unknown set of round keys K_2. Clearly, the attacker is choosing $T_2 \neq T_1$ on purpose, since otherwise the oracle will return P after decrypting C' and then the attacker can not learn anything from single known P.

Now, let us assume that the goal of the attacker is to attack TKS by collecting a sufficient amount of round key sets. Assume that these round key sets are obtained by attacking the U-shape (at a fixed address A), by having sufficient input-output pairs. In this scenario, we have that the complexity of attacking the U-shape will be approximately equal to the complexity of the first (best) attack that was applied to recover the first round key of the first set. Denote this complexity by 2^c, which essentially represents the most dominant value in terms of time/data/memory complexities. Furthermore, assume that the attack needs 2^a round key sets in order to attack TKS. This means that the overall cost of obtaining the underlying round key sets is $\approx 2^{c+a}$.

Consequently, in order to break the twekey schedule and extract the main key K, one needs to apply the best attack on TKS which recovers at least some portion of K. Let the complexity of this attack be given as 2^t. Here we consider that the complexity of recovering whole K is possible by re-applying the best attack on TKS sufficiently many times, after which the total complexity of recovering full K is approximatelly equal to the complexity of the very first attack which recovered portion of K (that is 2^t).

We observe that the security of the cipher is not boosted by the security of TKS, but rather the construction and security of TKS prevent easier application of different attacks on the cipher itself. On the other hand, the t-bits of security of TKS also means that the main key K is not exposed directly, where in practice the assumption of obtaninig sufficiently many input-output pairs to break the U-shape, is strongly depending on the type of the memory as discussed earlier.

The main properties of the TKS are thus given as follows:

1. Full diffusion is achieved after 3 rounds, greatly supported by the Feistel Type-2 layer. The performance numbers related to time and area are given in Table 5.

Table 5: Performance numbers (on NanGate 15nm) of TKS, using the S-Box in Table 12a.

Number of rounds	Time (ps)	Area (μ^2)	Area (GE)
4	172	2092	10640
5	212	2631	13383

2. The number of differentially active S-Boxes based on a cell-based MILP model is given in Table 6. Regarding the linear cryptanalysis, we recall that the PRINCE MixColumn block matrices $\hat{M}^{(i)}$ $(i = 0, 1)$ are involutive and symmetric. Thus, the cell-based MILP modeling of M_{128} in terms of linear maskings stays the same as for modeling the propagation of differentials. In addition, our model confirms that the number of linearly active S-Boxes is the same as in Table 6, since the cell-additions of linear masks in Feistel Type-2 layer are modeled in the same way as cell-additions of differentials. Consequently, using the bias of SubCells given by 2^{-2}, the overall bias is derived based on Matsui's PilingUp Lemma [Mat93] as $\varepsilon = 2^{AS-1} \cdot 2^{-2 \cdot AS} = 2^{-AS-1}$, where AS is the number of linearly active S-Boxes. The data complexity is then roughly estimated as $\varepsilon^{-2} = 2^{2 \cdot AS+2}$. Regarding the impossible differential cryptanalysis on TKS, we confirm the existence of distinguishers for 3 and 4 rounds, where at least more distinguishers one can find by considering low-weighted input and output differentials (activity of a differentially active cell is denoted by 0 or 1).

Table 6: Number of active S-Boxes of the proposed TKS.

Number of rounds	3	4	5	6
Number of active S-Boxes	8	18	24	27

5 Security Analysis

Under the assumption that the ciphertext is not available to the attacker (Section 2.1), we use the U-shape attack scenario. For a cipher $E_{K,T}(P) = \overline{C}$ with tweak-key pair (K, T) and non-available $\overline{C}$, this means that the attacker queries the MMU to decrypt $\overline{C}$ with a different tweak T'. The tweak-key schedule will produce round keys different from the correct one (for the correct tweak), and thus the information flow will look as follows:

$$P \rightarrow E_{K,T}(P) = \overline{C} \rightarrow E^{-1}_{K,T'}(\overline{C}) = C,$$

where the secret main key K can not be changed/affected by the attacker and $T' \neq T$. In this context, we view $U = E^{-1}_{K,T'} \circ E_{K,T}$ as an encryption scheme that produces (P, C) pairs $U(P) = E^{-1}_{K,T'}(E_{K,T}(P)) = C$, where the round tweak-keys used in $E_{K,T}$ and $E_{K,T'}$ are independent due to the strong tweak-key schedule. The parts $E_{K,T}$ and $E^{-1}_{K,T'}$ will be called *branches* of the U-shape (left l and right r branches). Stemming from the attack setting, in a round-reduced version of the cipher, both branches get shortened by the same number of rounds. However, the reduced round R' is always applied in the last round.

The cipher achieves full diffusion after 2 rounds, except when the last round R' is involved. In this case, we need 3 rounds for full diffusion. It may take up to 5 rounds to achieve full diffusion when the bottom layers of the U-shape are considered, since the transitions from left to right branches annuls parts of the diffusion of the linear layers. Note that the round key recovery attacks will target the initial rounds of the left branch $E_{K,T}$ (thus from the plaintext side) or the last round keys of the right branch $E^{-1}_{K,T'}$ (from the ciphertext side). Therefore, the inferior diffusion properties towards the last rounds of the cipher have little impact on the key recovery.

Distinguishers used in attacks can always be mirrored due to the U-shape. Thus, a $(r_l + r_r)$-round distinguisher stretching over r_l-rounds of $E_{K,T}$ and r_r-rounds of $E^{-1}_{K,T'}$ (connected at the bottom of U-shape), imply the existence of a $(r_r + r_l)$-round distinguisher.

Once we recovered the subkeys k belonging to a (K, T) pair, it is straightforward to gather additional subkeys derived from another tweak T'. This is because we then know the encrypted value $E_{K,T}$ and therefore only need to attack $E^{-1}_{K,T'}$. Thus, we can obtain an arbitrary number of data to attack the tweak-key schedule. Furthermore, the tweak-key schedule is easier to attack than the data path. We therefore assume that a valid attack on the data path leads to a valid attack on the cipher in the U-shape attack setting.

We assume that the first 4 round keys are independent (due to the strength of the TKS) and thus we do not consider the existence of exploits that come from the tweak-key schedule. In particular, choosing a pair of tweaks that lead to the same round keys in the last round is prevented by the differential properties of the TKS. However, even in the generic case, we on average get a collision in the last round keys of $E_{K,T}$ and $E^{-1}_{K,T'}$ after trying 2^{32} different tweaks for T'. If this is the case, the last round of $E_{K,T}$ and the first round of $E^{-1}_{K,T'}$ cancel each other. Thus, by increasing the time and data complexities by a factor of 2^{32} we can extend each attack by one round in the left and right branch. Note that it is also possible to exploit a partial collision in the last round key (see Subsubsection 5.3.1).

In Table 7 we provide a quick overview of the best attacks with the U-shape attack setting we found. We also include extensions we get from generic last-round key collisions. We believe that the attack breaking the full round SCARF [FGLL+25] is not applicable to our cipher, since we use a more traditional approach of adding the round keys to the state.

Table 7: Best attacks on the data path in the U-shape attack setting. R denotes the number of rounds of $U = E^{-1}_{K,T'} \circ E_{K,T}$ in the U-shape setting. R Dist. is the number of rounds covered by an underlying distinguisher. T, D and M denote time, data and memory complexities. We extend some attacks using the generic last-round key collision.

Type	R	R Dist.	T	D	M	Reference
Boomerang	4+4	2+4	2^{62}	2^{29}	2^{30}	Subsection A.2
	5+5	2+4	2^{94}	2^{61}	2^{30}	Generic coll. extention
Impossible diff.	4+4	4+2	2^{73}	2^{11}	-	Subsection A.3
	5+5	4+2	2^{105}	2^{43}	-	Generic coll. extention
Impossible diff. partial coll.	5+5	4+3	2^{112}	2^{30}	-	Subsection 5.3.1
Integral	4+4	4+2	2^{67}	2^{18}	-	Subsection A.4
	5+5	4+2	2^{99}	2^{50}	-	Generic coll. extention
DS-MitM	5+5	4+3	2^{112}	2^{16}	2^{109}	Subsection A.5

5.1 Differential and Linear Cryptanalysis

Bounds. We search for the best differential and linear characteristics with a bit-based SAT model. Table 12b shows the difference distribution table (DDT) with a maximum differential probability of 2^{-2}. Similarly, the linear approximation table (LAT), displayed in Table 12c, has a maximal linear bias and thus a maximal squared correlation of 2^{-2}. This observation and the fact that the transpose of the MixColumns MDS matrix retains the MDS properties lead us to expect similar bounds for both differential and linear characteristics. Indeed, our bit-based model confirms this; for different branch lengths of r_l and r_r, the probability resp. squared correlation of the best characteristics is the same for both cases, as shown in Table 8.

Table 8: The differential probability (resp. squared correlation) of the best differential and linear characteristics for various rounds of r_l and r_r. We can calculate the probability p from the table entries v via $p = 2^{-v}$. Bold entries are exact probabilities, the rest are upper bounds. The border divides distinguishable from indistinguishable characteristics.

$r_l \setminus r_r$	0	1	2	3	4	5
1	**2**	**4**	**12**	**22**	32	34
2	**10**	**12**	**20**	**26**	40	
3	**16**	**22**	**26**	32		
4	**30**	32	40			
5	32	34				
6	40					

When using the $3 + 2$-round differential/linear distinguisher to attack $5 + 5$ rounds, more than 128 bits of round key material are involved in the key recovery. We therefore expect our cipher to resist classical differential and linear cryptanalysis, even when using advanced techniques like FFT [CSQ07] for key recovery.

Clustering effect, EDP, and ELP. To complete the picture and reduce the reliance on assumptions such as the dominant trail assumption, we also analyze clustering effects of differential and linear characteristics to estimate the actual Expected Differential Probability (EDP) and Expected Linear Potential (ELP) of promising differentials and linear hulls. These estimates still rely on the Markov assumption, which we consider justified for smaller round numbers due to the large key size compared to the state size.

Different approaches are available and have been used for other block ciphers with a relatively small state, such as BEANIE. One option is to recover individual rows/columns

Table 9: Estimated EDP resp. ELP of the best found differentials and linear hulls based on clustering. Results are based on the top t best characteristics ($t = 100$, in some cases fewer due to search time-outs) and then clustering all characteristics up to a threshold of probability 2^{-8} lower than the reference characteristic (or 2^{-4} for the harder cases $r_l + r_r$).

(a) Best found EDP based on clustering

$r_l \setminus r_r$	0	1	2	3	4	5
1					31.5	33.4
2					34.0	44.4
3				25.8	41.7	
4		31.5	34.0	41.7		
5	33.5	33.4	44.4			
6	40.3	43.9				

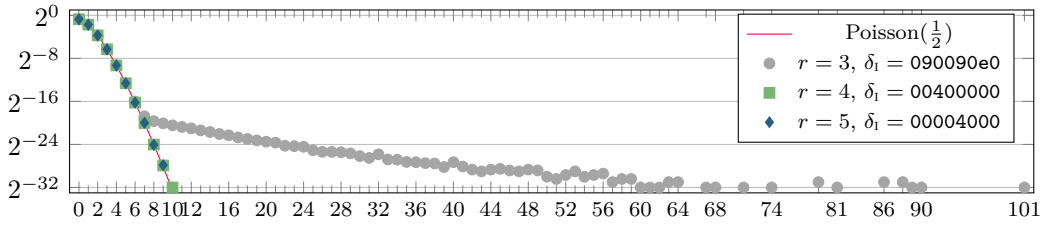
(b) Best found ELP based on clustering

$r_l \setminus r_r$	0	1	2	3	4	5
1					25.4	26.2
2					29.4	40.3
3				21.9	39.5	
4		25.4	29.4	39.5		
5	26.0	26.2	40.3			
6	33.3	39.6				

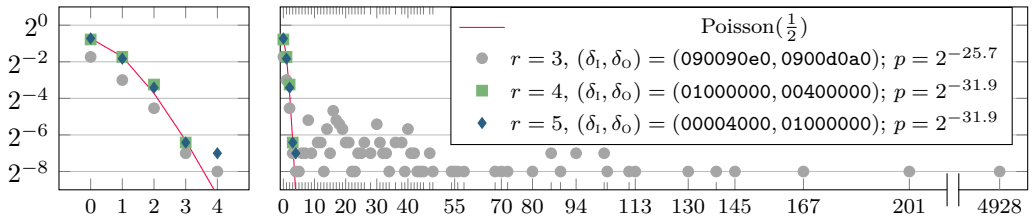
of the differential or linear transition matrix for the r-round cipher [ELR20], though the cost per row or column is non-negligible for a 32-bit block cipher (unless restricted to certain differential or linear activity patterns). An alternative approach that allows us to investigate more differentials and linear hulls in some detail is to measure the clustering effect. To this end, we use the same bit-wise SAT model as above to first find the top t best differential and linear characteristics. In a second step, we find all characteristics up to a certain probability threshold for each of the identified differentials and linear hulls of the first step. Adding their probabilities resp. squared correlations gives us better bounds for the EDP resp. ELP. Table 9 summarizes the best found results.

Fixed-key results. In addition to results under the Markov assumption, we investigate the behaviour under fixed keys. First, we check for potential high-probability differentials that we might have missed with characteristics-based approaches. By using the memoryless algorithm for surrogate differentiation by Dinur et al. [DDK+23], we find differentials with probability $p \approx 2^{-12.2}$ for 2+2-round BEANIE and with $p \approx 2^{-20.8}$ for 2+3-round BEANIE for the tested tweak-keys. To prevent spurious results, we used a different master key for running the surrogate differentiation algorithm and for verifying the probabilities.

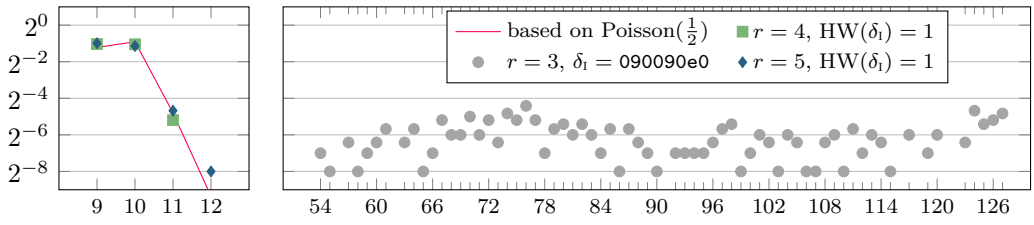
Second, we run a number of practical experiments across 2^8 random fixed tweak-keys, for each of which we precisely measure across all 2^{32} inputs. Figure 5 shows the results for differential cryptanalysis (counting the number $N(\delta_\text{I}, \delta_\text{O})$ of unordered right pairs), Figure 6



(a) Across all 2^{32} possible δ_O for fixed (T, K)



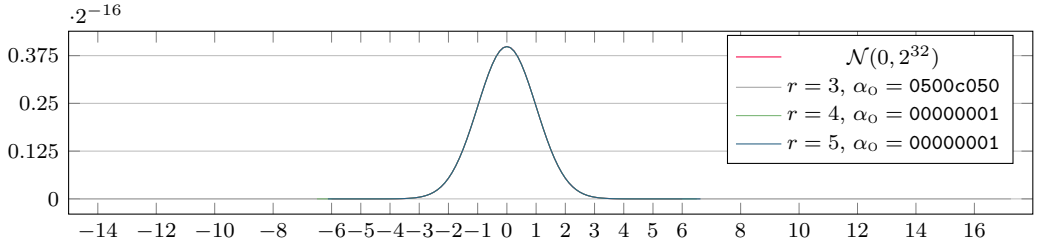
(b) Across 2^8 random (T, K) for fixed δ_O



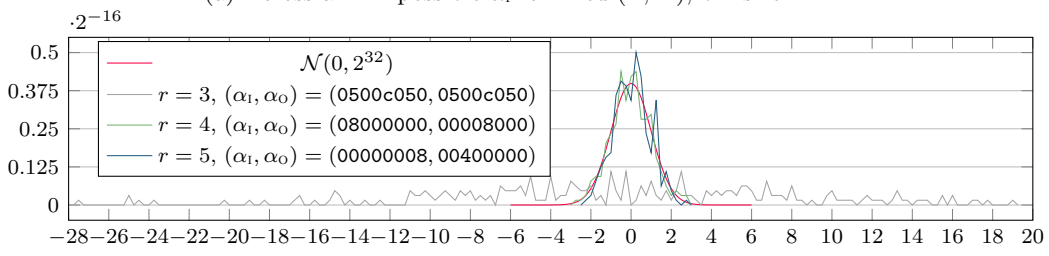
(c) Maximum (differential uniformity) across 2^8 random (T, K) and the best δ_I

Figure 5: Experimental and theoretical distributions of the number of (unordered) right pairs $N(\delta_\text{I}, \delta_\text{O})$ with fixed input difference δ_I for 3+3, 4+4, and 5+5-round BEANIE.

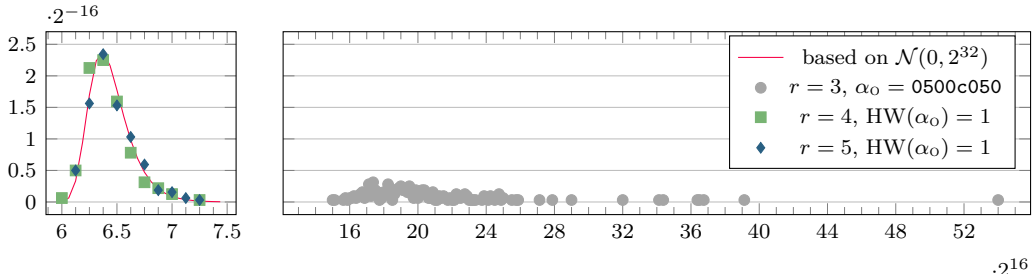
those for linear cryptanalysis (giving the imbalance of approximations, i.e., LAT entries). In each, we provide three plots: (a) the complete distribution of the metrics for the 2^{32} output differences δ_o for a fixed input difference (resp. input masks for a fixed output mask, obtained by applying the Walsh transform) under a single fixed key; (b) the distribution of the metrics for a fixed differential (δ_i, δ_o) resp. linear hull (α_i, α_o) across a sample of 2^8 tweak-keys; (c) the distribution of the maximum metrics (i.e., differential uniformity resp. linearity) for a fixed δ_i (resp. α_o) across 2^8 tweak-keys. Each plot compares the experimental data with the expected idealized behaviour, i.e., a Poisson distribution with parameter $\frac{1}{2}$ for differentials and a normal distribution with mean 0 and standard deviation 2^{16} for linear approximations, showing their probability mass functions resp. probability distribution functions (a,b) and their respective expected maxima for the given number of samples (c). Where necessary due to the distinguishability of $r = 3$, we split the plots into two selected sections of the x-axis. The fixed input differences (resp. output masks) used in the experiments are selected based on the best clustering result (Table 9a, Table 9b) for $r = 3$, and otherwise chosen with a single active bit (with varying bit positions, each of which leads to a minimal number of active S-boxes) for $r = 4, 5$. Overall, these results are well in line with what we would expect from the characteristics-based results: $r = 3$ is well-distinguishable and matches the expected probability, while $r = 4, 5$ appear infeasible to distinguish. There is one interesting outlier in Figure 5b with 4928



(a) Across all 2^{32} possible α_i for fixed (T, K); bin size 2^{13}



(b) Across 2^8 random (T, K) for fixed α_i; bin size 2^{14};
excluding 5 outliers for $r = 3$ at $(-52.75,\ -35.75,\ -32.75,\ 53,\ 62) \cdot 2^{16}$



(c) Maximum (linearity) across 2^8 random (T, K) and the best α_o; bin size 2^{13}

Figure 6: Experimental and theoretical distributions of the imbalance $N(\alpha_i, \alpha_o)$ with fixed output mask α_o for 3+3, 4+4, and 5+5-round BEANIE.

right pairs (DP $= 4928 \cdot 2^{-31} \approx 2^{-18.7}$ for one tweak-key pair for $r = 3$: for these specific tweak-keys, the last round keys collide in multiple relevant cells, thus rendering several S-box transitions "free". This effect does not significantly influence the EDP, however.

5.2 Boomerang Attack

We analyze the cipher with the sandwich technique [DKS14], which refines the boomerang attack [Wag99]. First, we search for a middle part E_m of the sandwich with a high probability. For this, we construct a cell-wise MILP model for a given number of rounds. In this model, we let the input/output of E_m propagate forward/backward with probability 1, respectively. The model returns an input and output mask for which a minimum number of S-Boxes overlap in these two propagations. We then fix the input and output to a certain difference that honors the input/output mask obtained by the previous step. Afterward, we experimentally calculate the probability p of E_m given these input/output differences. For a selection of rounds, we present the probabilities in Table 10.

Table 10: Probabilities of the middle part of the sandwich distinguisher.

R	p	Input	Output
2+2	1	`0xf0f00f0f`	`0x0f0ff0f0`
3+2	$\approx 2^{-13.6}$	`0x0000000f`	`0x0000000f`
3+3	$\approx 2^{-27}$	`0x00000f00`	`0x000f0000`

To find the best differential characteristics for the outer parts of the sandwich, we reused the SAT model of Subsection 5.1. The best boomerang distinguisher we find spans $2 + 4$ rounds and has a probability of $\approx 2^{-27}$. When we try to use this distinguisher to attack $5 + 5$ rounds, 112 subkeys bits are involved in the key recovery. Since we need to guess these subkeys for each data, the time complexity exceeds 2^{128}. Furthermore, the key recovery part is added to both ends of the distinguisher, thus we need to convert the boomerang attack to a rectangle attack [BDK01]. This further increases the data complexity. Due to the aforementioned reasons we are confident that our cipher is not vulnerable to this type of attack. See Subsection A.2 for a 4+4 rounds boomerang attack.

5.3 Impossible Differential and Zero-Correlation Linear Cryptanalysis

We use the meet-in-the-middle technique to find impossible differential (ID) and zero-correlation (ZC) distinguisher. This is done by a cell based MILP model that maximizes the number of active inputs [HGSE24]. We model deterministic forward and backward propagations and look for invalid overlaps. If this is the case, or when the branch number of the MixColumns operation is smaller than 5, then we have found a distinguisher. As already outlined in Subsection 5.1, in both the differential and the linear case, the MixColumns operation behaves the same. Since we model the S-Box as a black box, each impossible differential distinguisher is therefore also a valid zero-correlation distinguisher.

The longest distinguisher we find with our technique covers 4+2 (=2+4) rounds and is shown in Figure 9. In Subsection A.3 this distinguisher is used to attack 4+4 rounds. When mounting 5+5 round ID/ZC attacks with this distinguisher, 128 bits of key material are involved. However, as shown in Subsection 5.3.1, a partial round key collision can be used to extend this attack to 5+5 rounds. Due to the high time complexity, we argue that this attack does not threaten the security of our cipher.

5.3.1 Impossible Differential Attack with Partial Collision

When assuming a collision in one column of the last round key, we can mount an improved ID attack. We use the 4+3 round distinguisher displayed in Figure 7 and append one

round at the beginning and two rounds at the end for key recovery. Due to the subkey collision, we get the same values at $Z[0,2,5,7]$ and $\overline{Z}[0,2,5,7]$. This leads to two cells with zero difference in X_3. In case we have only 2 active cells at W_1, we get at least 3 active cells after the MixColumns due to the MDS property. This means that both columns at W_2 have at least one active cell, so we get a contradiction at this MixColumns operation.

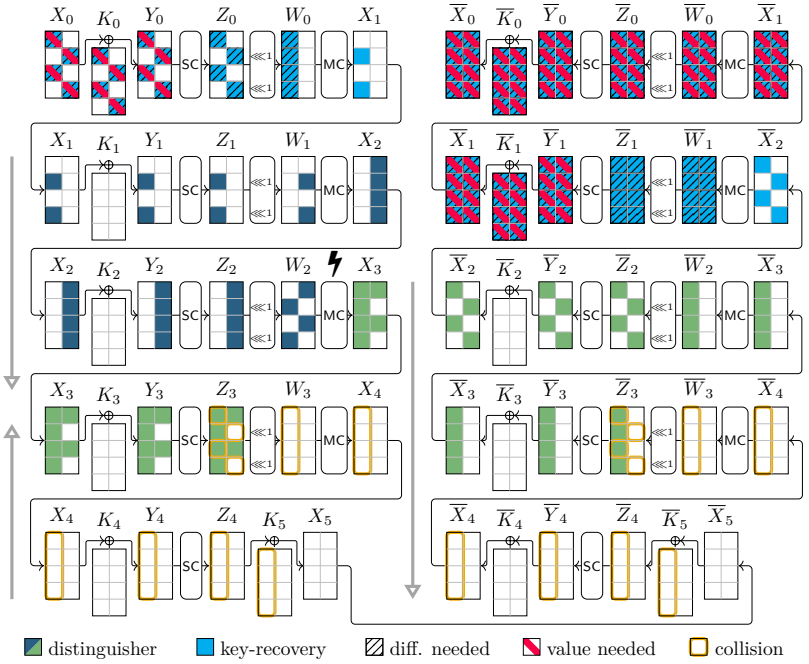


Figure 7: Impossible differential attack on 5+5 rounds. The distinguisher spans 4+3 rounds from X_1 to $\overline{X}_2$. Due to symmetries, there are two valid output masks to the distinguisher. $|k_b \cup k_f| = 80$, $c_b = 8$, $c_f = 15$, $|\Delta_b| = 16$, $|\Delta_f| = 32$. The gold frame indicates the 16-bit round key collision needed. Due to symmetries, we expect a collision after trying 2^{15} tweak pairs.

Key Recovery. We follow the framework by Boura et al. [BNS14] to mount a key recovery. The probability that a key guess gets filtered by our distinguisher is $2^{-c_b-c_f} = 2^{-8-15} = 2^{-23}$. In order to discard g bits of key material, we need at least $N = 2^{c_b+c_f} g \cdot ln(2) = 2^{23} g \cdot ln(2)$ differences at X_0.

First, we query D ciphertexts to build N differences at X_0. Next, we use these differences to filter the subkeys in two steps:

1. Guess K_0. Afterward we get an 8-bit filter from W_0 to X_1. This step takes $N \cdot 2^{16}$ time and leaves us with $N \cdot 2^{-8}$ data.

2. Guess $\overline{K}_0$ and $\overline{K}_1[0,2,5,7]$. Afterward we get a 7-bit filter (due to symmetries) from $\overline{W}_1$ to $\overline{X}_2$. This step takes $N \cdot 2^{-8} \cdot 2^{64}$ time and leaves us with $N \cdot 2^{-15}$ data.

3. Guess $\overline{K}_1[1,3,4,6]$. This step takes $N \cdot 2^{-15} \cdot 2^{80}$ time.

Complexity. We use $g = 35$ when minimizing the time complexity. Because there are enough inputs active, we can calculate the needed data to construct $N = 2^{23} \cdot 35 \cdot ln(2) = 2^{27.6}$ differences with $D = \sqrt{N \cdot 2} = \sqrt{2^{28.14+1}} = 2^{14.6}$.

The time complexity consists of querying the data, applying the sieving step, and brute-forcing the remaining key bits. Step 3 dominates the time complexity of the key sieving step. This step involves 4 out of 80 S-Boxes and therefore takes $N \cdot 2^{-15} \cdot 2^{80} \cdot 2^{-4.32} = N \cdot 2^{69.3}$ equivalent cipher encryptions. The time complexity is $T = D + N \cdot 2^{69.3} + 2^{128-g} = 2^{14.6} + 2^{27.6} \cdot 2^{69.3} + 2^{128-51} = 2^{96.9}$ encryptions, and the required memory is $N = 2^{27.6}$ block sizes. Since, on average, we need to repeat our attack for 2^{15} different tweaks, the total time complexity is $2^{96.9+15} = 2^{112}$ encryptions and we require $2^{14.6+15} = 2^{29.6}$ plaintext-ciphertext pairs.

5.4 Integral Cryptanalysis

We use a monomial prediction [HSWW20] SAT model to find integral distinguishers. To propagate the monomials through the MDS matrix of the MixColumns layer, we follow the method described by Hu et al. [HWW20]. Although this method perfectly models possible monomial transitions, it significantly increases the runtime of our program. Because of this, we are only able to detect the absence of monomial trails. As shown by Hu et al. [HSWW20] this proves that the monomial is not present in the Algebraic Normal Form (ANF). We can use this to construct an integral distinguisher. However, there might be a longer distinguisher, since we do not check the parity of the number of monomial trails.

For 5+5 rounds, the longest integral distinguisher we find with our model spans 5+1 rounds and requires 2^{31} data. In the key recovery 112 key-bits are involved. Even when using the partial-sum technique [FKL+00] or FFT [TA14], we do not expect it is possible to mount an attack with a time complexity below 2^{128}. For 4+4 rounds we show an integral attack in Subsection A.4.

5.5 Demirci-Selçuk MitM

We use a cell-based MILP model [SSD+18] to search for distinguisher of the Demirci-Selçuk MitM attack [DS08]. Due to the large key size in relation to the block size, we are able to find long distinguisher. Indeed, this is the only attack we found that spans 5+5 rounds without round-key collisions below the 128-bit security level. However, due to the large memory and time requirements, this attack does not threaten our practical security claims.

5.6 Non–U-Shape Attacks

As described in Subsection 2.1, in case the attacker has physical access to the memory and is able to read out the ciphertext, then we only need to attack the left branch of the U-shape (that is the encryption function). Compared to the U-shape attack model, it is not easily possible to query an arbitrary amount of tweak/subtweak-key pairs to recover the actual key from the tweak-key schedule. Because of this, the overall time complexity to attack the cipher given the time complexity to attack the data path T_d and the data D_t and time T_t to attack the tweak-key schedule is $T = T_d \cdot D_t + T_t$. The overall data complexity is $D = D_d \cdot D_t$. However, as shown in the attack presented in Subsection A.6, even when taking into account the tweak-key schedule, there are valid attacks on the cipher below the practical security claim. We therefore do not claim security in this setting.

6 Hardware Cost

We provide a comparison between BEANIE and other low-latency ciphers with smaller states, such as SCARF [CGL+23b], BipBip [BDD+23b], and ChiLow [BDG+25], as well as a comparison to the 64-bit state low-latency ciphers QARMAv2 [ABD+23] and PRINCE. Additionally, to substantiate our design rationale, we present performance estimates for different choices of the round function R in Table 14 in the appendix.

6.1 Comparison to Small-State Ciphers

In Table 11, we provide a hardware performance comparison with the small-state ciphers SCARF (10 bits) [CGL+23b], BipBip (24 bits) [BDD+23b], and ChiLow [BDG+25]. However, we emphasize that it is difficult to provide a truly meaningful comparison due to very different construction goals, target applications, and security assumptions. The technology for BEANIE and its TKS are done using the NanGate 15nm Open Chip Library (OCL) at 0.88 volts and 0 degrees. For other ciphers, we recall previously published results [ABD+23, BDG+25] which use the same 15nm OCL. The number of considered rounds for these ciphers is equal to the number of S-Box layers counted in their original design. For ChiLow, we only compare with the decryption and note that the encryption function is significantly slower.

From the security point of view, we note that SCARF and BipBip claim the same security level as BEANIE (2^{80} time and 2^{40} data complexity), while ChiLow claims 128-bit security allowing $\leq 2^8$ queries with the same tweak and $\leq 2^{40}$ queries for encryption and decryption [BDG+25, Claims 1 and 2].

Single Circuit. It is possible to implement the encryption and decryption of the datapath of BEANIE in a single circuit. Depending on the desired latency and technology, this can decrease the area of the circuit (see Table 11, split vs. single circuit). Combining the encryption and decryption works because $\mathsf{SR} \circ \mathsf{SC} = \mathsf{SC} \circ \mathsf{SR}$ and due to the involutive property of MC. Thus, we only need a multiplexer selecting either SC or its inverse and supply the keys in reverse order for the decryption.

Table 11: Comparison between BEANIE and other ciphers on NanGate 15nm technology. All performance numbers for SCARF, BipBip, Qarmav2, and ChiLow are taken from the given references. Since the latency and area numbers of our PRINCE and Orthros implementation matches quite closely with the one provided in these references, we expect the results to be well-comparable.

		Latency optimized			Area optimized		
Cipher	Block bits	Time ps	Area μ^2	Area GE	Time ps	Area μ^2	Area GE
SCARF [BDG+25]	10	217	1096	5577	298	969	4928
BipBip (11R Enc) [ABD+23]	24	683	10 334	52 561	1520	7031	35 760
BipBip (11R Dec) [ABD+23]	24	327	6554	33 843	710	5344	27 180
ChiLow-32 [BDG+25]	32	283	3582	18 218	302	3418	17 384
ChiLow-40 [BDG+25]	40	278	3205	16 303	299	3043	15 480
BEANIE (5R Enc)	32	192	677	3443	289	477	2424
BEANIE (5R Dec)	32	203	769	3910	293	473	2407
BEANIE (split circuit)	32	203	1446	7352	293	950	4831
BEANIE (single circuit)	32	232	1446	7353	467	810	4118
PRINCE (8 R)	64	251	1566	7964	362	1145	5823
PRINCE (10 R)	64	312	2083	10 592	446	1438	7316
PRINCE (12 R)	64	376	2464	12 531	537	1723	8762
QARMAv2-64-σ_0 ($r = 4$) [ABD+23]	64	305	2381	12 110	447	1583	8053
BEANIE TKS (5R)	128	212	2631	13 383	341	1862	9470
QARMAv2-128-128 ($r = 9$)	128	620	9052	46 043	925	6272	31 901
Orthros	128	356	7482	38 055	516	5811	29 555

6.2 Comparison to PRINCE

We now consider the comparison to larger 64-bit state ciphers, particularly PRINCE. Recall that our cipher is suitable for applications with 32-bit blocks whose ciphertext is not accessible and thus is set in a U-shape form (to speed up data path processing), while the 64-bit PRINCE would be used in a normal setting due to its larger state. In order to provide a somewhat fair comparison between these two schemes in terms of latency, we can take into consideration their security levels with respect to different attacks. For this reason, we provide numbers for multiple different configurations of PRINCE for 8 to 12 rounds. If we take proxy metrics such as the number of differentially active S-boxes and key-guessing complexity (with all the attached caveats of such a rough comparison for different block sizes), then 5+5-round BEANIE (latency 203 ps) compares roughly to 8-round or 10-round PRINCE [CXTQ24, Table 5] (latency of 251 and 312 ps).

7 Conclusion

In this work, we present a 32-bit blockcipher designed for memory encryption on MCUs. A distinguishing feature of our cipher is the complex tweak-key schedule realized in the form of a separate SPN cipher. The aim is to prevent easily inducible subtweak-key collisions that can compromise the security of the cipher in the U-shape attack setting, as well as to provide more resistance against general attacks on the cipher in tweak-related scenarios. Due to the small security margin, we conduct an in-depth security analysis of the new cipher design. The U-shape attack setting introduces some weakness in the diffusion properties of the cipher. However, since the U-shape comes from the attack setting, not the cipher itself, we can not employ measures like the reflector used in PRINCE and QARMA. This weakness in diffusion as well as colliding round keys are major reasons for new dedicated designs.

This cipher is optimized for low latency in hardware, which we substantiate by providing latency and area numbers. Since the round-key preparation and data processing by the cipher can be done in separate cycles (as explained in Section 2.2), the overall latency of our encryption system is mainly determined by the data path. We also compare these numbers with different cipher configurations. Finally, we provide a comprehensive hardware comparison to various comparable cipher designs, highlighting the favorable latency numbers of BEANIE. In conclusion, we believe that this makes our design more suitable for enabling memory encryption for embedded memories on MCUs.

Acknowledgments. This research was co-funded by the Austrian Science Fund (FWF) SFB project SPyCoDe (10.55776/F85), by the European Union (ERC Starting Grant KEYLESS, #101165216), and by the SINFONIA project, which has received funding from the Recovery and Resilience Facility (RRF) as the centrepiece of NextGenerationEU via the Austrian Research Promotion Agency (FFG) and Austria Wirtschaftsservice Gesellschaft mbH (aws) in the frame of the IPCEI ME/CT – Important Project of Common European Interest on Microelectronic and Communication Technologies, under FFG project No. 917423 and AWS project No. P2431566.

References

[ABB+21] Elham Amini, Kai Bartels, Christian Boit, Marius Eggert, Norbert Herfurth, Tuba Kiyan, Thilo Krachenfels, Jean-Pierre Seifert, and Shahin Tajik. Special session: Physical attacks through the chip backside: Threats, challenges, and opportunities. In *VTS 2021*, pages 1–12. IEEE, 2021. doi:10.1109/vts50974.2021.9441006.

[ABC+24] Ravi Anand, Subhadeep Banik, Andrea Caforio, Tatsuya Ishikawa, Takanori Isobe, Fukang Liu, Kazuhiko Minematsu, Mostafizar Rahman, and Kosei Sakamoto. Gleeok: A family of low-latency prfs and its applications to authenticated encryption. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, 2024(2):545–587, 2024. doi:10.46586/tches.v2024.i2.545-587.

[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*, 2023(3):25–73, 2023. doi:10.46586/tosc.v2023.i3.25-73.

[ADM24] Roberto Avanzi, Orr Dunkelman, and Kazuhiko Minematsu. MATTER: A wide-block tweakable block cipher. IACR Cryptology ePrint Archive, Paper 2024/1186, 2024. URL: https://eprint.iacr.org/2024/1186.

[AEBD+24] Parisa Amiri-Eliasi, Yanis Belkheyar, Joan Daemen, Santosh Ghosh, Daniël Kuijsters, Alireza Mehrdad, Silvia Mella, Shahram Rasoolzadeh, and Gilles Van Assche. Koala: A low-latency pseudorandom function. In *SAC 2024*, volume 15517 of *LNCS*, pages 239–266. Springer, 2024. doi:10.1007/978-3-031-82841-6_10.

[AJK+24] Elham Amini, Jörg Jatzkowski, Tuba Kiyan, Lars Renkes, Thilo Krachenfels, Shahin Tajik, Christian Boit, Frank Altmann, Sebastian Brand, and Jean-Pierre Seifert. Comparative study of E-beam and optical probing approaches in attacking the ICs. *Journal of Failure Analysis and Prevention*, 24:2184–2193, 2024. doi:10.1007/s11668-024-02019-0.

[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 Transactions on Symmetric Cryptology*, 2017(1):4–44, 2017. doi:10.13154/tosc.v2017.i1.4-44.

[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 *ASIACRYPT 2015*, volume 9453 of *LNCS*, pages 411–436. Springer, 2015. doi:10.1007/978-3-662-48800-3_17.

[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 *ASIACRYPT 2012*, volume 7658 of *LNCS*, pages 208–225. Springer, 2012. doi:10.1007/978-3-642-34961-4_14.

[BDD+23a] Yanis Belkheyar, Joan Daemen, Christoph Dobraunig, Santosh Ghosh, and Shahram Rasoolzadeh. Bipbip: A low-latency tweakable block cipher with small dimensions. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, 2023(1):326–368, 2023. doi:10.46586/tches.v2023.i1.326-368.

[BDD+23b] Yanis Belkheyar, Joan Daemen, Christoph Dobraunig, Santosh Ghosh, and Shahram Rasoolzadeh. Bipbip: A low-latency tweakable block cipher with small dimensions. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, 2023(1):326–368, 2023. doi:10.46586/tches.v2023.i1.326-368.

[BDG+25] Yanis Belkheyar, Patrick Derbez, Shibam Ghosh, Gregor Leander, Silvia Mella, Léo Perrin, Shahram Rasoolzadeh, Lukas Stennes, Siwei Sun, Gilles Van Assche, and Damian Vizár. Chilow and chichi: New constructions for code encryption. In *EUROCRYPT 2025*, volume 15601 of *LNCS*, pages 212–243. Springer, 2025. doi:10.1007/978-3-031-91107-1_8.

[BDK01] Eli Biham, Orr Dunkelman, and Nathan Keller. The Rectangle attack – rectangling the Serpent. In *EUROCRYPT 2001*, volume 2045 of *LNCS*, pages 340–357. Springer, 2001. doi:10.1007/3-540-44987-6_21.

[BEK+20] Dušan Božilov, Maria Eichlseder, Miroslav Knežević, Baptiste Lambin, Gregor Leander, Thorben Moos, Ventzislav Nikov, Shahram Rasoolzadeh, Yosuke Todo, and Friedrich Wiemer. PRINCEv2 – more security for (almost) no overhead. In *SAC 2020*, volume 12804 of *LNCS*, pages 483–511. Springer, 2020. doi:10.1007/978-3-030-81652-0_19.

[BIL+21] Subhadeep Banik, Takanori Isobe, Fukang Liu, Kazuhiko Minematsu, and Kosei Sakamoto. Orthros: A low-latency PRF. *IACR Transactions on Symmetric Cryptology*, 2021(1):37–77, 2021. doi:10.46586/tosc.v2021.i1.37-77.

[Bir03] Alex Biryukov. Analysis of involutional ciphers: Khazad and Anubis. In *FSE 2003*, volume 2887 of *LNCS*, pages 45–53. Springer, 2003. doi:10.1007/978-3-540-39887-5_5.

[BNS14] Christina Boura, María Naya-Plasencia, and Valentin Suder. Scrutinizing and improving impossible differential attacks: Applications to CLEFIA, Camellia, LBlock and Simon. In *ASIACRYPT 2014*, volume 8873 of *LNCS*, pages 179–199. Springer, 2014. doi:10.1007/978-3-662-45611-8_10.

[CGL+23a] Federico Canale, Tim Güneysu, Gregor Leander, Jan Philipp Thoma, Yosuke Todo, and Rei Ueno. SCARF – A low-latency block cipher for secure cache-randomization. In *USENIX Security 2023*, pages 1937–1954. USENIX Association, 2023. URL: https://www.usenix.org/conference/usenixsecurity23/presentation/canale.

[CGL+23b] Federico Canale, Tim Güneysu, Gregor Leander, Jan Philipp Thoma, Yosuke Todo, and Rei Ueno. SCARF – A low-latency block cipher for secure cache-randomization. In *USENIX Security 2023*, pages 1937–1954. USENIX Association, 2023. URL: https://www.usenix.org/conference/usenixsecurity23/presentation/canale.

[cor] CoreMark CPU benchmark. URL: https://eembc.org/coremark/.

[CSQ07] Baudoin Collard, François-Xavier Standaert, and Jean-Jacques Quisquater. Improving the time complexity of Matsui's linear cryptanalysis. In *ICISC 2007*, volume 4817 of *LNCS*, pages 77–88. Springer, 2007. doi:10.1007/978-3-540-76788-6_7.

[CSW16] Franck Courbon, Sergei Skorobogatov, and Christopher Woods. Reverse engineering flash EEPROM memories using scanning electron microscopy. In *CARDIS 2016*, volume 10146 of *LNCS*, pages 57–72. Springer, 2016. doi:10.1007/978-3-319-54669-8_4.

[CV3] OpenHW Group CORE-V CV32E40P RISC-V IP. URL: https://github.com/openhwgroup/cv32e40p.

[CXTQ24] Yaxin Cui, Hong Xu, Lin Tan, and Wenfeng Qi. Automatic search of differential characteristics and improved differential cryptanalysis for PRINCE, QARMA, and MANTIS. *IET Inf. Secur.*, 2024(1), 2024. doi:10.1049/2024/5574862.

[Dae91] Joan Daemen. Limitations of the Even-Mansour construction. In *ASIACRYPT '91*, volume 739 of *LNCS*, pages 495–498. Springer, 1991. doi:10.1007/3-540-57332-1_46.

[DDK+23] Itai Dinur, Orr Dunkelman, Nathan Keller, Eyal Ronen, and Adi Shamir. Efficient detection of high probability statistical properties of cryptosystems via surrogate differentiation. In *EUROCRYPT 2023*, volume 14007 of *LNCS*, pages 98–127. Springer, 2023. doi:10.1007/978-3-031-30634-1_4.

[DEKM16] Christoph Dobraunig, Maria Eichlseder, Daniel Kales, and Florian Mendel. Practical key-recovery attack on MANTIS5. *IACR Transactions on Symmetric Cryptology*, 2016(2):248–260, 2016. doi:10.13154/tosc.v2016.i2.248-260.

[DF13] Patrick Derbez and Pierre-Alain Fouque. Exhausting Demirci-Selçuk meet-in-the-middle attacks against reduced-round AES. In *FSE 2013*, volume 8424 of *LNCS*, pages 541–560. Springer, 2013. doi:10.1007/978-3-662-43933-3_28.

[DKS10] Orr Dunkelman, Nathan Keller, and Adi Shamir. Improved single-key attacks on 8-round AES-192 and AES-256. In *ASIACRYPT 2010*, volume 6477 of *LNCS*, pages 158–176. Springer, 2010. doi:10.1007/978-3-642-17373-8_10.

[DKS14] Orr Dunkelman, Nathan Keller, and Adi Shamir. A practical-time related-key attack on the KASUMI cryptosystem used in GSM and 3G telephony. *Journal of Cryptology*, 27(4):824–849, 2014. doi:10.1007/s00145-013-9154-9.

[DS08] Hüseyin Demirci and Ali Aydin Selçuk. A meet-in-the-middle attack on 8-round AES. In *FSE 2008*, volume 5086 of *LNCS*, pages 116–126. Springer, 2008. doi:10.1007/978-3-540-71039-4_7.

[ELR20] Maria Eichlseder, Gregor Leander, and Shahram Rasoolzadeh. Computing expected differential probability of (truncated) differentials and expected linear potential of (multidimensional) linear hulls in SPN block ciphers. In Karthikeyan Bhargavan, Elisabeth Oswald, and Manoj Prabhakaran, editors, *INDOCRYPT 2020*, volume 12578 of *LNCS*, pages 345–369. Springer, 2020. doi:10.1007/978-3-030-65277-7_16.

[FGL+24] Antonio Flórez-Gutiérrez, Lorenzo Grassi, Gregor Leander, Ferdinand Sibleyras, and Yosuke Todo. General practical cryptanalysis of the sum of round-reduced block ciphers and ZIP-AES. In *ASIACRYPT 2024*, volume 15492 of *LNCS*, pages 280–311. Springer, 2024. doi:10.1007/978-981-96-0947-5_10.

[FGLL+25] Antonio Flórez-Gutiérrez, Eran Lambooij, Gaëtan Leurent, Håvard Raddum, Tyge Tiessen, and Michiel Verbauwhede. Cryptanalysis of full SCARF. In *EUROCRYPT 2025*, volume 15601 of *LNCS*, pages 397–426. Springer, 2025. doi:10.1007/978-3-031-91107-1_14.

[FKL+00] Niels Ferguson, John Kelsey, Stefan Lucks, Bruce Schneier, Michael Stay, David A. Wagner, and Doug Whiting. Improved cryptanalysis of Rijndael. In *FSE 2000*, volume 1978 of *LNCS*, pages 213–230. Springer, 2000. doi:10.1007/3-540-44706-7_15.

[GMW24] Patricia Greene, Mark Motley, and Bryan Weeks. ARADI and LLAMA: low-latency cryptography for memory encryption. IACR Cryptology ePrint Archive, Paper 2024/1240, 2024. URL: https://eprint.iacr.org/2024/1240.

[GSP+23] Lukas Giner, Stefan Steinegger, Antoon Purnal, Maria Eichlseder, Thomas Unterluggauer, Stefan Mangard, and Daniel Gruss. Scatter and split securely: Defeating cache contention and occupancy attacks. In *SP 2023*, pages 2273–2287. IEEE, 2023. doi:10.1109/sp46215.2023.10179440.

[HGSE24] Hosein Hadipour, Simon Gerhalter, Sadegh Sadeghi, and Maria Eichlseder. Improved search for integral, impossible differential and zero-correlation attacks application to Ascon, ForkSKINNY, SKINNY, MANTIS, PRESENT and QARMAv2. *IACR Transactions on Symmetric Cryptology*, 2024(1):234–325, 2024. doi:10.46586/tosc.v2024.i1.234-325.

[HKPT24] Kai Hu, Mustafa Khairallah, Thomas Peyrin, and Quan Quan Tan. uknit: Breaking round-alignment for cipher design – featuring uknit-bc, an ultra low-latency block cipher. IACR Cryptology ePrint Archive, Paper 2024/1962, 2024. URL: https://eprint.iacr.org/2024/1962.

[HSWW20] Kai Hu, Siwei Sun, Meiqin Wang, and Qingju Wang. An algebraic formulation of the division property: Revisiting degree evaluations, cube attacks, and key-independent sums. In *ASIACRYPT 2020*, volume 12491 of *LNCS*, pages 446–476. Springer, 2020. doi:10.1007/978-3-030-64837-4_15.

[HWW20] Kai Hu, Qingju Wang, and Meiqin Wang. Finding bit-based division property for ciphers with complex linear layers. *IACR Transactions on Symmetric Cryptology*, 2020(1):396–424, 2020. doi:10.13154/tosc.v2020.i1.396-424.

[JPST17a] Jérémy Jean, Thomas Peyrin, Siang Meng Sim, and Jade Tourteaux. Optimizing implementations of lightweight building blocks. *IACR Transactions on Symmetric Cryptology*, 2017(4):130–168, 2017. doi:10.13154/tosc.v2017.i4.130-168.

[JPST17b] Jérémy Jean, Thomas Peyrin, Siang Meng Sim, and Jade Tourteaux. Optimizing implementations of lightweight building blocks. *IACR Transactions on Symmetric Cryptology*, 2017(4):130–168, 2017. doi:10.13154/tosc.v2017.i4.130-168.

[KLS+20] Thilo Krachenfels, Heiko Lohrke, Jean-Pierre Seifert, Enrico Dietz, Sven Frohmann, and Heinz-Wilhelm Hübers. Evaluation of low-cost thermal laser stimulation for data extraction and key readout. *J. Hardw. Syst. Secur.*, 4(1):24–33, 2020. doi:10.1007/s41635-019-00083-9.

[KR11] Ted Krovetz and Phillip Rogaway. The software performance of authenticated-encryption modes. In *FSE 2011*, volume 6733 of *LNCS*, pages 306–327. Springer, 2011. doi:10.1007/978-3-642-21702-9_18.

[LMMR21] Gregor Leander, Thorben Moos, Amir Moradi, and Shahram Rasoolzadeh. The SPEEDY family of block ciphers engineering an ultra low-latency cipher from gate level for secure processor architectures. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, 2021(4):510–545, 2021. doi:10.46586/tches.v2021.i4.510-545.

[LTBS16] Heiko Lohrke, Shahin Tajik, Christian Boit, and Jean-Pierre Seifert. No place to hide: Contactless probing of secret data on fpgas. IACR Cryptology ePrint Archive, Paper 2016/593, 2016. URL: http://eprint.iacr.org/2016/593.

[Mat93] Mitsuru Matsui. Linear cryptanalysis method for DES cipher. In *EUROCRYPT '93*, volume 765 of *LNCS*, pages 386–397. Springer, 1993. doi:10.1007/3-540-48285-7_33.

[MH22] Jubayer Mahmod and Matthew Hicks. SRAM has no chill: exploiting power domain separation to steal on-chip secrets. In *ASPLOS '22*, pages 1043–1055. ACM, 2022. doi:10.1145/3503222.3507710.

[NSUH22] Shoei Nashimoto, Daisuke Suzuki, Rei Ueno, and Naofumi Homma. Bypassing isolated execution on RISC-V using side-channel-assisted fault-injection and its countermeasure. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, 2022(1):28–68, 2022. doi:10.46586/tches.v2022.i1.28-68.

[NSW+21] Pascal Nasahl, Robert Schilling, Mario Werner, Jan Hoogerbrugge, Marcel Medwed, and Stefan Mangard. Cryptag: Thwarting physical and logical memory vulnerabilities using cryptographically colored memory. In *CCS '21*, pages 200–212. ACM, 2021. doi:10.1145/3433210.3453684.

[Saa11] Markku-Juhani O. Saarinen. Cryptographic analysis of all 4 × 4-bit S-boxes. In *SAC 2011*, volume 7118 of *LNCS*, pages 118–133. Springer, 2011. doi:10.1007/978-3-642-28496-0_7.

[SKOP15] Siang Meng Sim, Khoongming Khoo, Frédérique E. Oggier, and Thomas Peyrin. Lightweight MDS involution matrices. In *FSE 2015*, volume 9054 of *LNCS*, pages 471–493. Springer, 2015. doi:10.1007/978-3-662-48116-5_23.

[SSD+18] Danping Shi, Siwei Sun, Patrick Derbez, Yosuke Todo, Bing Sun, and Lei Hu. Programming the Demirci-Selçuk meet-in-the-middle attack with constraints. In *ASIACRYPT 2018*, volume 11273 of *LNCS*, pages 3–34. Springer, 2018. doi:10.1007/978-3-030-03329-3_1.

[SSW+21] Stefan Steinegger, David Schrammel, Samuel Weiser, Pascal Nasahl, and Stefan Mangard. SERVAS! secure enclaves via RISC-V authenticryption shield. *CoRR*, abs/2105.03395, 2021. URL: https://arxiv.org/abs/2105.03395, arXiv:2105.03395.

[TA14] Yosuke Todo and Kazumaro Aoki. FFT key recovery for integral attack. In *CANS 2014*, volume 8813 of *LNCS*, pages 64–81. Springer, 2014. doi:10.1007/978-3-319-12280-9_5.

[Wag99] David A. Wagner. The boomerang attack. In *FSE '99*, volume 1636 of *LNCS*, pages 156–170. Springer, 1999. doi:10.1007/3-540-48519-8_12.

[WHG+21] Senpeng Wang, Bin Hu, Jie Guan, Tairong Shi, and Kai Zhang. Research on the security criterion of S-boxes against division property. *Chinese Journal of Electronics*, 30(1):85–91, 2021. doi:10.1049/cje.2020.10.013.

[WHWL24] Jianhua Wang, Tao Huang, Shuang Wu, and Zilong Liu. Twinkle: A family of low-latency schemes for authenticated encryption and pointer authentication. *IACR Communications in Cryptology*, 1(2):20, 2024. doi:10.62056/a3n59qgxq.

[ZLG24] Xiao Mei Zeng, Qing Liu, and Chee Lip Gan. A comprehensive data retrieval and correction approach from 40-nm flash memory with selective chemical engraving. *IEEE Trans. Inf. Forensics Secur.*, 19:1031–1040, 2024. doi:10.1109/tifs.2023.3327857.

A Security Analysis

A.1 S-Box Properties

In Table 12b we provide the difference distribution table (DDT) of the S-Box shown in Table 12a. Furthermore, Table 12c shows the linear approximation table (LAT). To conduct monomial prediction, we also provide the monomial prediction table (MPT) in Table 13.

Table 12: Difference Distribution Table and Linear Approximation Table of S-Box.

(a) The 4-bit S-Box of BEANIE.

x	0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
$S(x)$	0	4	2	b	a	c	9	8	5	f	d	3	7	1	6	e
$S^{-1}(x)$	0	d	2	b	1	8	e	c	7	6	4	3	5	a	f	9

(b) Difference Distribution Table (DDT)

I \ O	0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
0	16	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
1	.	2	.	.	2	.	4	.	2	2	2	.	.	.	2	.
2	.	2	2	2	2	.	.	.	2	.	.	.	2	.	.	4
3	.	.	4	.	.	2	4	2	.	2	.	2	.	.	.	.
4	.	.	2	2	.	.	.	.	2	.	2	4	.	2	2	.
5	.	.	2	2	2	2	.	.	2	.	2	.	2	.	2	.
6	.	2	2	2	.	.	.	2	2	2	2	.	2	.	.	.
7	.	2	.	.	2	.	.	.	2	2	.	2	2	2	2	.
8	.	.	.	.	.	2	2	.	2	.	.	2	.	4	.	4
9	.	4	.	.	.	.	2	2	.	.	.	4	.	.	2	2
a	.	.	2	.	2	.	.	4	.	2	.	.	2	2	2	.
b	.	.	.	2	2	.	.	.	2	2	2	.	.	2	2	2
c	.	.	.	2	4	4	.	2	.	.	.	2	.	.	.	2
d	.	2	.	2	.	4	.	.	.	2	2	.	2	2	.	.
e	.	.	.	.	.	2	2	4	.	.	4	.	2	.	.	2
f	.	2	2	2	.	.	2	.	.	2	.	.	2	2	2	.

(c) Linear Approximation Table (LAT)

I \ O	0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
0	8	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
1	.	.	.	-4	.	4	.	.	2	2	-2	2	-2	2	2	2
2	.	.	2	2	-2	2	.	-4	2	-2	4	.	.	.	2	2
3	.	.	-2	2	2	2	.	4	.	-4	2	2	-2	2	.	.
4	.	-2	.	2	.	2	-4	2	2	4	2	.	2	.	-2	.
5	.	2	4	-2	.	2	.	2	.	-2	.	-2	4	2	.	-2
6	.	2	2	.	-2	.	4	2	.	2	2	.	-2	.	-4	2
7	.	-2	2	.	2	-4	.	2	2	.	.	-2	.	2	2	4
8	.	4	2	2	4	.	-2	-2	-2	2	.	.	-2	2	.	.
9	.	.	-2	-2	4	.	2	-2	.	.	2	2	4	.	-2	2
a	.	4	.	.	-2	-2	-2	2	.	.	.	4	2	-2	2	2
b	.	.	.	.	2	2	2	2	-2	2	2	-2	.	-4	4	.
c	.	2	-2	.	.	-2	2	.	4	2	2	.	.	2	2	-4
d	.	2	-2	4	.	2	2	.	2	.	-4	-2	2	.	.	2
e	.	-2	4	2	2	.	2	.	2	.	-2	4	.	-2	.	-2
f	.	-2	.	2	-2	.	2	.	-4	2	.	2	2	4	2	.

A.2 Boomerang Attack on 4+4 rounds

In Figure 8 we display a 2+4 round boomerang distinguisher with a probability of $\approx 2^{-27}$. It is not possible to add a differential at the beginning of E_m, since the resulting distinguisher would have a probability smaller than 2^{-n}. Given the distinguisher from Figure 8, we append 2 rounds at the beginning to mount a chosen-plaintext adaptive-ciphertext attack on $4 + 4$ rounds of the cipher.

Key Recovery. First, we construct a set of 2^{29} different plaintexts P. We then let the oracle encrypt P to obtain the corresponding C. For each C_i we calculate $C'_i = C_i \oplus \overline{X}_0$. The oracle then gives us the corresponding plaintext P'_i. We store the 2^{29} pairs (P_i, P'_i).

Next, we guess all key cells in K_0 and store the values at state X_1 calculated from each P_i, P'_i in a hashtable. In this table we use the constant cells in X_1 as keys. This has the effect that each entry with the same key follows the constraint imposed by the distinguisher. Next, we guess K_1 for each element with the same key and again store it in a hashtable. If a pair of states calculate from P_i and P'_i have the same key in the hashtable

Table 13: Monomial Prediction Table of S-Box. This table shows the possible transitions between the input monomial I and the output monomial O.

I \ O	0	1	2	4	8	3	5	6	9	a	c	7	b	d	e	f
0	×	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
1	.	.	.	×	.	.	.	.	.	.	.	.	.	.	.	.
2	.	.	×	.	.	.	.	.	.	.	.	.	.	.	.	.
4	.	.	×	.	×	.	.	.	.	×	.	.	.	.	.	.
8	.	×	.	×	.	.	×	.	.	.	.	.	.	.	.	.
3	.	×	.	×	×	×	.	.	×	×	.	.	×	.	.	.
5	.	.	×	.	.	.	.	.	.	×	×	.	.	.	.	.
6	.	×	.	.	.	.	.	.	×	×	.	.	.	.	.	.
9	.	.	×	×	×	×	.	×	×	×	×	×	×	×	×	×
a	.	.	×	.	×	.	.	.	×	.	×	.	.	×	.	.
c	.	.	.	.	×	×	.	×	.	×	.	×	.	.	.	.
7	.	.	×	.	×	×	.	.	.	.	×	.	×	.	.	.
b	.	×	.	.	×	×	×	×	×	.	.	×	.	.	×	×
d	.	.	×	×	×	.	×	.	×	.	.	.	×	×	×	×
e	.	.	.	.	×	×	×	.	.	×	×	×	.	×	.	.
f	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	×

we increment a counter. This is the case if they adhere to the filter at state X_2. After repeating this process for all key candidates, we expect that the counter corresponding to the right key guess has the highest value.

Complexity. Since the filter at X_2 has 28 bits, we expect that a difference calculated from two (P_i, P_i') fulfills this property for a wrong key guess with a probability of $2^{-2 \cdot 28} = 2^{-56}$. In case of using our distinguisher, which holds with a probability of $\approx 2^{-27}$, we instead fulfill the property at X_2 with about $2^{-28-27} = 2^{-55}$. From the 2^{29} pairs (P_i, P_i') we obtain $2^{2 \cdot 29 - 1} = 2^{57}$ differences. For a right key guess, we therefore expect $2^{57-55} = 2^2$ differences left after the filter, compared to $2^{57-56} = 2^1$ for a wrong key guess. Depending on the targeted success probability and the advantage we have to repeat the whole procedure. Calculating the table at X_1 involves the 2^{32} keys in K_0 and 2^{29} data. The time complexity of this step is therefore 2^{61}, and we need to store 2^{29} pairs in memory. Due to the 2^{16} bit filter in X_1 we only need $2^{13+48} = 2^{61}$ time to calculate the table containing the values at X_2. The overall time complexity is therefore lower bounded by $2 \cdot 2^{61} = 2^{62}$. Due to the low signal-to-noise ratio of our distinguisher, we need to repeat this procedure multiple times. The data complexity is 2^{29} and the memory complexity is $2 \cdot 2^{29} = 2^{30}$ for storing the pairs (P_i, P_i').

A.3 Impossible Differential Attack

We use this distinguisher to mount an impossible differential attack on 4+4 rounds. This entails adding 2 rounds at the end of the distinguisher for the key recovery.

Key Recovery. We follow the framework by Boura et al. [BNS14] to mount a key recovery. The probability that a key guess gets filtered by our distinguisher is $2^{-c_b - c_f} = 2^{-15}$. In order to discard g bits of key material, we need at least $N = 2^{c_b + c_f} g \cdot ln(2) = 2^{15} g \cdot ln(2)$ differences at $\overline{X_0}$.

First, we query D ciphertexts to build N differences at $\overline{X_0}$. Next, we use these differences to filter the subkeys in two steps:

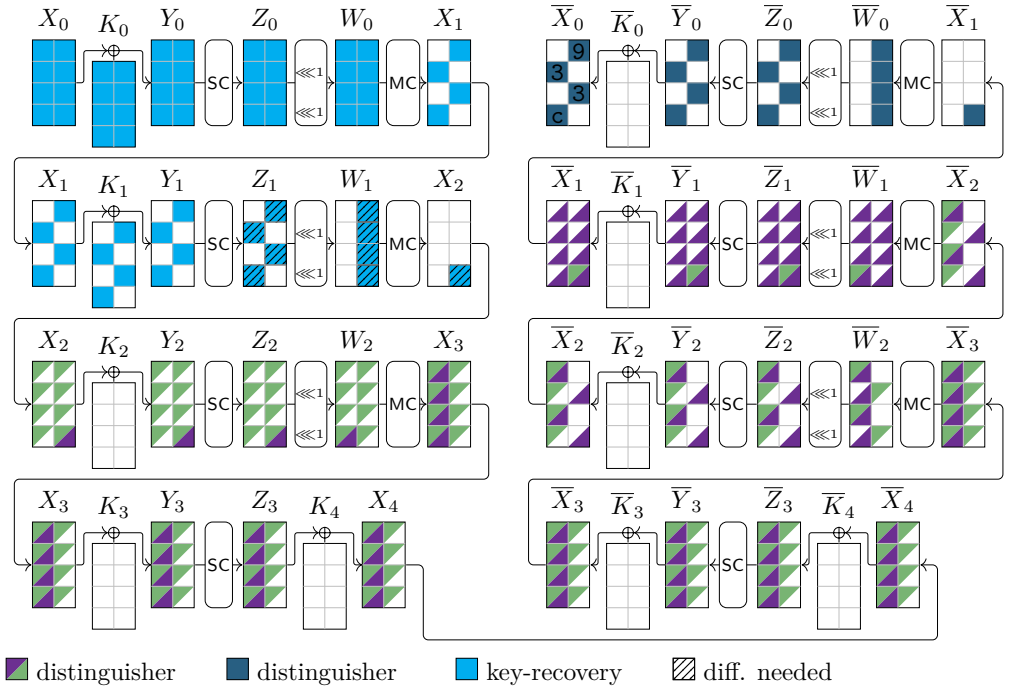


Figure 8: Boomerang Attack on 4+4 rounds. The distinguisher spans over 2+4 rounds. The middle part of the sandwich has a probability of $r = 2^{-13.6}$ and spans from X_2 to $\overline{X}_1$. We omit adding a differential in front of the middle part. The second differential of the boomerang distinguisher spans from $\overline{X}_0$ to $\overline{X}_1$ and has 4 active S-Boxes each with a probability of 2^{-2}. The total probability of the boomerang differential distinguisher when setting a fixed difference at the input of the middle part is therefore $2^{-13.6-4\cdot2\cdot2} = 2^{-29.6}$. Since we can use a truncated difference at the input of the middle part we experimentally found that the probability of the boomerang distinguisher is lower at $\approx 2^{-27}$. The used differences of the distinguisher are displayed in $\overline{Y}_0$.

1. Guess $\overline{K}_0$ and $\overline{K}_1[0, 2, 5, 7]$. Afterward we get an 7-bit filter from $\overline{W}_1$ to $\overline{X}_2$. This step takes $N \cdot 2^{48}$ time and leaves us with $N \cdot 2^{-7}$ data.

2. Guess $\overline{K}_1[1, 3, 4, 6]$. This step takes $N \cdot 2^{-7} \cdot 2^{64}$ time.

Complexity. We use $g = 60$ when minimizing the time complexity. Because there are enough inputs active, we can calculate the needed data to construct $N = 2^{15} \cdot 60 \cdot ln(2) = 2^{20.38}$ differences with $D = \sqrt{N \cdot 2} = \sqrt{2^{20.38+1}} = 2^{10.7}$.

The time complexity consists of querying the data, applying the sieving step and brute-forcing the remaining key bits. Step 2 dominates the time complexity of the key sieving step. This step involves 4 out of 64 S-Boxes and therefore takes $N \cdot 2^{-7} \cdot 2^{64} \cdot 2^{-4} = N \cdot 2^{53}$ equivalent cipher encryptions. The total time complexity is $T = D + N \cdot 2^{53} + 2^{128-g} = 2^{10.7} + 2^{20.38} \cdot 2^{53} + 2^{128-60} = 2^{73.4}$ and the required memory is $N = 2^{20.38}$.

A.4 Integral Attack

From transforming the $4 + 2$ round zero-correlation distinguisher (see Subsection 5.3) to an integral distinguisher we can mount the integral attack on 4+4 rounds displayed in Figure 10. We also confirm this distinguisher with our monomial prediction model.

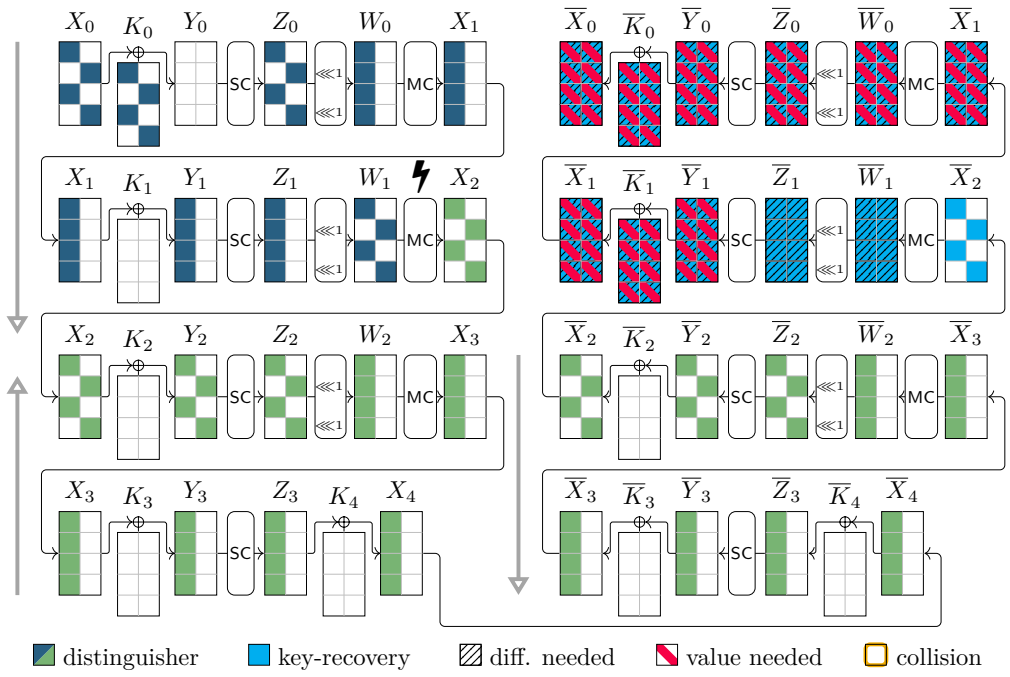


Figure 9: Impossible differential Attack on 4+4 rounds. The distinguisher spans 4+2 rounds from X_0 to $\overline{X}_2$. Due to symmetries, there are two valid input and output masks to the distinguisher. This changes the filter c_f and Δ_b by one bit. $|k_b \cup k_f| = 64$, $c_b = 0$, $c_f = 15$, $|\Delta_b| = 17$, $|\Delta_f| = 32$.

Interestingly, we are not able to find longer integral distinguisher with our monomial prediction model.

Key Recovery. The integral attack displayed in Figure 10 has a data complexity of 2^{16} since 4 cells are active at the input. We append two rounds at the end of the distinguisher. In total 48 bits of the subkeys are involved in the key recovery. From $\overline{W_0}$ to $\overline{X_1}$ the state only gets reduced to 2^{16}. This is the same as the data size, we therefore do not gain anything from using the partial-sum technique. Hence, the time complexity is $2^{16+48} = 2^{64}$, and we do not need to store any data in memory. Since we can check the balanced property at $\overline{X_2}[0]$ and $\overline{X_2}[2]$ we get a 8-bit filter.

Complexity. We can now repeat this procedure 8 times. Note that we can reuse the same data twice, since we have to balanced property in both columns at $\overline{X_2}$. In the end we have a data complexity of 2^{18} and a time complexity for the filter step of $8 \cdot 2^{64} = 2^{67}$ and $2^{128-8\cdot 8} = 2^{64}$ to brute-force the remaining keys.

A.5 Demirci-Selçuk MitM Attack

We use our tool to mount an attack on 5+5 rounds of BEANIE, as outlined in Figure 11. The distinguisher maps a 8-bit input ($Y_1[5,7]$) to a 4-bit output ($\overline{Y}_2[5]$). This mapping depends on 104 bits of the intermediate states. In the offline phase, we construct a hashtable with an entry for each of these 2^{104} possible values of the intermediate states. Each entry in the hashtable contains the function outputs for all 256 possible inputs. In the offline phase we therefore need to calculate $2^{104} \cdot 256 = 2^{112}$ partial encryptions.

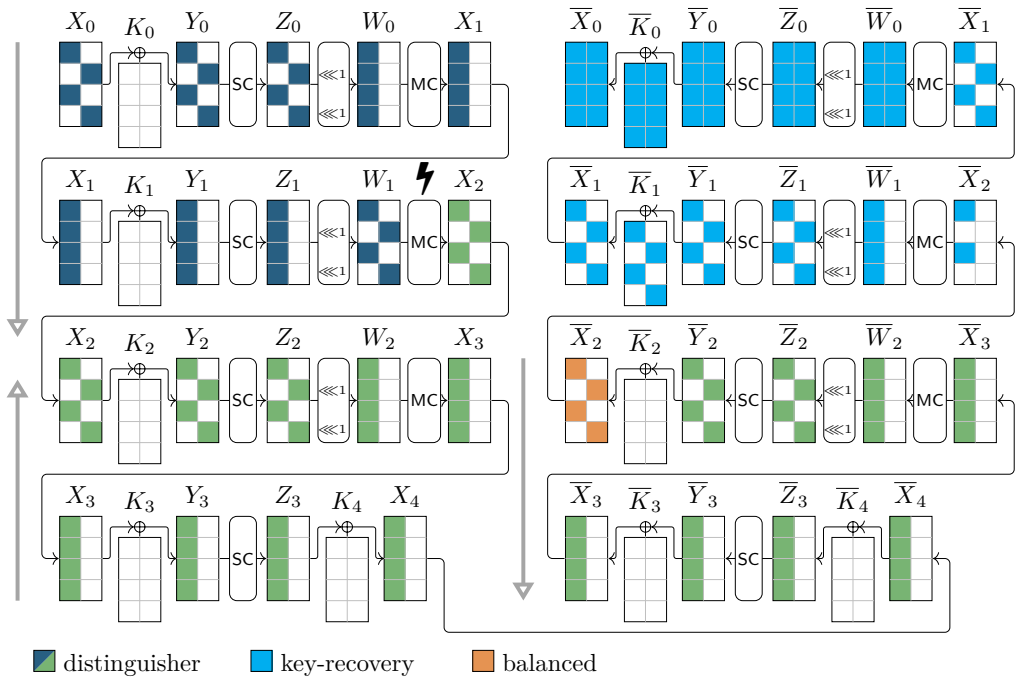


Figure 10: Integral Attack on 4+4 rounds. The distinguisher spans 4+2 rounds, starting at X_0 and ending in 4 balanced cells at $\overline{X}_2$. Keys in $\overline{K}_0$ and $\overline{K}_1$ are involved in the key recovery.

Key Recovery. Given the hashtable calculated in the offline phase, we mount a key recovery by the following steps.

1. Gather 2^{16} data (Active cells at X_0).
2. For an abitrary plaintext guess K_0 and K_1 to get state X_1.
3. Based on this state, we construct all possible inputs to the distinguisher (2^8).
4. With the key guess of Step 2 decrypt this set of distinguisher inputs backwards to find out which 2^8 plaintexts of the 2^{16} data we use for the next steps.
5. For this set of plaintexts we decrypt the corresponding ciphertexts backward from $\overline{X}_0$ with all possible keys of $\overline{K}_0$ $\overline{K}_1$ $\overline{K}_2$.
6. For each key guess we compare the state $\overline{X}_2$ with the values of the hashtable.
7. If they match, we found the right keys with an overwhelming probability, otherwise try different keys.

Complexity. The attack has a data complexity of 2^{16}. When scaling the memory to the block size, the hashtable consumes $2^{104} \cdot 256 \cdot 4/32 = 2^{109}$ block size equivalents. The time complexity of the offline phase is 2^{112} partial encryptions and of the online phase $256 \cdot 2^{84} = 2^{92}$ partial encryptions. When doing a time-memory trade off, using a multiset [DKS10], using the differential enumeration technique [DKS10], exploit the MDS matrix of the MixColumns [DF13] or other optimizations, we expect further improvements on the complexities.

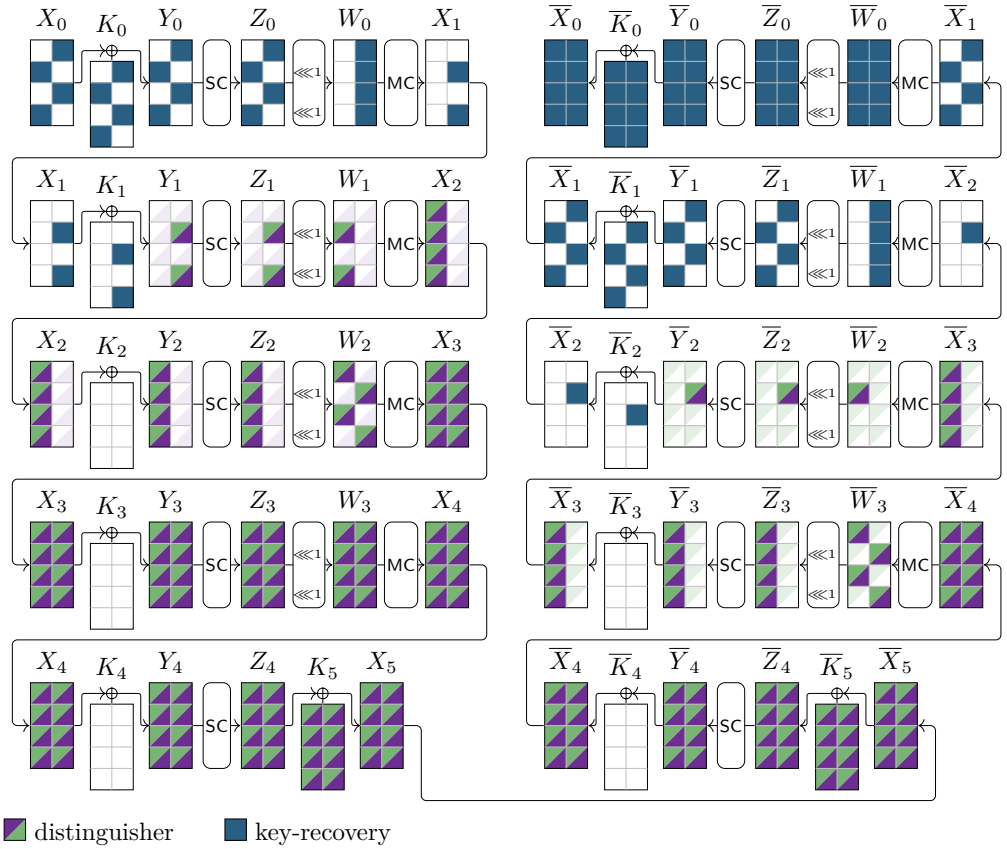


Figure 11: The distinguisher (green and purple cells) spans from Z_1 to $\overline{Z_2}$. If we store differences in the hash table we do not need to guess the state at $\overline{Z_2}$. Furthermore, K_5 and $\overline{K_5}$ are linearly dependent, so we only need to guess 32-bit for these key additions. In total, we need to guess the state at Z_1 Z_2, Z_3, Z_4 and $\overline{Z_3}$ in the offline phase. This amounts to a hash table with 2^{104} entries. The online phase (blue cells) consists of guessing 2^{84} key bits at K_0, K_1, $\overline{K_2}$, $\overline{K_1}$, $\overline{K_0}$.

A.6 Impossible-Differential attack on Non U-shape

We use the techniques described in Subsection A.3 to find the 4 round impossible differential distinguisher displayed in Figure 12. For the key recovery we append one round at the beginning.

Key Recovery. We follow the framework by Boura et al. [BNS14] to mount a key recovery. The probability that a key guess gets filtered by our distinguisher is $2^{-c_b-c_f} = 2^{-15}$. In order to discard g bits of key material, we need at least $N = 2^{c_b+c_f} g \cdot ln(2) = 2^{15} g \cdot ln(2)$ differences at X_0.

First, we query D ciphertexts to build N differences at X_0. Next, we use these differences to filter the subkeys in two steps:

1. Guess $K_0[0, 2, 5, 7]$. Afterward we get an 7-bit filter from W_0 to X_1. This step takes $N \cdot 2^{16}$ time and leaves us with $N \cdot 2^{-7}$ data.

2. Guess $K_0[1, 3, 4, 6]$. This step takes $N \cdot 2^{-7} \cdot 2^{32}$ time.

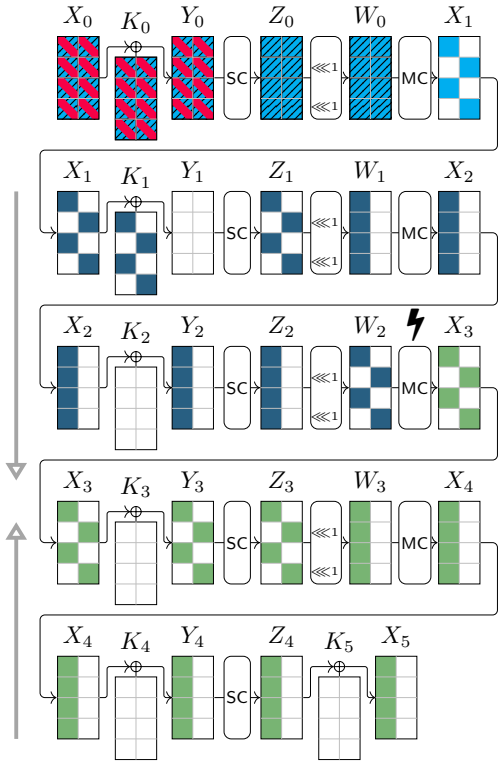


Figure 12: Impossible differential Attack on 5 rounds. The distinguisher spans 4 rounds from X_1 to X_5. Due to symmetries, there are two valid input and output masks to the distinguisher. This changes the filter c_f and Δ_b by one bit. $|k_b \cup k_f| = 32$, $c_b = 0$, $c_f = 15$, $|\Delta_b| = 32$, $|\Delta_f| = 17$.

Complexity. We use $g = 90$ when minimizing the time complexity. Because there are enough inputs active, we can calculate the needed data to construct $N = 2^{15} \cdot 90 \cdot ln(2) \approx 2^{21}$ differences with $D = \sqrt{N \cdot 2} = \sqrt{2^{21+1}} \approx 2^{11}$.

The time complexity consists of querying the data, applying the sieving step and brute-forcing the remaining subtweak-key bits. Step 2 dominates the time complexity of the key sieving step. This step involves 4 out of 64 S-Boxes and therefore takes $N \cdot 2^{-7} \cdot 2^{32} \cdot 2^{-4} = N \cdot 2^{21}$ equivalent cipher encryptions. The total time complexity is $T = D + N \cdot 2^{21} + 2^{128-g} = 2^{11} + 2^{21} \cdot 2^{21} + 2^{128-90} \approx 2^{42}$ and the required memory is $N \approx 2^{21}$.

For the recovery of the key we in the next step need to attack the tweak-key schedule. Here, an attack with a low data complexity d is beneficial, since we need to repeat the attack on the datapath d times.

Extension to main-key recovery. By using the above attack, we can recover input/output pairs of the tweak-key schedule at a cost of $D_d \approx 2^{11}$ data and $T_d = 2^{21}$ time. To recover the main key, we will use an impossible differential attack on the tweak-key schedule as depicted in Figure 13. To perform the attack, we run the above attack to recover the round tweak-keys for $D_t = 188$ chosen tweaks according to the impossible differential distinguisher. Concretely, we keep the white tweak cells constant and choose 188 random values for the orange cells (■). This way, we receive 17 578 pairs, and since we are looking for a nonzero difference after MC in cell 8 (■) or alternatively in 11, i.e, we are

matching one of 30 differences. Now we guess the 16-bit key for the orange cells. For each guessed key, we can find at least 5 pairs that leads to the desired difference after MC with probability $\Pr(\mathcal{B}(n = 17\,578, p = \frac{30}{65535}) \geq 5) \approx 90.3\,\%$. Next, we guess 24 bits of the linearly equivalent keys after the last SC layer (■). Based on the 5 pairs, which must have a zero-difference in after the second-to-last SC layer, we can reduce the 2^{24} guesses to about 2^8. The other 3 cells that are guaranteed to be inactive, further reduce the key space by 16 bits each. Overall, we are left with a space of 2^{48} that we need to bruteforce. Hence, the total data complexity is $D = 188 \cdot 2^{11} \approx 2^{18.6}$ and total time complexity of $188 \cdot 2^{21} + 4 \cdot 2^{16+24} + 2^{48} \approx 2^{48}$. The overall time complexity could be lowered even more by using more data and more detailed analysis of the key-recovery step, like [DEKM16].

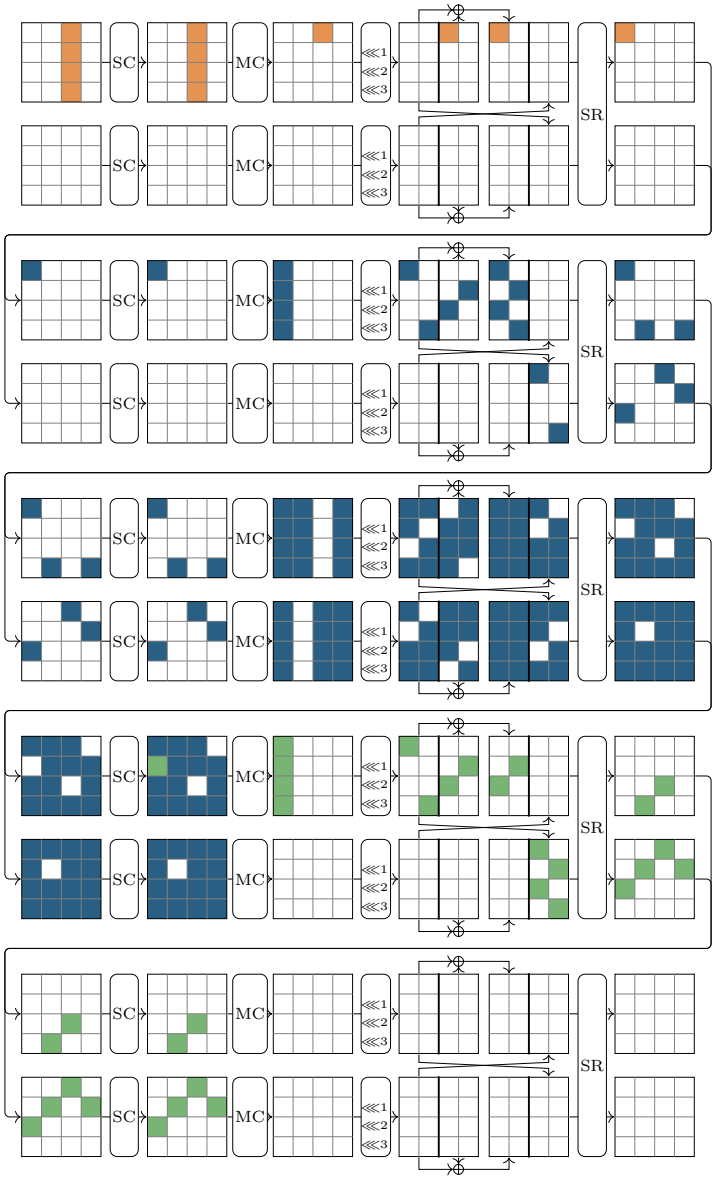


Figure 13: Attack on 5 rounds of the tweak-key schedule of BEANIE.

B Hardware Cost

In Table 14 we provide latency and area numbers for different MixColumns layers and S-Boxes. For simplicity, our S-Box (see 12a) is labeled as $G7$, the S-Box [035869c7dae41fb2] is labeled as $G9$ as its representative (see [Saa11, Table 4]), while R_4 and R_{13} are given in [WHG+21, Table 2]. Note that in Table 14 the number of rounds r denotes the application of full $r-1$ rounds of $R = \mathsf{MC} \circ \mathsf{SR} \circ \mathsf{SC}$, where the last r-th round applies only the S-Box layer SC.

Table 14: The performance comparison (on 16nm lib) between different configurations of the round function R with rotation layer $\ggg [0,1,0,1]$. The comparison is done on 5 rounds, i.e. four full rounds and S-Box layer ($Input \to R^4 \to SC \to Output$).

MixColumns	S-Box	Direction	Time (ps)	Area (μ^2)
$M_{4,i,4}$	$G7$	Enc.	987.0	949.6
	$G9$		1024.0	1052.7
	PRESENT		1009.0	984.6
	R_4		1033.0	1100.0
	R_{13}		1033.0	982.9
Hadamard	$G7$	Enc.	1008.0	947.1
	$G9$		1047.0	1014.2
	PRESENT		1032.0	976.2
	R_4		1050.0	1070.5
	R_{13}		1042.0	1084.5
$M_{4,i,4}$	$G7$	Dec.	1035.0	926.7
	$G9$		1117.0	1106.6
	PRESENT		1100.0	1054.5
	R_4		1108.0	1047.6
	R_{13}		1102.0	1122.7
Hadamard	$G7$	Dec.	1052.0	979.0
	$G9$		1134.0	1133.2
	PRESENT		1126.0	982.9
	R_4		1118.0	1063.2
	R_{13}		1121.0	1105.8
SKINNY	$G7$	Enc.	753.0	627.1
	$G9$		781.0	739.1
	PRESENT		773.0	644.1
	R_4		784.0	756.2
	R_{13}		787.0	737.0
MIDORI	$G7$	Enc.	814.0	786.2
	$G9$		850.0	865.0
	PRESENT		834.0	780.5
	R_4		858.0	871.6
	R_{13}		857.0	836.3

C Test Vectors

In Table 15 we provide test vectors for BEANIE. Here, we also list the round keys that are obtained from the tweak-key schedule. For debugging, we also provide each state of a specific encryption in Figure 15 (tweak-key schedule) and in Figure 14 (data path).

Table 15: Test vectors

	Test vector 1	Test vector 2
Key	0000000000000000	bd9c9afe2626f233
	0000000000000000	706ac764af470a53
Tweak	0000000000000000	2518c65012c8cdfb
	0000000000000000	84064a42a281c3aa
Round Keys	beedff0f	93061e07
	f8a29afc	87607a4d
	9369ab08	d7d11b34
	7391f5d3	b1769b2e
	464f65f3	1466644a
	e0f85edb	66a7801a
Plaintext	00000000	1841938a
Ciphertext	da46f4d3	092c2fea

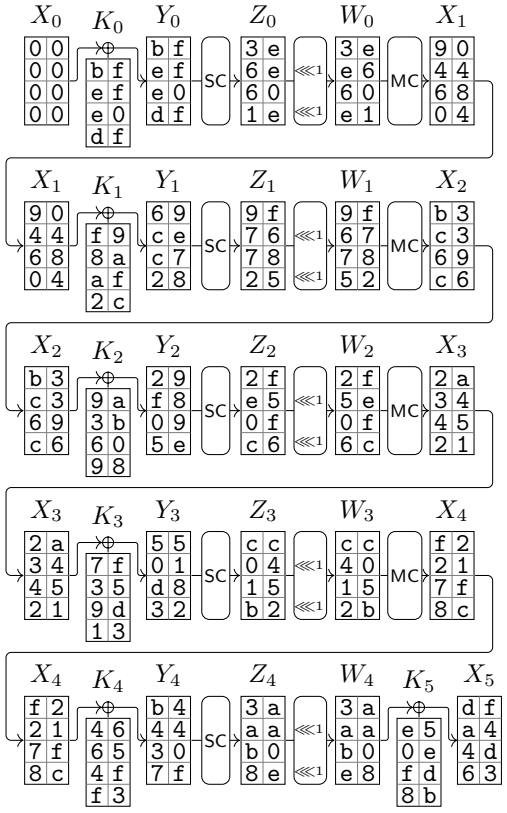


Figure 14: Test vector for the data path of BEANIE.

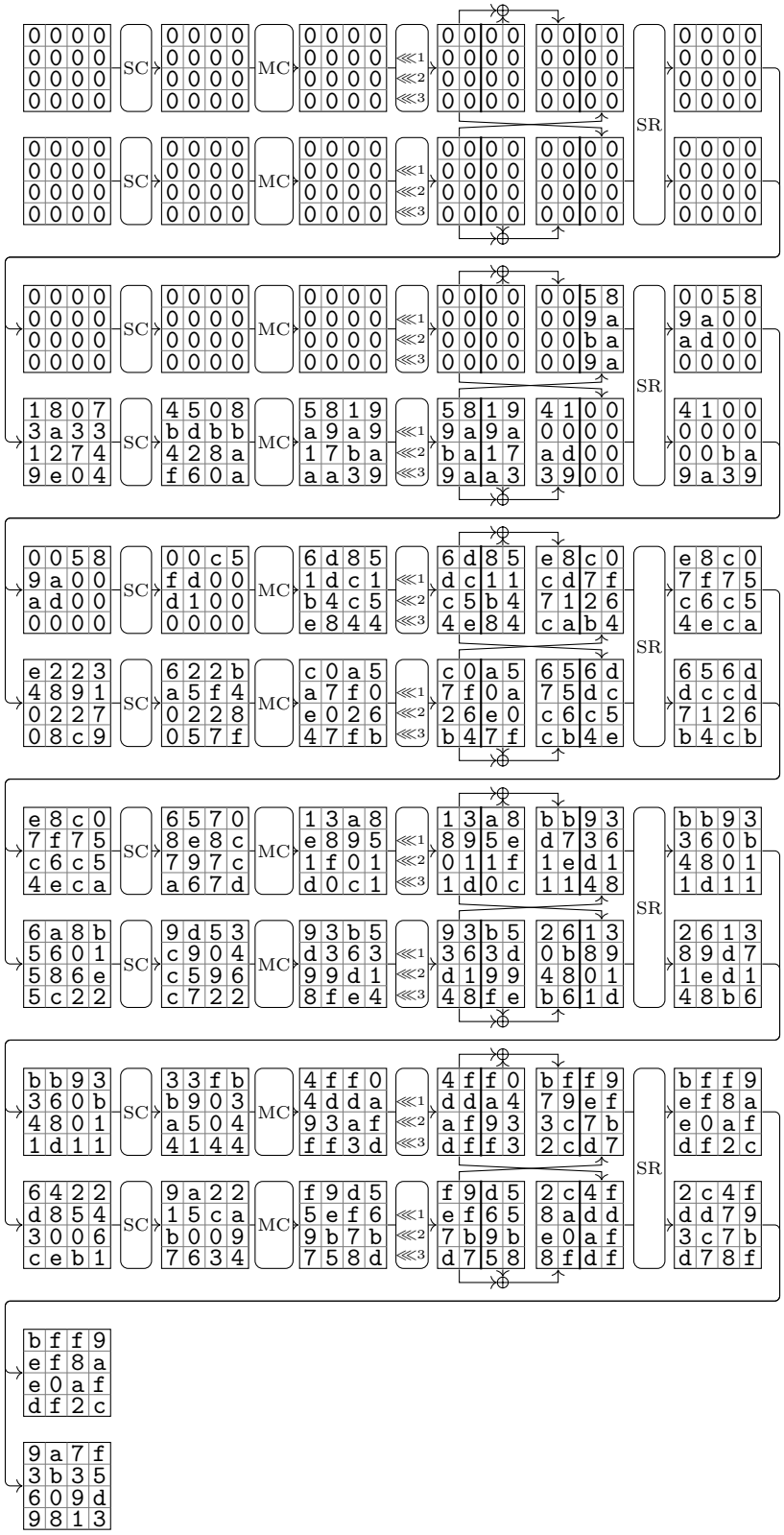


Figure 15: Test vector for the TKS. In the beginning and after each row the Key (0) and RC are added.