IACR Transactions on Cryptographic Hardware and Embedded Systems
ISSN 2569-2925, Vol. 2023, No. 1, pp. 326–368. DOI:10.46586/tches.v2023.i1.326-368

BipBip: A Low-Latency Tweakable Block Cipher with Small Dimensions

Yanis Belkheyar[1], Joan Daemen[1], Christoph Dobraunig[2], Santosh Ghosh[2] and Shahram Rasoolzadeh[1]

[1] Digital Security Group, Radboud University, Nijmegen, The Netherlands
firstname.lastname@ru.nl

[2] Intel Labs, Hillsboro, USA firstname.lastname@intel.com

Abstract. Recently, a memory safety concept called Cryptographic Capability Computing (C^3) has been proposed. C^3 is the first memory safety mechanism that works without requiring extra storage for metadata and hence, has the potential to significantly enhance the security of modern IT-systems at a rather low cost. To achieve this, C^3 heavily relies on ultra-low-latency cryptographic primitives. However, the most crucial primitive required by C^3 demands uncommon dimensions. To partially encrypt 64-bit pointers, a 24-bit tweakable block cipher with a 40-bit tweak is needed. The research on low-latency tweakable block ciphers with such small dimensions is not very mature. Therefore, designing such a cipher provides a great research challenge, which we take on with this paper. As a result, we present BipBip, a 24-bit tweakable block cipher with a 40-bit tweak that allows for ASIC implementations with a latency of 3 cycles at a 4.5 GHz clock frequency on a modern 10 nm CMOS technology.

Keywords: BipBip · low-latency · tweakable block cipher

1 Introduction

In this paper, we present BipBip, an ultra-low-latency 24-bit tweakable block cipher for the primary use in Cryptographic Capability Computing (C^3) [LRD+21]. C^3 aims to provide a generic low-overhead solution against long-lasting memory safety problems. In particular, it hardens systems against attackers that exploit software bugs and vulnerabilities like buffer overflows, use-after-free, etc. A capability architecture [Lev84] provides isolated environment for each object by defining precise access rights inherently with the object identifier. Traditionally, capability architectures for memory safety require the expansion of pointers enforcing radical changes to the microarchitectural structures throughout processors [WWC+14]. On the contrary, C^3 relies on the existing 64-bit pointer format utilizing the power of tailored symmetric cryptography while requiring minimal changes of the microarchitecture. For current 64-bit computer systems, C^3 encrypts one part of a 64-bit pointer and uses all other parts as tweak of the encryption. More specifically, as depicted in Figure 1, it encrypts 24 bits within the 64-bit pointer and the other 40 bits are used as a tweak.

Each pointer in C^3 is encrypted only once for each memory allocation and used repeatedly within the following code execution whenever it is dereferenced to access the data it points to. The pointer value needs to be decrypted within the processor core (more specifically, inside the memory execution or load/store pipeline) whenever it is dereferenced. Therefore, the pointer decryption procedure is latency-critical and directly related to the overall performance overhead of C^3.

Received: 2022-07-15 Accepted: 2022-09-15 Published: 2022-11-29

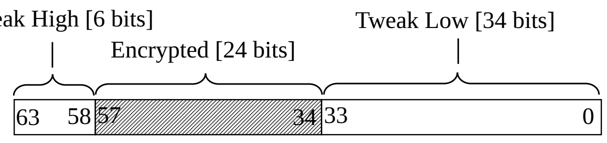

Figure 1: Pointer encryption format in Cryptographic Capability Computing.

Challenges and contributions. As far as we know, no one ever attempted to design an ultra-low-latency tweakable block cipher with a block size smaller than 32 bits while aiming to achieve a meaningful security level. Until very recently, there was simply a lack of applications for such ciphers. Hence, we faced several challenges imposed by the following restrictions stemming from the envisioned real-world application:

- *Ultra-low-latency.* The integration of the cipher for decrypting the pointers directly lies in the critical path of the pipeline of modern CPUs. Hence, one design goal for BipBip was to have a latency of under 3 cycles at 4 GHz on a modern CMOS technology to keep performance impacts small.

- *Small block size.* Since software should still be able to manipulate pointers within allocations, the allocations can vary in size, and for performance reasons (see [LRD$^+$21]), not the whole 64-bit pointer can be encrypted, but only a fraction of it. Such a small block size allows for attacks that are typically out of range for ciphers with a larger block size. For instance, if the full codebook for a single tweak is available, e.g., the difference distribution table (DDT) can be computed for a given key and tweak, which is infeasible for 64-bit or 128-bit block ciphers. Pairing the small block size with the stringent latency requirement increases the challenges to overcome even more, since only 24 bits of key material can be processed per round. This means that already 4 rounds are needed to process 96 bits of key material whereas a 64-bit and 128-bit block cipher can do this in 2 rounds and 1 round, respectively.

- *Low area.* Modern CPUs typically consist of many processors, each of them typically allowing for out-of-order (parallel) execution of instructions. Therefore, we have a multitude of places where pointers must be decrypted, leading to a multitude of hardware implementations of the tweakable block cipher within a single CPU. Thus, for a matter of cost, but also to save energy, it is important to allow for rather compact hardware implementations.

As we can see, the constraints imposed by the real-world use-case do not allow us trading to be exceptionally good in one metric, e.g., the latency, for being exceptionally bad in another one, e.g., the area. The small reliefs we get in the stringent constraints are that the cipher does not need to perform well in software, that low latency is only required for decryption and not encryption, and that efficient implementations of side-channel and fault injection attack countermeasures like masking [GP99, CJRR99] are not a primary concern. The reason that we do not focus on protection against physical attacks is because our primary application (C^3) aims to protect from remote software adversaries against exploiting software bugs, vulnerabilities, and memory safety issues by encrypting each pointer/linear address by the memory allocator with per process key, where decryption is only performed deep into the processor pipeline through a pipelined implementation of BipBip. However, for other applications and adversary models, the user should apply known SCA and FI countermeasures, where at least in one cipher evaluation direction, the rather low algebraic degree of the S-box of 2/3 should facilitate masking. Nevertheless, to meet the stringent latency requirements, we had to explore uncommon design paths.

- *Non-linear tweak schedule.* The majority of recently designed popular tweakable block ciphers like Skinny [BJK+16], Qarma [Ava17] and other ciphers following the tweakey framework [JNP14] use a linear tweak schedule and a linear key schedule. As with most design choices, the choice of a linear tweak schedule comes with benefits, e.g., easier to analyze related-tweak differential trails, but also bears some drawbacks. In this case, the most important drawback is the limited ability of a linear tweak schedule to contribute to the cryptographic strength of the tweakable block cipher. In essence, having a linear tweak schedule means that only the datapath provides the necessary "non-linearity" required to make the cipher secure. Hence, the only way to increase the cryptographic strength for a fixed block size is to either add more rounds or use stronger round functions. Both options typically have a negative impact on the latency.

  In contrast, a non-linear tweak schedule can significantly impact the cryptographic strength against attacks aiming to exploit tweak relations in the case of differential [BS90] and linear [Mat93] cryptanalysis. While the time budget of 3 cycles at 4 GHz limits the usable number of sequential operations a.k.a. rounds, we can adjust the width of the tweak schedule rather freely. Hence, we can increase the number of operations by computing them in parallel. This means, it is to a certain extent possible to increase the cryptographic strength of a cipher by increasing the width of the non-linear tweak schedule. If area was of no concern, we could use a sponge function [BDPV11b] with a capacity bigger than our aimed security level of 96 bits as tweak schedule. However, we have to find a balance between area and strength of the tweak schedule, as we did in choosing a tweak schedule of 53-bit width to process a 40-bit tweak.

- *Heterogenous rounds.* One very strong challenge we faced in the design of a 24-bit tweakable block cipher was the limited amount of key material that can be processed per round of the datapath. In essence, this means that we have to deal with attacks that allow to partially en-/decrypt ciphertexts and plaintexts under key guesses for a considerable number of rounds. In addition, also meet-in-the-middle attacks become a considerable threat. To increase the number of rounds while keeping the latency constant, we decided to use a layered structure similar to the block cipher Mars [BCD+99] with more lightweight rounds surrounding a core consisting of cryptographically stronger rounds. Essentially, this follows the insight that mainly the core of a cipher affects the strength of distinguishers, while the outer rounds are likely to be bridged by key guessing in attacks.

- *Larger masterkey.* Guessing bits of the round keys used in the datapath or tweak schedule in attacks should be as costly as possible to allow for a cipher with a low number of rounds. Essentially, what we want to prevent is that a very small advantage on a guess of a small fraction of the round keys in the datapath or tweak schedule can be easily turned into an advantage in recovering the masterkey. Or conversely, that the guess of a single masterkey bit reveals many more round key bits in an exploitable manner. We denote the ability to extract x bits of round key material out of y bits of the masterkey ($x \geq y$) with key bridging.

  Therefore, we decided to use more masterkey bits than minimally needed to meet the claimed security level. In particular, while aiming at 96 bits of security, we use a 256-bit masterkey. This decision has also an interesting effect on the area of ASIC implementations. The expected dominant use of BipBip for C^3 most likely relies on pipelined hardware implementations. Having a key of 96 bits would likely require a strong key schedule to evolve the round key in each pipeline stage. Hence, additional registers are needed to store the round key, which sum up to $3 \cdot 96 = 288$ registers in a three-stage pipeline. This requires already more area than storing a 256-bit static masterkey, so the larger masterkey even saves us some area.

- *Cryptanalyst's playground.* The small block size, together with our design decisions, make BipBip quite unique in the realm of cryptographic primitives with real-world usage. The small block size allows to apply cryptographic techniques or evaluate cryptographic properties practically on the real cipher, minimizing the need to create smaller toy versions. In addition, the small block size allows for the application and evaluation of little followed concepts, e.g., attacks using multiple differentials [AL12] up to the reconstruction of the full DDT for a given key and tweak.

Influence of specialized ciphers. While the lion's share of symmetric cryptographic tasks like (authenticated) encryption, tag computation, and hashing is fulfilled by general-purpose primitives like AES [DR20], or ChaCha [Ber08], primitives dedicated to specific applications, such as BipBip are increasingly important. Even if such a specialized symmetric primitive is just used for a single use-case, the ubiquity and large-scaling of information technology can lead to an increase in security for billions of devices and users, often impossible to achieve with general-purpose primitives. A prime example for this is the tweakable block cipher Qarma [Ava17] and its use for pointer authentication in processors.

The strict requirements that specialized primitives have to fulfill often leads to unconventional design choices, like using a nonce-dependent affine layer [DEG$^+$18], or security via secret algorithm variability [KLPR10]. At the time of the publishing of these designs, the security implications of such decisions might not be completely explored or understood. However, even if such choices lead to a security flaw, we can benefit from the discovery of new cryptanalytic techniques and concepts and learn more about the impact of the taken design choices. In a bigger picture, this allows us to increase our knowledge on symmetric cryptography paving the way to more efficient and secure symmetric cryptographic algorithms. Examples of ciphers leading to novel cryptanalytic insights include a cipher for IC printing called PRINTcipher [KLPR10] and the invariant subspace attack [LAAZ11], or a cipher designed for multi-party computation called LowMC [ARS$^+$15] and advancements in algorithms for solving multivariate equation systems [Din21].

Related work. A very active research area that deals with (tweakable) block ciphers having a small block size is format preserving encryption (FPE). Although the focus of FPE is to provide flexible solutions for varying block sizes and not on low latency, we can still learn from design decisions and cryptanalysis in that area. Prominent tweakable block ciphers for FPE are the NIST standards FF1 and FF3/FF3-1, and the South-Korean standards FEA-1 and FEA-2. Those standards have in common that they are based on the Feistel structure using very strong round functions like truncated AES. Furthermore, FF3/FF3-1, FEA-1, and FEA-2 have a very simplistic linear tweak schedule. Despite their very strong round functions, numerous attacks on those FPE schemes have been published in recent years [BHT16, DV17, HTT18, DKLS20, ADK$^+$21, Bey21]. Many of these attacks [DV17, DKLS20, Bey21] exploit the simplicity of the tweak schedule and hence, strengthened our confidence in the choice of a non-linear tweak schedule.

Other prominent block ciphers with small block length are the 32-bit block versions of Katan and Ktantan [DDK09]. They aim for hardware implementations with very low area and have many rounds and hence, no low latency. Still, meet-in-the-middle attacks on Ktantan [BR10] demonstrate the challenges faced in designing a good key schedule for block ciphers with small block sizes.

Outline. The paper starts with the specification of BipBip in Section 2. After that, we give insight into our design decisions in Section 3. In Section 4, we show our preliminary

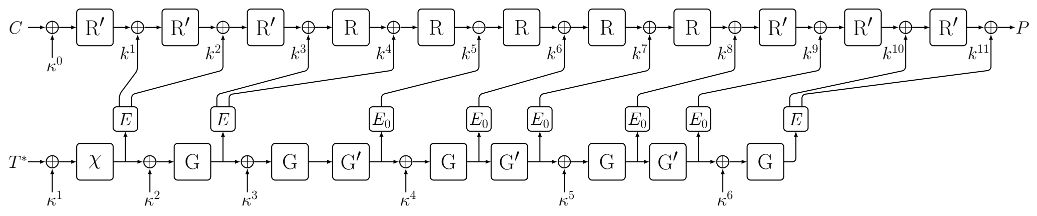

Figure 2: Structure of BipBip.

cryptanalysis to support our security claim. The implementations discussed in Section 5 demonstrate that BipBip indeed has a low latency. Finally, we conclude in Section 6.

2 Specification of BipBip

BipBip is a tweakable block cipher designed to have low-latency decryption when implemented on ASICs. BipBip has a block size of 24 bits, masterkey length of 256 bits, and tweak length of 40 bits. Compared to most block cipher specifications that describe the cipher as transformation from the plaintext to the ciphertext, we do the main description from ciphertext to plaintext, since this is the transformation with the more stringent latency requirements.

2.1 High-level Structure

Figure 2 depicts the high-level structure of the data flow in BipBip's decryption. BipBip consists of three main parts, the datapath, the tweak schedule, and the key schedule. The key schedule forms the tweak-round keys κ^i and the whitening key κ^0 by selecting bits from the 256-bit masterkey K. For clarity, the key schedule is omitted in Figure 2.

The datapath starts with the addition of the whitening key κ^0 to the ciphertext C, followed by the alternating application datapath rounds R or R′ and data-round key additions with k^i to finally output the plaintext P. The data-round keys k^i are extracted from the tweak schedule.

The tweak schedule processes the tweak T and tweak-round keys κ^i to derive the data-round keys k^i. This is done by the alternating application of tweak-round keys κ^i and tweak schedule rounds G or G′ on a 53-bit state. The data-round keys k^i are extracted from the 53-bit state between the round applications. The 53-bit state is initialized with the padded 40-bit tweak.

To derive round-reduced versions of BipBip for cryptanalysis, we use the naming convention BipBip$_{x,y,z}$. This means for the decryption the first part is constituted of x shell rounds R′, the middle part of y core rounds R and the final part of z shell rounds R′. So, the full version of BipBip can be denoted with BipBip$_{3,5,3}$. The tweak schedule and key schedule is the same for round-reduced versions and the data-round keys are used in the order κ^0, k^1, k^2, …, for as many data-round keys as needed to instantiate the round-reduced version of BipBip.

2.2 Datapath

The datapath uses two round functions, one called *core* round function R and the other *shell* round R′. The shell round function has no mixing layer and therefore has lower latency than the core round function. The core round function R consists of an S-box layer S, a linear mixing layer θ_d, and two-bit shuffles π_1 and π_2. The shell round function

Table 1: The truth table for BipBipBox of the S-box layer.

x_5x_4	.0	.1	.2	.3	.4	.5	.6	.7	.8	.9	.a	.b	.c	.d	.e	.f
0.	00	01	02	03	04	06	3e	3c	08	11	0e	17	2b	33	35	2d
1.	19	1c	09	0c	15	13	3d	3b	31	2c	25	38	3a	26	36	2a
2.	34	1d	37	1e	30	1a	0b	21	2e	1f	29	18	0f	3f	10	20
3.	28	05	39	14	24	0a	0d	23	12	27	07	32	1b	2f	16	22

(Column header: $x_3x_2x_1x_0$)

R′ includes the same S-box layer S and a different bit shuffle π_3. We have:

$$\mathrm{R} = \pi_2 \circ \theta_d \circ \pi_1 \circ S\,, \quad \text{and} \quad \mathrm{R}' = \pi_3 \circ S\,.$$

2.2.1 S-box layer S

The S-box layer of BipBip's datapath is based on a 6-bit S-box, called BipBipBox. The state is divided into 4 *words* of 6 bits and applies BipBipBox to each word in parallel:

$$S: \quad (y_{6i+5}, y_{6i+4}, y_{6i+3}, y_{6i+2}, y_{6i+1}, y_{6i}) \\ \leftarrow \text{BipBipBox}(x_{6i+5}, x_{6i+4}, x_{6i+3}, x_{6i+2}, x_{6i+1}, x_{6i})\,,$$

for $0 \leq i < 4$. We specify BipBipBox with a truth table using hexadecimal notation in Table 1 and the following algebraic representation.

$$\begin{aligned}
y_5 &= x_4x_3x_2 + x_3x_2x_1 + x_5x_0 + x_4x_3 + x_3x_2 + x_2x_1 + x_5 \\
&= x_4x_3\overline{x_2} + \overline{x_3}x_2x_1 + x_5\overline{x_0} + x_3x_2, \\
y_4 &= x_1x_2 + x_0x_3 + x_4 + x_1x_4 + x_5 + x_3x_5 = x_1x_2 + x_0x_3 + \overline{x_1}x_4 + \overline{x_3}x_5, \\
y_3 &= x_1x_2 + x_3 + x_0x_3 + x_4 + x_2x_4 + x_0x_5 = x_1x_2 + \overline{x_0}x_3 + \overline{x_2}x_4 + x_0x_5, \\
y_2 &= x_2 + x_1x_3 + x_2x_3 + x_0x_4 + x_5 + x_4x_5 = x_1x_3 + x_2\overline{x_3} + x_0x_4 + \overline{x_4}x_5, \\
y_1 &= x_5x_3x_2 + x_3x_2x_0 + x_5x_3 + x_4x_1 + x_3x_2 + x_2x_0 + x_1 \\
&= x_5x_3\overline{x_2} + \overline{x_3}x_2x_0 + \overline{x_4}x_1 + x_3x_2, \\
y_0 &= x_0 + x_0x_2 + x_2x_3 + x_4 + x_1x_5 + x_4x_5 = x_0\overline{x_2} + x_2x_3 + x_1x_5 + x_4\overline{x_5}.
\end{aligned}$$

2.2.2 Mixing layer θ_d

The mixing layer multiplies the datapath state with a binary circulant matrix:

$$\theta_d: \quad y_i \leftarrow x_i + x_{i+2 \bmod 24} + x_{i+12 \bmod 24}\,, \qquad \text{for all } 0 \leq i < 23\,.$$

2.2.3 Bit permutations π_1, π_2 and π_3

We permute bits in the datapath state:

$$\pi_1: \quad y_i \leftarrow x_{P_1(i)}\,, \qquad \pi_2: \quad y_i \leftarrow x_{P_2(i)}\,, \qquad \pi_3: \quad y_i \leftarrow x_{P_3(i)}\,,$$

for all $0 \leq i < 23$ where P_1, P_2 and P_3 are permutations of $\mathbb{Z}/24\mathbb{Z}$ specified by following tables:

$$\begin{aligned}
P_1 &= [1, 7, 6, 0, 2, 8, 12, 18, 19, 13, 14, 20, 21, 15, 16, 22, 23, 17, 9, 3, 4, 10, 11, 5]\,, \\
P_2 &= [0, 1, 4, 5, 8, 9, 2, 3, 6, 7, 10, 11, 16, 12, 13, 17, 20, 21, 15, 14, 18, 19, 22, 23]\,, \\
P_3 &= [16, 22, 11, 5, 2, 8, 0, 6, 19, 13, 12, 18, 14, 15, 1, 7, 21, 20, 4, 3, 17, 23, 10, 9]\,.
\end{aligned}$$

2.3 Tweak schedule

The tweak schedule operates on a 53-bit state, called the *tweak state*. It consists of the application of two types of round functions G and G′, and the addition of the tweak-round keys κ^i. The tweak-round keys come from the key schedule that selects them from the masterkey.

2.3.1 Round functions

The first tweak schedule round function G has 4 steps:

$$\mathrm{G} = \chi \circ \pi_5 \circ \theta \circ \pi_4 \ ,$$

with χ being a non-linear layer, θ a linear mixing layer and π_4 and π_5 bit permutations. We have, $\forall i$:

$$\begin{aligned}
\chi : &\quad a_i \leftarrow a_i + (a_{i+1} + 1)\, a_{i+2} \ , \\
\pi_4 : &\quad a_i \leftarrow a_{13i} \ , \\
\theta_t : &\quad a_i \leftarrow a_i + a_{i+1} + a_{i+8} \\
\pi_5 : &\quad a_i \leftarrow a_{11i} \ ,
\end{aligned}$$

Here, all operations are done in $\mathbb{F}_2$ for the state bits and modulo 53 for indices.

The second tweak schedule round function G′ differs from G by the mixing layer:

$$\mathrm{G}' = \chi \circ \pi_5 \circ \theta' \circ \pi_4 \ ,$$

with

$$\theta' : \quad a_i \leftarrow a_i + a_{i+1} \text{ for all } 0 \leq i < 52, \text{ and } a_{52} \leftarrow a_{52},$$

2.3.2 Tweak state initialization

We extend the 40-bit tweak T to 53 bits by padding it with a single 1 bit and twelve 0 bits. So, we initialize the state of the tweak schedule with $T^* = T \| 1 \| 0^{12}$.

2.3.3 Extracting data-round keys (E)

We derive 24-bit data-round keys from the 53-bit intermediate state value x^i using two extractor functions E_0 and E_1. We have

$$k^i = E_0(x^i) = (x^i_0, x^i_2, .., x^i_{46}) \ , \qquad k^{i+1} = E_1(x^i) = (x^i_1, x^i_3, .., x^i_{47}) \, .$$

2.4 Key schedule

We compute the following keys from the 256-bit masterkey K of bits K_j with $j \in 0, \ldots, 255$:

- one 24-bit data-round key κ^0,
- six 53-bit tweak-round keys κ^i with $1 \leq i \leq 6$.

We build the 24-bit k^0 as:

$$\kappa^0 = (K_3, K_{3^2 \bmod 256}, \ldots, K_{3^{24} \bmod 256}) \, .$$

and the six 53-bit tweak-round keys as:

$$\kappa^i = (K_{53i \bmod 256}, K_{53i+1 \bmod 256}, \ldots, K_{53i+52 \bmod 256}) \, .$$

2.5 Security Claim

We make our security claim in reference to the probability of correctly guessing that a ciphertext C_i maps to a plaintext P_i under a certain tweak T_i ($C_i \leftrightarrows P_i|T_i$) for an instance of BipBip denoted by E_K where the masterkey K has been randomly chosen and that (P_i, C_i) pair has not been queried before. The probability of a correct guess depends on the queries made before the guess and the computational resources spent by an adversary, which we denote with:

q: total number of queries to both encryption and decryption of E_K,

q_{T_i}: number of queries to both encryption and decryption of E_K with tweak of value T_i,

t: computation time with the unit amount of computation equivalent to evaluating E_K.

Since a look-up in the L1 cache has a latency (3–4 cycles) similar to that of a pipelined implementation of BipBip, we attribute one memory lookup to be equivalent to one E_K.

Our security claim is the following bound on the probability of a correct guess

$$P_{BipBip}(C_i \leftrightarrows P_i|T_i) \leq \frac{1}{\max(2^{24-\mu} - q_{T_i}, 1)} + \frac{q}{2^{96}} + \frac{t}{2^{96}} + \frac{qt}{2^{120}},$$

where $\mu = 0.5$ and the reason behind this choice is explained in detail at Section 3.5 and Section 4.

3 Design Rationale

As we already detailed in the introduction, the goal of our design process was to fit into the stringent requirements imposed by the application within C^3. In this section, we discuss the decisions we made when designing the datapath, tweak schedule, and key schedule in more detail.

3.1 Preliminaries on differential and linear propagation

Here we give a short introduction to differential and linear propagation as they play an important role in our rationale.

Differential cryptanalysis exploits *differentials* (a, b) with high probability $\mathrm{P}(a, b)$ over (part of) a cipher. Here a is a difference at the input and b a difference at the output. We denote with *differential probability* (DP) the number of input pairs with difference a that map to output difference b divided by the total number of input pairs with input difference a for a certain fixed key. DP of differentials applies to building blocks such as S-boxes but also to larger structures such as a data path of a block cipher or a permutation. For a keyed function B_k, the DP of a differential is in general key-dependent and we can write $\mathrm{DP}_{B_k}(a, b)$. The average of the DP taken over all sequences of round keys is called the *expected DP* (EDP) and we denote it as $\mathrm{EDP}_B(a, b)$ or for short $\mathrm{EDP}(a, b)$ if the function B is known from the context. As we will see in Section 4.3.3, the EDP value of a differential says something about the distribution of the number of pairs in that differential over all keys.

In an iterated cipher, a differential can be decomposed in *differential trails.* A trail specifies not only the input difference a and output difference b but also the intermediate differences: it can be seen as a chain of round differentials. The DP of a trail is the probability that an input pair with difference a exhibits the differences specified by the trail. The EDP of a trail is the expected value over all round key sequences and can be computed by multiplying the DP values of the round differentials. Clearly, the (E)DP of a differential (a, b) is the sum of the (E)DP values of all differential trails with initial difference a and final difference b.

Linear cryptanalysis exploits *linear approximations* (a, b) with high correlation $\mathrm{C}(a, b)$ over (part of) a cipher. Here a is a mask at the input and b a mask at the output and the correlation is between the linear Boolean functions defined by these masks and has a value in the range $[-1, 1]$. The linear potential (LP) of a linear approximation is the square of its correlation: $\mathrm{LP}(a, b) = \mathrm{C}^2(a, b)$. LP of linear approximations applies to building blocks such as S-boxes but also to larger structures such as a data path of a block cipher or a permutation. For a keyed function B_k, the correlation of a linear approximation is in general key-dependent and we can write $\mathrm{C}_{B_k}(a, b)$. The linear potential (LP) of a linear approximation is the square of its correlation: $\mathrm{LP}_{B_k}(a, b) = \mathrm{C}^2_{B_k}(a, b)$. The average of the linear potential taken over all sequences of round keys is called the expected linear potential (ELP) and we denote it as $\mathrm{ELP}_B(a, b)$ or for short $\mathrm{ELP}(a, b)$ if the function B is known from the context. As we will see in Section 4.4.2, the ELP value of a linear approximation says something about the distribution of the correlation of that linear approximation over all keys.

In an iterated cipher, a linear approximation can be decomposed in *linear trails*. A trail specifies not only the input mask a and output mask b but also the intermediate masks: it can be seen as a chain of linear approximations over the rounds. The correlation contribution of a trail is the product of the correlations of its round linear approximations and its LP is simply the square of its correlation contribution. The correlation of a linear approximation (a, b) is the sum of the correlation contributions of all linear trails with initial mask a and final mask b. The ELP of a linear approximation (a, b) is the sum of the LP values of all linear trails with initial mask a and final mask b.

3.2 Datapath

3.2.1 High-level structure

Probably the first decision to be made when designing a datapath is to follow either a (generalized) Feistel approach, or to construct the datapath as iterated application of full-width non-linear permutations, linear permutations, and key additions, which we shorten as NLK approach that is also called Substitution-Permutation-Network (SPN) in the literature. While one round of the Feistel approach influences only a part of the state, one round of NLK influences the whole state and hence, is considered by us to be favorable in low-latency applications.

The next high-level decision made was to use two different round functions. We use strong rounds R which build the core of the datapath and more lightweight rounds R' on the outside (shell) of the datapath. The only difference between the core rounds and the shell rounds lies in their linear layer, which is a bit-permutation in the case of the shell rounds. The main motivation behind this choice lies in the rather small data-round keys we have versus the security level we aim to achieve. As we can see, e.g., in meet-in-the-middle attacks of Section 4.8, it is feasible for an attacker to guess whole data-round keys to calculate internal state values. If a whole data-round key is guessed, an attacker can calculate the output of one round from the input of one round independently of the strength of the round function.

Hence, we decided to use shell rounds R' at places in the cipher that are likely to be skipped by key guesses in attacks in order to increase the total number of rounds of the cipher compared to a cipher with the same latency that just uses the core rounds R.

3.2.2 S-box

Our non-linear layer is the parallel application of four 6-bit S-boxes. The reason we did not choose a 3-bit, or 4-bit S-box is based on the upper bounds given in [BCD11] for estimating growth of the algebraic degree. 3-bit or 4-bit S-boxes provide a slow growth in comparison with a 6-bit S-box of the same algebraic degree.

The combination of a bit permutation with the non-linear layer of 6-bit S-boxes allows the input of each S-box to depend on the previous round outputs of all S-boxes.

Since there exist 6-bit S-boxes with a rather low latency while having good cryptographic properties [LMMR21], we decided to use 6-bit S-boxes in our design. We think that 6-bit S-boxes provide a good trade-off between latency and security especially when paired with a good linear mixing layer, as used for the core rounds R.

Our S-box BipBipBox is built in a way that its hardware implementation for achieving the lowest latency has gate depth of 4 in the basis of 2-bit NAND and NOR gates. This S-box provides the smallest nontrivial maximum DP and LP, namely 2^{-4} for both values, within all the bijective 6-bit S-boxes with gate depth of 4 in the same basis. Note that the smallest nontrivial maximum DP and LP among all bijective 6-bit S-boxes are 2^{-5} and 2^{-4} respectively that is known as Dillon's S-box [BDMW10]. However, it is not possible to implement any of such S-boxes within gate depth of 4 in the basis of 2-bit NAND and NOR gates [Ras22].

Besides, each coordinate of BipBipBox non-linearly involves all the input bits, so-called full dependency and it is the same for its inverse S-box. This property helps us to provide full diffusion in the datapath after two S-box layers. Furthermore, the algebraic degree of BipBipBox is 3 (for only two coordinates) and 2 (for other coordinates), while for its inverse it is 4 (for four coordinates) and 3 (for other coordinates). For more information about the implementation of BipBipBox, see Section 5.

The BipBipBox was previously found in [Ras22] which introduces a new technique for building S-boxes. The basic idea behind technique is to build low-latency bijective n-bit S-boxes with an upper limit on the linear potential (LP) and differential probability (DP). We first need to find $\mathcal{F}$, the set of all low-latency and balanced n-bit Boolean functions with LP of less or equal than the limit. Then, we start to build the S-box. We choose $f_0, f_1 \in \mathcal{F}$ as two coordinates of the S-box. We call f_1 is compatible with f_0, if $f_1 + f_0$ is balanced and has an LP of less or equal than the limit. If f_1 is not compatible with f_0, we choose another function for f_1. But if it is, then we continue to select the third coordinate $f_2 \in \mathcal{F}$. We call f_2 to be compatible with f_0 and f_1, if $f_2 + f_0$, $f_2 + f_1$, $f_2 + f_1 + f_0$ are balanced and have LP of less or equal to the limit. Again, if f_2 is not compatible with f_0 and f_1, we choose another function for f_2. But if it is, then we select the fourth coordinate $f_3 \in \mathcal{F}$. We continue in this way to have n coordinates that are jointly compatible with each other. That is all $2^n - 1$ component functions (any linear combinations of f_i functions) are balanced and have LP of less or equal than the limit. Then, we can check for DP of the S-box to be less or equal to the given limit. If the S-box fulfills this condition, then it is a low-latency S-box with given conditions in LP and DP. For more details on this algorithm of building S-box and the possibilities for making it efficient, see [Ras22].

We have chosen the BipBipBox over, e.g., the Speedy [LMMR21] S-box, since its cryptographic properties provide a good balance in the number of rounds needed to defend against differential, linear, and higher-order differential attacks. Such a balance in attack vectors is important for low latency ciphers, since the security is determined by the strongest attack vector and so is the number of rounds that have a huge influence on the latency. For example, the Speedy [LMMR21] S-box and the BipBipBox have very similar implementation characteristics. However, the Speedy [LMMR21] S-box has a higher algebraic degree, but worse maximum DP and LP. Hence, we would need more rounds to defend against differential and linear cryptanalysis, leading to more rounds and a higher latency. The same argument is true if we would take an S-box with an algebraic degree two in all coordinates. We would need many more rounds to defend against higher-order differential attacks leading to a cipher having a higher latency as indicated by the degree evolution in Table 7.

3.2.3 Linear layers

For the shell rounds R', we have decided to pair the non-linear layer with a linear layer that is just a bit permutation, π_3. The first criterion for choosing the bit permutation π_3 is that the input of each S-box of a round depends on at least one output bit of each of the 4 S-boxes from the previous round. To be precise, we use two inputs from outputs of one S-box, two from another S-box, and one from each of other remaining S-boxes. This way, all the output bits after the second round are non-linearly dependent on all the input bits of the first round. Note that the same holds for the inverse of two shell rounds. The input bit of the first round non-linearly depends on all the output bits after the second round.

The second criterion is to optimize the degree in the algebraic normal form (ANF) representation for r consecutive shell rounds. We maximize the minimum algebraic degree of the coordinates after two rounds. For this, we try all the bit permutations satisfying the first criterion and compute the algebraic degree after two rounds. The minimum algebraic degree of the coordinates after two rounds for the best bit permutations with respect to this criterion is 5. In more detail, in the output of each S-box, there are two coordinates with degree 5, two with degree 6 and the other two are with degree 8. Note that there are some bit permutations which make the algebraic degree of some coordinates to be 9, but on the other hand, the minimum algebraic degree for the other coordinates will be smaller than 5.

The minimum degree of the coordinates after 3, 4 and 5 shell rounds, for the bit permutation we used as π_3, is 13, 20 and 23, respectively. Precisely, after 3 rounds there are two coordinates with algebraic degree 13, six with 14, eight with 16, two with 18, five with 19 and one with 20. After 4 rounds, the algebraic degrees are either 20, 21 or 22, that each happens exactly for eight coordinates, and after 5 rounds all the coordinates have the maximum algebraic degree, 23.

For the core rounds R, we use two bit permutations and a linear mixing layer θ which calculates the sum of 3 rotated states. Hence, the key addition and linear mixing layer can be computed with a gate depth 2. Note that compared to the shell rounds, the core round has only one extra gate depth and using mixing layers with an output that sums more than 3 input bits will require higher gate depth value, which will cause a significant cost (more than 50%) on the latency of the linear mixing layer.

There are about $\binom{24}{3}^{24} \approx 2^{11 \cdot 24}$ of such linear mixing layers. So it is impossible to exhaustively analyze every single mapping in this space. That is why we restrict ourselves to use a linear mixing layer based on a circulant matrix together with bit shuffles in its input and output.

Up to bit shuffles in the input and in the output bits, in dimension 24, there are only 15 circulant invertible matrices which sum 3 input bits in each output bit. For each of these 15 matrices, we check all the input- and output-bit shuffle pairs to find one suiting the following criteria.

For the first criterion we decide that for each output bit of the linear layer, the three corresponding input bits come from outputs of different S-boxes. It is similar for the inverse that each input bit is computed from inputs of at least 3 different S-boxes in the next round. This criterion's intent is to improve the diffusion within the cipher.

The second criterion is that we want the word-wise (words of 6 bits) branch number for the linear layer and its inverse to be the maximum, 4. Therefore, in any differential or linear trails for two rounds (such that the first one is a core round), there are at least four active S-boxes. This condition alone guarantees that in any differential or linear trail for $2r$ and $2r+1$ core rounds, there are at least $4r$ and $4r+1$ active S-boxes. Note that the first condition is already included in the second one, but for a given circulant matrix it is easier to first filter the bit shuffle pairs satisfying the first condition and then from the remaining bit shuffle pairs, we check for the second condition.

From those 15 matrices, for only 6 of them there are some bit shuffle pairs fulfilling these two criteria. Moreover, from those 6 matrices, only for one matrix there are bit shuffle pairs providing semi-uniform distribution of input words of the linear layer in its output words. That is, since in each output word $3 \cdot 6$ bits are involved, for a uniform distribution of the input words in the output words, we need that those 18 bits are 5 from one word, 5 from another word and 4 from each of other words. The θ_d chosen for BipBip's mixing layer is the corresponding mapping for this matrix.

As the last criterion for the linear layer of core rounds, we considered the algebraic degree of the coordinates after each round. Similar to the one for shell rounds, within the remaining bit shuffle pairs for the chosen θ_d, we prefer the ones which cause the highest minimum algebraic degree for coordinates of the output after r core rounds. Using P_1 and P_2 bit shuffles, after two rounds algebraic degree of the coordinates are either 6 or 9, after three rounds are 16 or 20, after four rounds are 22 or 23 and after 5 rounds all coordinates are full degree. The complementing preliminary algebraic cryptanalysis that aims to exploit low algebraic degrees can be found in Section 4.7.

3.3 Tweak schedule

3.3.1 High-level structure

Low-latency tweakable block ciphers like Mantis [BJK+16] or Qarma [Ava17] have a low latency for encryption and decryption. One ingredient that makes this possible is their simple and structured tweak schedule. However, in the case of BipBip, we only require ultra-low latency for decryption. Therefore, from a performance perspective, there is no issue with using a non-linear tweak schedule with high diffusion. In contrast, using a non-linear tweak schedule is an opportunity to lower the latency compared to using a simple and structured tweak schedule, since non-linearity paired with good diffusion in the tweak schedule can significantly contribute to the cryptographic strength of the cipher.

We decided to use a tweak schedule of 53-bit width, adding 53-bit tweak-round keys to its internal state whenever we have extracted 48 bits of data-round key material. With 53, we have chosen the nearest prime number that allows to initialize the state with a 40-bit tweak and extract two 24-bit data-round keys from the state at once. In addition, a 53-bit state allows to make a considerable impact on the cryptographic strength of the tweakable block cipher by ensuring, e.g., the non-existence of high-DP differential trails that use differences in the tweak as shown in Section 4.5.1.

3.3.2 Round functions

The operation of the cipher does not require the computation of the inverse of the tweak schedule and therefore its round functions do not need to have efficient inverses. This is a feature in common with the sponge and duplex construction [BDPV11a] and hence, it makes sense to get inspiration from permutations from prominent permutation-based schemes like Keccak [BDPV11c], Ascon [DEMS21], Norx [AJN14], Subterranean v2.0 [DMMR20], or Xoodyak [DHP+20]. From these options, Subterranean v2.0 has the round with the lowest number of sequential operations, while providing good cryptographic properties. In particular, the single-circle χ has an inverse with a high degree that complicates cryptanalysis.

Therefore, the core round function G we use in the tweak schedule is a scaled-down version of that of Subterranean. Moreover, we use a round function G′, which has a linear layer with a lower latency. We use this variant to maximize the number of rounds in the tweak schedule given the latency constraints.

3.4 The envisioned use of BipBip

The primary use of BipBip is within C^3. Here it is used for the encryption of pointers. More specifically, 24 bits of 64-bit pointers are encrypted, while other parts of the pointer are used as tweak. If an attacker now manipulates such an encrypted pointer, on use of this pointer, the pointer gets decrypted. Dependent on the memory load of the system, the decrypted pointer now points to an unmapped page with a rather high probability and triggers a page fault. Encrypting pointers can be seen as a form of implicit pointer authentication. However, compared to pointer authentication using a MAC truncated to a few bits (e.g., as low as 3 to 32 bits) [Ava17], C^3 has the benefit that address space has not to be shrunk to store the tag, while still providing reasonable security in this online attack use-case.

As we can see from above, the envisioned use-case for BipBip is as a stand-alone tweakable blockcipher ensuring authenticity rather than confidentiality. Hence, it is thinkable to use BipBip directly for other authentication tasks where only short messages need to be authenticated. Apart from that, modes-of-operation exist that allow to authenticate longer messages. For instance, PMAC_TBC3k [Nai15], which is a parallel message authentication code. Here, the security degenerates quadratically with the number of messages authenticated but does not depend on the length of the messages (number of blocks per message). In particular, denoting with q the number of queries to the oracle and n the blocksize, the dominating term in the security bound is

$$\frac{0.5q^2}{(2^n - q)^2}$$

Hence, this seems to be a candidate when only a few messages need to be authenticated and the number of forgery attempts is restricted, e.g., communication stops if a few forgery attempts are detected. For instance, when plugging in $n = 24$ and $q = 2^{12}$, the advantage of an adversary bound by this term is $\approx 2^{-25}$. Hence, the best chance for an adversary to craft a forgery in this case seems to be still guessing the 24-bit tag.

Note that many so called beyond birthday bound secure MAC construction for tweakable blockciphers exist, e.g., also nonce-based ones that just have the number of verification attempts but not the number of correctly queried MACs as part of the security bound [CIL$^+$20]. To sum up, BipBip seems to be suited for authentication purposes requiring extremely low latency and small tags even apart from the use of C^3.

3.5 Security Claim

The maybe biggest notable feature of our security claim is that we do not set the goal of the adversary $\mathcal{A}$ to distinguish between E_K and an array of random permutations π, e.g., by making the claim in reference to strong tweakable pseudorandom permutation (**STPRP**) security. In particular, our analysis in Section 4.3.2 suggests that BipBip does not achieve **STPRP** security. Making the cipher secure against distinguishing attacks would require so many rounds and high latency that it would not be usable for C^3. However, we argue that C^3 does not need a cipher with such strong properties. What is of interest for C^3 is the probability with which an adversary can predict a plaintext from a ciphertext or vice versa. Even for an ideal 24-bit tweakable block cipher, this probability is $(2^{24} - q_{T_i})^{-1}$, assuming that an adversary has not recovered the secret key. Hence, the security of C^3 is only mildly influenced if this probability is increased to $(2^{23.5} - q_{T_i})^{-1}$.

As we can see from the description of the use of BipBip for C^3 in Section 3.4, we have to definitely assume that an attacker can read and choose ciphertexts and tweaks. However, to be on the safe side, in our security claim we assume that an attacker can also read and choose plaintexts. An attacker does not know the secret key and we require the key to be randomly picked from a uniform distribution.

As indicated by our analysis in Section 4, the more data is available, the higher is the number of rounds an attacker can distinguish. However, for practical applications, high data complexity attacks are often not a primary concern. For example, Qarma [Ava17], or Mantis [BJK+16] provide instances that only claim to be secure as long as the number of chosen plaintext/ciphertext pairs stays below 2^{30}. We do not give such a strict bound. However, as common with other low-latency block ciphers, e.g., Prince [BCG+12], Qarma [Ava17], or Mantis [BJK+16], we include in our security claim also the product of time and available data, e.g., $d \cdot t$. In contrast to the above schemes, we do not embed this limit generically by using the FX-construction [KR96].

As we can see in Section 2.5, we claim at most 96 bits of security against key recovery attacks, which is less than the more common 128 bits. The reason for this lies in the stringent low latency required due to the use in C^3. BipBip targeting 96 bits of security needs fewer rounds than a hypothetical version of BipBip aiming at 128 bits of security. We think that 96 bits of security are more than enough for the envisioned application in C^3. Here, an attacker just benefits from recovering the key as long as the key is used by a process. Hence, the window for a key recovery attack to gain a benefit is controlled by the application and can be made small.

3.6 Key schedule

For our security claim to hold, key recovery has to be hard. One key ingredient of our cipher for making key recovery hard is to have a larger masterkey than the claimed security level in bits. If we would have a masterkey matching the security level in bits, a small distinguishing advantage under a key guess can very likely be turned into an attack on the whole cipher by using brute-force encryptions according to the key ranking. Having a larger masterkey forces the correct key of a partial key guess to be ranked relatively high. Another concern we had is key bridging that has the potential to either lower the complexities of attacks or allow to cover more rounds.

As we can see in Figure 2, we have a 24-bit whitening key κ^0 and six 53-bit tweak-round keys κ^i. So, in total, we could use a 342 bit masterkey. However, we wanted the masterkey to match typical power of 2 storage requirements and hence, decided to use a 256-bit masterkey. As discussed in the introduction, even from an implementation perspective, it seems favorable to use a 256-bit masterkey and just select the 342 bits needed from it over just having a 96-bit, or 128-bit masterkey and use a stronger key schedule to derive tweak-round keys.

4 Preliminary Security Analysis

4.1 Classification of Attack Strategies

To ease the discussion for the upcoming analysis, we categorize attacks into 3 attack strategy classes. For all these classes, an attacker can know/choose ciphertexts, plaintexts, and tweaks. However, according to our security claim (Section 2.5), the secret masterkey K of the cipher is randomly chosen.

- **Single-tweak:** In this case BipBip acts as a block cipher and the attack has a natural data limit of $\leq 2^{24}$, as dictated by the block size of BipBip.

- **Multi-tweak:** Here the attacker utilizes the tweak to lift the data-limit of the cipher. The types of attacks that we consider in this category do not need to exploit tweak-relations, or tweak values apart from them being different. Because an attacker largely treats a change in tweak as a new selection of hard to predict data-round keys in this category, only the datapath provides security against such attacks.

- **Multi-chosen-tweak:** Here the attacker considers the actual value of the tweak or relations between tweaks. In contrast to the single-tweak and multi-tweak scenario that are largely agnostic to the specifics of the tweak schedule, here the attacker considers the details of the tweak schedule. Hence, the complex non-linear key schedule contributes to the cryptographic strength of BipBip for these attacks.

4.2 Round Key Guessing

In many attacks, guessing round keys in outer rounds and evaluating the guess against a distinguisher spanning inner rounds is an important part of an attack. For many (tweakable) block ciphers, an attacker is free to choose if it guesses the round keys at the beginning of the cipher, the end of the cipher, or a mixture between those two. This is also the case for BipBip for single-tweak attacks, where the data-round keys k_i do not change over the course of an attack and hence, the attacker can aim to recover, e.g., the last used data-round key k^{10}.

However, for multi-tweak and multi-chosen-tweak attacks, the situation is different. Here, the changing tweak leads to changing data-round keys k_i during the attack. Due to the use of a complex non-linear tweak schedule, it gets harder and harder to predict how a change in the tweak influences a data-round key k_i for rising i. Hence, we assume that an attacker has to perform guesses on the tweak-round keys of the first few rounds of the tweak schedule if it wants to guess more than the first 24 bits of κ^0. Furthermore, since, our tweak schedule has width of 53 bits, an attacker is forced to guess more key material on the tweak schedule to reveal the data-round keys used in the datapath.

4.3 Differential Cryptanalysis

In this section we discuss the resistance of BipBip against differential cryptanalysis. First, we will study attacks where the goal is to distinguish BipBip loaded with a secret 256-bit masterkey from an array of random 24-bit permutations. Then we will take a look at divide-and-conquer type attacks where part of the key is guessed, and differential propagation serves as a distinguisher to identify the correct guess.

4.3.1 Expected differential probabilities (EDP) of the datapath

The word-wise branch number of the linear layer of a core round is 4. This guarantees that in any single-tweak differential trail for $2r$ core rounds, we have at least $4r$ active S-boxes, and similarly for any single-tweak differential trail for $2r+1$ core rounds, we have at least $4r+1$ active S-boxes. We checked this lower bound for some differential trails of 3 to 6 rounds and found that it is the exact value for the minimum number of active S-boxes in any single-tweak differential trail.

Using the fact that the maximum DP for BipBipBox is 2^{-4}, together with the minimum number of active S-boxes in differential trails gives us an upper bound for EDP of any single-tweak differential trails in BipBip core rounds. It is 2^{-16r} and 2^{-16r-4} for $2r$ and $2r+1$ rounds, respectively. We emphasize that this is only an upper bound on the EDP of differential trails and does not imply the existence of such a differential trail.

However, we need to consider the clustering effect in the differentials and for that we wish to compute the EDP values of differentials over r core rounds. Here Markov cipher theory comes to the rescue [LMM91]. If we arrange the EDP (= DP for the round function) of all round differentials (a, b) in an array with $\mathrm{DP}(a, b)$ in row a and column b, we obtain what we call its Markov EDP matrix. As shown in [LMM91], the Markov EDP matrix for r rounds is then the simply the Markov EDP matrix to the r-th power.

For relatively simple round functions, such as those in BipBip, we can easily determine the elements of the matrix of EDPs of all differentials (a, b) over a round. However, its

Table 2: EDP of single-tweak differential trails, highest and lowest EDP found for single-tweak differentials and variances of EDP for 3 to 6 core rounds. n_r and n_s denotes the number of core rounds and the minimum number of active S-boxes in the differential trails. The *variance*, is the variance for EDP of all differentials with the same input difference.

n_r	n_s	upper bound on EDP of differential trails	minimum EDP of differentials	maximum EDP of differentials	maximum of variances	average of variances
3	5	2^{-20}	0	$2^{-19.74}$	$2^{-57.85}$	$2^{-59.46}$
4	8	2^{-32}	$2^{-24} - 2^{-29.28}$	$2^{-24} + 2^{-28.95}$	$2^{-73.44}$	$2^{-75.22}$
5	9	2^{-36}	$2^{-24} - 2^{-37.08}$	$2^{-24} + 2^{-37.12}$	$2^{-89.31}$	$2^{-90.99}$
6	12	2^{-48}	$2^{-24} - 2^{-44.79}$	$2^{-24} + 2^{-45.07}$	$2^{-105.05}$	$2^{-106.74}$

size, $2^{24} \times 2^{24}$, is prohibitively large. Therefore, to compute the EDP of differentials over r core rounds of BipBip, we used a technique introduced in [ELR20], the so-called simple word-by-word method. In this technique, we consider the S-box layer S as a sequence of four sub-layers S_i with $0 \leq i < 4$. Each sub-layer S_i operates on a 24-bit state and applies a single S-box to the i-th word of the state, and the identity to the other words. This means the input and output differences in a differential for these sub-layers match in 3 word positions. In the word-by-word method of [ELR20], we need to do $2^{18\times 2}$ matrix multiplications of matrices of size $2^6 \times 2^6$ per S_i sub-layer. Hence, the computation cost for computing EDP of *all* $2^{24\cdot 2}$ differentials is reduced from $2^{24\cdot 3} = 2^{72}$ per round to $2^{18\cdot 2} \cdot 2^{6\cdot 3} \cdot 4 = 2^{56}$ per round. Moreover, to fit our computations in reasonable memory, we start from a given input difference and compute the EDP values of differentials from this input difference to all output differences. This method requires only 2^{24} memory blocks. Still, the number of non-zero input differences $2^{24} - 1$ is very large for our resources and we only made the computation for a subset of them: those that have at most 3 active words in the input difference totaling to about 2^{20} input differences. For more information on this technique, we refer to [ELR20].

In Table 2, we report on some findings on EDP of those 2^{44} differentials for r core rounds of BipBip with $r \in \{3, 4, 5, 6\}$. It turns out that in distinguishing attacks the deviation of the EDP of differentials from the expected values in case of a random permutation $1/(2^{24} - 1)$ is relevant.

4.3.2 BipBip, the Markov cipher

In a first approximation we will assume that the tweak randomly selects a round-key-sequence. When the round keys are assumed to be independent, the datapath forms a Markov cipher [LMM91]: the expected differential probability (EDP) of a differential trail is the product of its round differentials. The fact that the master key has 256 bits and that in total there are only 10 round keys in between the rounds supports this assumption.

In a classical differential attack, one uses a differential (a, b) over the cipher with last round removed with high $\text{EDP}(a, b)$ as a distinguisher. One queries the tweakable block cipher with plaintexts in pairs with a given difference a and observes the difference at the input of the last round by guessing part of the last round key. If the guess is correct, a fraction of (about) $\text{EDP}(a, b)$ of the pairs will exhibit the difference b and if not, it is likely to be less. For such an attack to work some conditions must be satisfied. First, $\text{EDP}(a, b)$ needs to be relatively high and of all 2^{n-1} pairs, the number of pairs with input difference a that give an output difference b shall be close to $2^{n-1}\text{EDP}(a, b)$ for quasi all keys.

In $\text{BipBip}_{x,y,z}$ with $y \geq 4$ such an attack cannot work. As can be seen in Table 2, we see that even after 4 rounds the EDP of all differentials is similar and very close to $1/(2^{24} - 1)$. This suggests that the number of pairs is $2^{23}/(2^{24} - 1) \approx 1/2$, while the actual number of pairs will be an integer. One may ask what such $\text{EDP}(a, b)$ values actually mean.

Fortunately, the case of differential attacks on small Markov ciphers was investigated in [AL12] and that analysis is fully applicable to BipBip.

4.3.3 Distributions of the number of pairs in a differential

We denote number of pairs in a differential (a, b) by $\mathrm{N}(a, b)$. In [AL12] it is argued that over all round-key-sequence values $\mathrm{N}(a, b)$ has a binomial distribution that is very close to Poisson distribution with $\lambda = 2^{23}\mathrm{EDP}(a, b)$. This follows from the fact that the set of pairs in a differential (a, b) are the union of the sets of pairs in the trails with initial difference a and final difference b. We consider the case where the trails Q have weight w higher than 23. Such a trail has the following distribution: 1 pair for a fraction $2^{23-\mathrm{w}(Q)}$ of the round-key-sequences and no pairs for remaining fraction $1 - 2^{23-\mathrm{w}(Q)}$ (and in some cases it has 2 or more pairs for a very very small fraction of the round-key-sequences). If the subsets of the round-key-sequences for which the trails have a pair are independent, then the distribution of the number of pairs in (a, b) is the convolution of the distribution of the number of pairs of its trails. This is the sum of the probabilities of rare events and hence will be very close to a Poisson distribution. And as $\mathrm{EDP}(a, b) = \sum_{Q \text{ in } (a,b)} 2^{-\mathrm{w}(Q)}$, this Poisson distribution has $\lambda = 2^{23}\mathrm{EDP}(a, b)$.

For a random permutation the distribution of $\mathrm{N}(a, b)$ was investigated in [O'C93]: it has a Poisson distribution with $\lambda = 2^{23}/(2^{24} - 1)$.

We experimentally verified these distributions for 4 core rounds of BipBip by computing $\mathrm{N}(a, b)$ for some specific values of a and b for 14×2^{12} random round keys (so the 3 round keys in between the 4 core rounds). The resulting distribution of $\mathrm{N}(a, b)$ for fixed a and b over these round keys is a sampling of the distribution over all round key values, that is supposed to have a Poisson shape with $\lambda = 2^{23}\mathrm{EDP}(a, b)$. We did the experiment for $(a, b) = (\texttt{0x00002e}, \texttt{0x220020})$ that has $2^{23}\mathrm{EDP} = 0.516122$. The distribution we obtained experimentally has the characteristic shape of Poisson distribution and has mean 0.516095 and so is only at a distance of 0.000027 (= 0.054%) of the λ value based on the EDP. We provide the full distribution in Table 10 and Figure 13.

4.3.4 Single-tweak differential distinguishing attack

We assume an attacker is in a setup where she must distinguish between BipBip for a given tweak value and a random permutation and has obtained from her queries a number of ciphertext-plaintext pairs for the given tweak value. In the worst case the attacker may have the whole codebook for that tweak value. We assume this case.

From ciphertext-plaintext pairs the adversary can compute the number of pairs in differentials (a, b) in some set Ω to base her decision on. As a matter of fact, she can compute $\mathrm{N}(a, b)$ for all $(2^{24} - 1)^2$ differentials (a, b), so Ω is the set of all differentials over the cipher.

The array $\mathrm{N}(a, b)$ is a sampling of a distribution: the one of BipBip determined by the array of values $\mathrm{EDP}(a, b)$ or the one of a random permutation where $\mathrm{N}(a, b)$ has a Poisson distribution with $\lambda = 2^{23}/(2^{24} - 1)$ for all differentials (a, b). As investigated in [AL12], the optimum criterion to decide which is the distribution the sample comes from, is the log-likelihood ratio. This is determined by the λ values of the Poisson distributions:

$$l' = \sum_{(a,b)\in\Omega} \mathrm{N}(a, b) \log\left(\frac{\mathrm{EDP}(a, b)}{1/(2^{24} - 1)}\right) \approx \sum_{(a,b)\in\Omega} \mathrm{N}(a, b)\frac{\mathrm{EDP}(a, b) - 1/(2^{24} - 1)}{1/(2^{24} - 1)},$$

where the approximation makes use of the fact that $\mathrm{EDP}(a, b)$ is very close to $1/(2^{24} - 1)$. For simplicity we can omit the denominator, yielding:

$$l = \sum_{(a,b)\in\Omega} \mathrm{N}(a, b)\left(\mathrm{EDP}(a, b) - \frac{1}{2^{24} - 1}\right).$$

Clearly, l, and hence the advantage, is fully determined by the array of values $\text{EDP}(a, b)$.

To determine the optimum guessing strategy, we need to compute the distributions for l for the case of a random permutation and for the cipher.

For both cases l is the weighted sum of $(2^{24} - 1)^2$ variables with each a Poisson distribution. Due to the central limit theorem, this results in Normal distributions.

For a random permutation all variables have $\lambda = 2^{23}/(2^{24} - 1)$, resulting in a Normal distribution with mean:

$$\frac{2^{23}}{2^{24}-1} \sum_{(a,b)\in\Omega} \left(\text{EDP}(a,b) - \frac{1}{2^{24}-1}\right) .$$

If Ω consists of the differentials (a, b) for a given number of input differences a but all output differences b, this mean is zero as for any a, $\sum_b \text{EDP}(a, b) = 1$. And there are $2^{24} - 1$ non-zero values of b. We will assume that computing the values $\text{N}(a, b)$ for a given differential from the queries is not less work than computing the values $\text{N}(a, b)$ for a given input difference a and all output differences b.

For BipBip the mean is:

$$\sum_{(a,b)\in\Omega} 2^{23}\text{EDP}(a,b) \left(\text{EDP}(a,b) - \frac{1}{2^{24}-1}\right) ,$$

resulting in the following difference of means:

$$\text{Dom} = 2^{23} \sum_{(a,b)\in\Omega} \left(\text{EDP}(a,b) - \frac{1}{2^{24}-1}\right)^2 .$$

For the random permutation the variance is:

$$\text{Var} = \frac{2^{23}}{2^{24}-1} \sum_{(a,b)\in\Omega} \left(\text{EDP}(a,b) - \frac{1}{2^{24}-1}\right)^2 ,$$

and for BipBip we get a variance very close to this due to the fact that $\text{EDP}(a, b)$ is very close to $1/(2^{24} - 1)$.

The optimum strategy of the adversary is now to guess that it is BipBip if l is higher than the average of the two means, namely $\text{Dom}/2$, and a random permutation otherwise. The advantage can be derived from:

$$\Pr(l > \text{Dom}/2 \mid \mathit{BipBip}) = \frac{1}{2}\left(1 + \text{erf}\left(\frac{\text{Dom}}{\sqrt{2\text{Var}}}\right)\right) .$$

So the advantage is $\text{erf}\left(\frac{\text{Dom}}{\sqrt{2\text{Var}}}\right)$. Filling in yields:

$$\text{Adv} = \text{erf}\left(\sqrt{2^{22}(2^{24}-1) \sum_{(a,b)\in\Omega} \left(\text{EDP}(a,b) - \frac{1}{2^{24}-1}\right)^2}\right) .$$

For very small values of x, $\text{erf}(x) \approx (2/\pi)x$ and we have

$$\text{Adv} \approx \frac{2^{24}}{\pi} \sqrt{\sum_{(a,b)\in\Omega} \left(\text{EDP}(a,b) - \frac{1}{2^{24}-1}\right)^2} . \quad (1)$$

In this formula the expression in the square root is the Euclidean distance between the $\text{EDP}(a, b)$ vector with coordinates in Ω and the vector with $\#\Omega$ components equal to

$1/(2^{24}-1)$. This distance can be seen as a *ripple* of the EDP around the average and it can be made smaller by adding rounds, as illustrated in Table 2. We will refer to it as R_Ω.

From Equation 1 we see that claiming 96 bits of SPRP security would be quite demanding. Given an attack complexity X, this would imply the distinguishing advantage can be at most 2^{-96+X}, or R_Ω can be at most 2^{-119+X}. The attack at hand has $q = 2^{24}$ queries and the computation of the quantity can have computational cost t up to 2^{48}, depending on the size of Ω. We consider that Ω contains for each input difference a it covers, all $2^{24}-1$ output differences b, i.e., $\Omega = \{(a,b) \mid a \in A, b \in \mathbb{F}_2^{24} \setminus \{0\}\}$.

The attack requires the adversary to compute differential distribution for all the differences in Ω. Since computing differential distribution for each fixed input difference a and all the output differences needs 2^{24} operations, the attack complexity will be $2^{24}|A|$. This would require the ripple to be smaller than $2^{-119+24}|A| = 2^{-95}|A|$. Note that for considered Ω, we have

$$\mathrm{R}_\Omega = \sqrt{(2^{24}-1)\sum_{a\in A}\sigma_a^2} \approx 2^{12}\sqrt{\sum_{a\in A}\sigma_a^2}\,. \tag{2}$$

that σ_a^2 denotes the variance for EDP values of all $2^{24}-1$ differentials with the same input difference a.

The best scenario for the attacker would be to consider A to include only a single input difference a_{max} which has the highest value for σ_a^2, i.e., $a_{max} = \arg\max_{a\in A}\sigma_a^2$. Consequently, if $\sigma_{a_{max}}^2 > 2^{-214}$, BipBip can be distinguished from a random permutation with higher advantage as implied to achieve a security strength of 96 bits.

In Table 2, we report about $\sigma_{a_{max}}^2$ values, called *maximum of variances*, and also about average of σ_a^2 values, called *average of variances*, that is equal to the variance of EDPs for all differentials we tested. Extrapolating Table 2, a 96-bit STPRP security claim would require 13 core rounds.

4.3.5 Multi-tweak differential distinguishing attack

An adversary may be able to obtain the full codebook for multiple tweak values. This can be seen as multiple samples of the distribution and the log-likelihood can easily be generalized. The adversary just computes the sum of the l values for each of the obtained codebooks and adds them. For M codebooks, this multiplies both the Dom and Var by N, and hence the advantage by $\sqrt{N}$ relative to a single codebook. Hence the advantage only increases with the square root of the data complexity and the STPRP security strength in the multi-tweak setting is determined by the single-tweak attack.

4.3.6 Attacks with key guessing

An attacker can peel off some rounds by guessing a part of the round-key-sequence or a part of the master key K. A correct key guess removes some rounds from the cipher. An incorrect key guess has the effect of adding rounds and will result in a smaller ripple. One can use the difference between the two ripples in defining the log-likelihood ratio to rank the key guesses. If the correct key guess comes out on top, the adversary has successfully recovered part of the key and can attack the reduced-round cipher to determine the rest of the key. Even if that is not the case but the correct key is ranked high enough in the list, the adversary may have an advantage. This depends on the effort to find the rest of the key.

We now study the following: Given a certain variance of the EDP and number of guessed key bits, what is the required effort to have the correct key guess to come out on top? For simplicity we will assume that the adversary has the whole codebook for N tweak values. If we assume an incorrect key guess has no ripple at all, the expected fraction of wrong key guesses that will give a higher l value than the correct key is given by

$$\Pr(l > \mathrm{Dom} \mid \mathrm{random}) = \frac{1}{2}\left(1 - \mathrm{erf}\left(\frac{\mathrm{Dom}}{\sqrt{2\mathrm{Var}}}\right)\right)$$

If we want this probability to be small enough that very likely there are no incorrect key guesses with l value higher than the correct one, $\frac{\mathrm{Dom}}{\sqrt{2\mathrm{Var}}}$ must be relatively large. For large enough values we have $\mathrm{erf}(x) \approx e^{-x^2}/(2\sqrt{\pi}x)$. If $x = 5$ the probability is of the order 2^{-32} but for $x = 10$ it becomes of the order 2^{-139}. In other words, if $\frac{\mathrm{Dom}}{\sqrt{2\mathrm{Var}}}$ is around 10 and the length of the guessed key is significantly smaller than 139, the correct key comes likely out on top.

As $\frac{\mathrm{Dom}}{\sqrt{2\mathrm{Var}}} \approx 2^{23}\sqrt{N}\mathrm{R}_\Omega$, this would require $\sqrt{N} \geq 2^{-19}/\mathrm{R}_\Omega$. Using 2, this translates to

$$N \geq \frac{2^{-62}}{\sum_{a \in A} \sigma_a^2} .$$

For example, for 5 core rounds plus 1 shell round this would require $N \geq 2^{105.05-62-24} > 2^{19}$. For x guessed key bits, this would correspond to a computational cost of about $2^x 2^{16} 2^{24} \approx 2^{x+40}$. As peeling off the shell rounds requires at least guessing 120 key bits, $x \geq 120$ and hence key recovery using this type of attack appears out of reach.

4.4 Linear Cryptanalysis

In this section we discuss the resistance of BipBip against linear cryptanalysis. Similar to the case for differential cryptanalysis, we first study distinguishing attacks and then divide-and-conquer type attacks where part of the key is guessed.

4.4.1 Expected Linear Potential (ELP) of the datapath

Similar to the Markov EDP matrix, we can build the Markov ELP matrix of a round function by arranging the ELP (=LP for a round function) values of all linear approximations (a, b) in a matrix with $\mathrm{LP}(a, b)$ in row a and column b. This is the correlation matrix of the round with its entries squared [DR20]. The ELP matrix for r rounds is then simply the Markov ELP matrix to the r-th power.

For relatively simple round functions, such as those in BipBip, we can easily determine the elements of the Markov ELP matrix and for finding multiple-round ELP values we can use the same techniques we applied for the Markov EDP matrix, see Section 4.3.2.

As can be seen in Table 3, we see that after 4 rounds the ELP of all linear approximations is similar and very close to $1/(2^{24} - 1)$. As for the case of EDP values, one may ask what $\mathrm{ELP}(a, b)$ values actually mean.

4.4.2 Distributions of the correlation value of linear approximations

$\mathrm{C}(a, b)$, the correlation of a linear approximation (a, b) depends on the round-key-sequence. It was proven, e.g., in [DR20], that in a key-alternating cipher (such as BipBip), $\mathrm{C}(a, b)$ has a distribution with mean 0 and variance $\mathrm{ELP}(a, b)$ over the round-key-sequences. If the linear approximation is composed of many linear trails this distribution will be close to a Normal distribution. For a random permutation it was shown in [DR07] that for any linear approximation $\mathrm{C}(a, b)$ has a Normal distribution with mean 0 and variance 2^{-n}.

The mean of the distributions is 0 in both cases and hence cannot be used to distinguish them. The deviation between the two is in the variance and that can be assessed by comparing the square of the correlation: the LP.

Table 3: ELP of single-tweak linear trails, highest and lowest ELP found for the single-tweak linear approximations and variances of ELP for 3 to 6 core rounds. n_r and n_s denotes the number of core rounds and the minimum number of active S-boxes in the linear trails. The *variance*, is the variance for ELP of all linear approximations with the same output mask.

n_r	n_s	upper bound on ELP of linear trails	minimum ELP of approximations	maximum ELP of approximations	maximum of variances	average of variances
3	5	2^{-20}	$2^{-74.00}$	$2^{-21.20}$	$2^{-58.54}$	$2^{-59.75}$
4	8	2^{-32}	$2^{-24} - 2^{-29.05}$	$2^{-24} + 2^{-28.85}$	$2^{-74.38}$	$2^{-75.55}$
5	9	2^{-36}	$2^{-24} - 2^{-37.17}$	$2^{-24} + 2^{-37.21}$	$2^{-90.20}$	$2^{-91.37}$
6	12	2^{-48}	$2^{-24} - 2^{-45.03}$	$2^{-24} + 2^{-45.03}$	$2^{-106.09}$	$2^{-107.18}$

For a random 24-bit permutation, the distribution of $\mathrm{LP}(a,b)$ has mean $1/(2^{24}-1)$ and variance 2^{-47} [DR07]. For a block cipher the distribution of $\mathrm{LP}(a,b)$ has mean $\mathrm{ELP}(a,b)$, but the variance depends on the number of linear trails in the approximation and their weight. For a linear approximation with many trails, the variance is close to $2(\mathrm{ELP}(a,b))^2$, as argued in [DR07].

We experimentally verified the distribution of the correlation of a linear approximation for 4 core rounds of BipBip by computing $\mathrm{C}(a,b)$ for some specific masks a and b for 40×2^{12} random round keys (so the 3 round keys in between the 4 core rounds). The resulting distribution of $\mathrm{C}(a,b)$ for fixed a and b over these round keys is a sampling of the distribution over all round key values, that is expected to have a Normal distribution with variance $\mathrm{ELP}(a,b)$. We did the experiment for $(a,b) = (\texttt{0x0008c0}, \texttt{0x1a0974})$ that has $2^{24}\mathrm{ELP} = 1.03477$. The distribution we obtained experimentally looks like a sampling of a Gaussian distribution and has variance 1.034764 and so is only at a distance of $(= 0.66 \times 10^{-5})$ of the variance based on the ELP. We provide a figure of the distribution in Figure 14.

4.4.3 Single-tweak distinguishing linear attack

We assume an attacker that has obtained from her queries the whole codebook for some tweak value and computes from these $\mathrm{LP}(a,b)$ for a large set of linear approximations (a,b). The array $\mathrm{LP}(a,b)$ is a sampling of a distribution, either that for BipBip or that for a random permutation. We can, as in the case of the differential attack, compute the log-likelihood ratio:

$$l' = \sum_{(a,b)\in\Omega} \mathrm{LP}(a,b) \log\left(\frac{\mathrm{ELP}(a,b)}{1/(2^{24}-1)}\right) \approx \sum_{(a,b)\in\Omega} \mathrm{LP}(a,b) \frac{\mathrm{ELP}(a,b) - 1/(2^{24}-1)}{1/(2^{24}-1)},$$

where the approximation makes use of the fact that $\mathrm{ELP}(a,b)$ is very close to $1/(2^{24}-1)$. Omitting the denominator yields:

$$l = \sum_{(a,b)\in\Omega} \mathrm{LP}(a,b) \left(\mathrm{ELP}(a,b) - \frac{1}{2^{24}-1}\right).$$

For both cases l is the weighted sum of $(2^{24}-1)^2$ variables resulting in Normal distributions. For a random permutation the mean is:

$$\frac{1}{2^{24}-1} \sum_{(a,b)\in\Omega} \left(\mathrm{ELP}(a,b) - \frac{1}{2^{24}-1}\right),$$

and for BipBip it is

$$\sum_{(a,b)} \mathrm{ELP}(a,b)\left(\mathrm{ELP}(a,b) - \frac{1}{2^{24}-1}\right) ,$$

resulting in the following difference of means:

$$\mathrm{Dom} = \sum_{(a,b)\in\Omega} \left(\mathrm{ELP}(a,b) - \frac{1}{2^{24}-1}\right)^2 .$$

For the random permutation the variance is:

$$\mathrm{Var} = 2^{-47} \sum_{(a,b)\in\Omega} \left(\mathrm{ELP}(a,b) - \frac{1}{2^{24}-1}\right)^2 ,$$

and for BipBip the variance very close to this as $\mathrm{ELP}(a,b) \approx 1/(2^{24}-1)$.

As in the case of the single-tweak differential attack, the advantage is $\mathrm{erf}\left(\frac{\mathrm{Dom}}{\sqrt{2\mathrm{Var}}}\right)$. Filling in and using the approximation $\mathrm{erf}(x) \approx (2/\pi)x$ for small x yields

$$\mathrm{Adv} \approx \frac{2^{24}}{\pi} \sqrt{\sum_{(a,b)\in\Omega} \left(\mathrm{ELP}(a,b) - \frac{1}{2^{24}-1}\right)^2} . \tag{3}$$

Remarkably, Equation 3 is identical to Equation 1 with ELP values taking the place of the EDP values. Moreover, even the results in Table 3 are very similar to those in Table 2, with the values for ELP being slightly smaller.

The similarity makes that the discussion of the exploitation of these advantages for linear cryptanalysis is essentially the same as for differential cryptanalysis, with two differences.

The first difference is about Ω: in differential attacks it contains all the output differences for some input differences while in linear attacks it contains all the input masks for some output masks: the attacker computes the LP values for a given number of output masks for all input masks by doing a Walsh-Hadamard transform of the Boolean function that represents an output mask applied to the queried codebook. The second difference is the computational cost: a Walsh-Hadamard transform computation requires $24 \cdot 2^{24}$, so a factor 24 more than the computational cost for computing an array of N values. This makes the differential approach the most efficient of the two attacks.

4.5 Multi-chosen-tweak Differential and Linear Cryptanalysis

This section complements Section 4.3.2 and Section 4.4 with results for differential and linear cryptanalysis that consider tweak relations. In this scenario, a big contributor to BipBip's security is the tweak schedule. Hence, we will make statements on differential trails in the tweak schedule first.

4.5.1 Differential trail search in the tweak schedule

We investigate EDP of differential trails of the tweak schedule using *differential trail cores*. We use the approach introduced by Mella et al. in [MDV17] to search efficiently for those differential trail cores. We recall that a differential trail core is a differential trail where the first and last difference are unspecified. It is shown that the weight of a differential trail is the minimum weight of all trail cores compatible with the given differential trail.

Table 4: Lower bounds on the weight of differential trails for tweak schedule. Note that the order of rounds is the same as depicted in Figure 2.

rounds	1	2	3	4	5	6	7	8	9
weight	2	8	24	$\geq$35	$\geq$43	$\geq$45	$\geq$53	$\geq$55	$\geq$63

We perform an exhaustive trail search up to 5 rounds. Thereby, we first compute all 2-round trail core (using two rounds of G round function) and then we extend them by one round in forward or backward directions to obtain all possible 3-round trail cores. For the case of 4- and 5- round differential trails, we use the same method as for 3-round trails; we extend the 3-round trail cores in the forward direction by one and two rounds, respectively.

Table 4 shows the results on the minimum weight of the differential trails over tweak rounds that the weight of a differential trail Q with an EDP equals p is defined as $w = -\log_2 p$. Note that this trail search is done for the same order of the rounds as tweak schedule that is depicted in Figure 2. That is in the case of 5 rounds, it starts with a round that is without a linear layer, then there are three rounds with a strong linear layers and one with a light linear layer.

We find only one trail of weight 24 for 3 rounds and there is no trail with a weight less than 24. For 4 rounds, we find no trail of weight of below 35. Therefore, we conclude that there is no trail of weight below 43 after 5 rounds.

4.5.2 Differential cryptanalysis

As we know from our trail search of Section 4.5.1, no differential trails for the tweak schedule with a high EDP exist. This means that it is infeasible for an attacker to force a certain differential pattern on all data-round keys k^i for a certain tweak difference ΔT. However, we do not wait until the whole tweak schedule is computed before extracting k^i, rather the extraction is a continuous process. Hence, an attacker can predict and influence the differences of the first few data-round keys with a rather high probability, while gradually losing this ability towards the data-round keys that are extracted at last. Therefore, we also have a look at the number of active S-boxes with respect to random difference patterns in the data-round keys.

Simple counting. For the simple counting, we assume that differences induced by all possible 2^{24} differences Δk^i can and will always be canceled with Δk^{i+1}. This leads to having always at least 4 inactive S-boxes per 2 rounds as shown in the example of Figure 3.

However, due to the construction of our tweak schedule, an attacker that has limited influence on the difference Δk^i has only limited influence on Δk^{i+1}, and hence, cannot enforce a differential pattern that can cancel induced differences. So, we overestimate the capabilities of an attacker here.

Based on this model, we calculated that for 5 rounds of BipBip, out of the 2^{72} possible

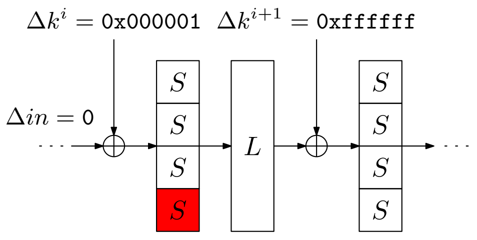

Figure 3: Example for model that assumes the differences always cancel. The active S-boxes are in red.

key differences, only the fraction of 2^{-32} difference patterns can lead to less than 6 active S-boxes in the datapath (details in Table 5). For 7 rounds of BipBip, the same holds for the fraction of 2^{-54} difference patterns out of 2^{96} possible ones.

Table 5: Probability that a random data-round key difference leads to x active S-boxes for 5 R$'$, or 5 R

Number of active S-boxes	0	1	2	3	4	5
Probability	$2^{-72.0}$	$2^{-62.4}$	$2^{-54.0}$	$2^{-46.2}$	$2^{-39.1}$	$2^{-32.4}$

Counting for shell round functions R$'$*.* Not every differential pattern induced by Δk^i can be canceled by every differential pattern induced by Δk^{i+1}. For instance, a differential pattern that leads to a single active S-box in round i cannot be canceled by 18 bits of Δk^{i+1} of a shell round. Even if we assume the 6 bits of Δk^{i+1} that can cancel the differences of single active S-box always cancel this difference, the remaining 18 bits can still lead to active S-boxes in this round as indicated in the example shown in Figure 4. We can count this pattern up 4 active S-boxes, where we assume that all 24 bits of Δk^{i+1} always cancel out the differences as before.

We implemented this model for the shell round functions R$'$. For 3 rounds of R$'$, we calculated that out of all possible 2^{72} data-round key differences, only a fraction of 2^{-14} difference patterns lead to less than 6 active S-boxes (details in Table 6). For hypothetical 4 and 5 consecutive rounds of R$'$, this fraction would be 2^{-30} and 2^{-51}, respectively.

Interpretation of Results. Let us combine the results of Table 5 and Table 6 and interpret them for BipBip$_{0,5,3}$. For random data-round key differences, we have following probabilities:

- 0 active S-boxes: $2^{-72} \cdot 2^{-72} = 2^{-144}$
- 1 active S-box: $2^{-62} \cdot 2^{-72} + 2^{-72} \cdot 2^{-57} = 2^{-129}$
- 2 active S-boxes: $2^{-72} \cdot 2^{-43} + 2^{-72} \cdot 2^{-54} + 2^{-62} \cdot 2^{-57} = 2^{-115}$
- 3 active S-boxes: $2^{-72} \cdot 2^{-32} + 2^{-72} \cdot 2^{-46} + 2^{-62} \cdot 2^{-43} + 2^{-54} \cdot 2^{-57} = 2^{-103}$
- 4 active S-boxes: $2^{-72} \cdot 2^{-22} + 2^{-62} \cdot 2^{-32} + 2^{-54} \cdot 2^{-43} + 2^{-46} \cdot 2^{-57} + 2^{-39} \cdot 2^{-72} = 2^{-93}$
- 5 active S-boxes: $2^{-72} \cdot 2^{-14} + 2^{-62} \cdot 2^{-22} + 2^{-54} \cdot 2^{-32} + 2^{-46} \cdot 2^{-43} + 2^{-39} \cdot 2^{-57} + 2^{-32} \cdot 2^{-72} = 2^{-83}$

Above results show us that considering the difference induced by data-round keys as random for a certain difference in the tweak will not allow for exploitable differential trails with high probability. In particular, for the results of Table 5 we have taken a very favorable model for the attacker that assumes that the S-boxes of every second

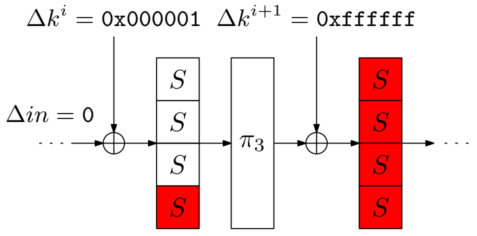

Figure 4: Example for model considering π_3. Active S-boxes in red.

Table 6: Probability that a random data-round key difference leads to x active S-boxes for 3 R′.

Number of active S-boxes	0	1	2	3	4	5
Probability	$2^{-72.0}$	$2^{-57.0}$	$2^{-43.8}$	$2^{-32.1}$	$2^{-22.0}$	$2^{-14.4}$

round are inactive independent of the actual induced differences by the data-round keys. On the other hand, we know from Section 4.5.1 that also predicting differences induced by data-round keys in the datapath is not a viable strategy, since no high probability differential trails for the tweak schedule exist. Taking all together, we expect BipBip to withstand multi-chosen-tweak differential attacks.

4.5.3 Linear cryptanalysis

For an attack based on linear cryptanalysis that aims to also exploit tweak relations, we have a considerable increase in the available data complexity from 2^{24} to 2^{64} considering a 40-bit tweak. This means that an attacker can potentially exploit linear trails with a smaller correlation than in single-tweak attacks. However, this comes at the cost that linear trails have to also take our non-linear tweak schedule into account.

Compared to the differential case, the additional consideration of the tweak schedule does not influence the minimum number of linearly active S-boxes in the datapath. Rather, we consider additionally the approximation of the data-round keys via the tweak schedule. Interestingly, it is also possible to approximate the influence of data-round keys of round r with the influence of $r+1$ with a trail just involving the part of the tweak schedule between those rounds. In other words, not necessarily all rounds of the tweak schedule have to be linearly active, which results to a certain degree in the dual of the differential case.

If we consider 4 R plus 3 R′ rounds, we have at least 11 active S-boxes in the datapath. In addition, we conservatively estimate that we only need three active bits at the input of χ for 3 rounds of the tweak schedule to fulfill the conditions imposed by a trail of the datapath. Hence, the best linear trail has a correlation of at most $2^{-(11\cdot 4+3)} = 2^{-47}$. Assuming that this trail is dominating, we need in the order of 2^{94} data to use it as a distinguisher in a potential key recovery attack. In addition, we expect that especially the tweak schedule involves much more complicated trails. Therefore, we expect that BipBip is secure against this type of attack.

4.6 Combined Differential and Linear Attacks

With combined differential and linear attacks, we denote attacks that do not directly evaluate the differential or linear approximation over a cipher E_w. In contrast, they aim to provide better distinguishers by dividing the cipher into subciphers, e.g., $E_w = E_e \circ E_b$ and combining shorter differential and linear trails over the subciphers. One prime example for such an attack is the Boomerang attack [Wag99] that combines differentials over E_b with differentials over E_e. Denoting the DP of the differential over E_w, E_b, and E_e with p_w, p_b, and p_e, respectively, the number of pairs needed to get a distinguishing advantage is approximately $(p_b p_e)^{-2}$ compared to p_w^{-1}. In the case of differential-linear attacks [LH94], we use a differential for the first part of the cipher and a linear trail for the second part. Denoting the correlation of the linear trail with c_e for E_e, the data complexity needed to gain a distinguishing advantage is approximately $(p_b c_e^2)^{-2}$ compared to c_w^{-2} when using a linear approximation for the whole cipher.

From the observations on the expected data complexities, it becomes clear that combined attacks are advantageous over differential and linear attacks for ciphers where the DP of differentials and the correlation of linear trails massively drops with a rising number of rounds. For BipBip, we do not see such a massive drop. When looking at BipBip, the number of active S-boxes for 2 core rounds in the best linear, or differential trail is 4. If we double the number of core rounds to 4, the number of active S-boxes doubles to 8.

To give a more illustrative example, let us compare a differential distinguisher for 5 core rounds of BipBip with a Boomerang distinguisher for 5 core rounds of BipBip in a single-tweak or multi-tweak setting. A differential trail has at least 9 active S-boxes for 5 core rounds of BipBip. For the Boomerang distinguisher, we can combine a 3 core round trail (5 active S-boxes) with a 2 core round trail (4 active S-boxes). Since we have $(p_b p_e)^{-2}$ versus p_w^{-1} for the data complexity, the data complexity needed to evaluate the Boomerang distinguisher is more-or-less equivalent to 18 active S-boxes in a differential distinguisher. Even if it would be possible to connect the Boomerang over two middle rounds with probability one using advanced techniques, we still combine a two-round trail (4 active S-boxes) with a 1 round trail (1 active S-box) giving 10 equivalent active S-boxes for a differential distinguisher. Analogue arguments can be made in the case of differential-linear attacks. Therefore, we conjecture that combined attacks are unlikely to perform much better than differential, or linear attacks.

4.7 Algebraic Attacks

In this category, we examine cryptanalytic techniques that work by exploiting knowledge on the algebraic degree of a cipher or isolated parts of the cipher. This category of attacks is rather large and contains higher-order differential [Lai94], integral [KW02], and cube [DS09] attacks, as well as the division property [Tod15]. In this section, we will first have a look at the algebraic degree of the datapath relevant for single-tweak, and multi-tweak attacks. After that, we have a look at the algebraic degree of the tweak schedule providing additional security in the multi-chosen-tweak scenario.

4.7.1 Algebraic degree of the datapath

Since the width of the datapath is so small, we are able to directly evaluate certain input sets. Hence, with the help of [HLLT20, HLLT21], we are able to show that all monomials of degree 23 are present in each linear combination of the output bits after a certain number of rounds. Moreover, we can show that sums of any selection of output bits are key dependent for any selection of input sets that are non-empty and not the full set.

Upper bounds on the degree. First, let us have a look at the upper bound we can give on the degree following [BCD11]. Since we have decided to use 6-bit S-boxes, we give the upper bounds on the degree assuming that an S-box has degree 5, 4, 3, or 2 for all output bits in Table 7.

Table 7 shows us that it is possible to reach a degree of 23 after 4 rounds for 6-bit S-boxes with a degree of 3 or 4 and after 3 rounds for 6-bit S-boxes with a degree of 5. However, those bounds are upper bounds and the degree after 5 rounds could be smaller, or high-degree terms could be missing. Furthermore, we use an S-box that has algebraic degree 3, but on some output bits only a degree of 2. Hence, we have a closer look at the monomials that appear in the ANF of 5 rounds of BipBip.

Checking for degree 23 monomials. In the spirit of [HLLT20], we check that for 5 rounds of R, each possible linear combination of the output bits can contain each possible degree 23 monomial of the input bits assuming independent data-round keys. Since we deal with monomials of degree 23, we can directly compute the evaluation of the superpoly [Din21] for each monomial for a certain data-round key.

Table 7: Upper bounds on the ANF degree of outputs after r rounds using 6-bit S-boxes with degree of d in all the coordinates of the S-box.

Rounds (r)	Degree (d)			
	2	3	4	5
1	2	3	4	5
2	4	9	16	21
3	8	21	22	23
4	16	23	23	23
5	22	23	23	23

Essentially, if we see an output bit flip for an evaluation of the superpoly using different data-round keys, we know that the superpoly for this specific monomial is not empty for this bit. To check if also all linear combinations of the output bit contain a specific monomial, we fix one single monomial and evaluate the higher-order differential of all 24 output bits under x different random data-round keys and form a $24 \times x$ matrix. If the rank of this matrix is 24, we know that the specific monomial of degree 23 is present in each linear combination of the output bits at least for one data-round key. We implemented this check for all 24 possible degree 23 monomials with $x = 13943$ and verified that all matrices have rank 24 for 6 core datapath rounds, 6 shell datapath rounds, their inverses, and mixtures of 3 rounds each. Furthermore, we evaluated that 5 core datapath rounds and 4 core datapath rounds plus 1 shell datapath rounds also achieve full degree. In contrast, 1 shell datapath round plus 4 core datapath rounds achieve a maximum algebraic degree of 22.

Arguments against integral distinguishers. An adversary does not have to select the inputs for an integral distinguisher so that they form an affine space. It is possible to pick any set and evaluate it. In [HLLT21], strong arguments against integral distinguishers are given. In essence, in the first step, for each output bit, one has to show that the superpolys associated with each high-degree term are linearly independent. So what we do to check this is to compute the evaluation of superpolys for all 24 high-degree terms for x different keys, where always at most 3 keybits are 1 and put them into a $x \times 24$ matrix. By adding the evaluation of the empty key to each superpoly, we recover a part of all monomials of key bits associated with each high-degree term. If the rank of this matrix is 24, the monomials of key bits are linearly independent.

In a next step, we again have to check if linear combinations of output bits can lead to monomials of key bits that are linearly dependent. We do this by arranging the 24 $x \times 24$ matrices into a single $x \times 24^2$ matrix, where each row represents superpoly evaluations for the same key. This matrix is required to also have full rank. We implemented the above-mentioned procedure with $x = 13943$ and verified that all resulting matrices have full rank for 6 core datapath rounds, 6 shell datapath rounds, their inverses, and mixtures of 3 rounds each. Furthermore, we evaluated that 5 core datapath rounds and 4 core datapath rounds plus 1 shell datapath round also fulfill these properties.

4.7.2 Algebraic degree of the tweak schedule

For the tweak schedule, the non-linear layer is a 53-bit permutation of degree 2. First, let us have a look at the upper bounds [BCD11] on the degree we can give assuming a 53-bit tweak. The result is given in Table 8. As we can see in Table 8, it is possible to reach full degree in 6 rounds. Since we only have a tweak of 40-bits, we assume that our tweak schedule has a sufficient number of rounds to prohibit the injection of exploitable sets into the datapath. Note that the non-linearity of the tweak schedule is not a necessity

Table 8: Upper bounds on the ANF degree after r quadratic (degree 2) round functions of 53 bits as used for the tweak schedule.

Rounds (r)	1	2	3	4	5	6
Degree (d)	2	4	8	16	32	52

in prohibiting algebraic attacks, since designs with a linear tweak schedule can provide resistance against algebraic attacks [BJK+20]. However, a non-linear tweak schedule highly complicates the attack path involving tweak relations.

4.7.3 Algebraic degree of the cipher

Considering the ciphertext and the tweak together as a modifiable input by an attacker, we are left with 64 bits of modifiable input. Assuming that both, the tweak schedule and the datapath can independently achieve the maximal degree and that in the case of the datapath, this is true for all possible linear combination of output bits, we assume that after injection of full degree monomials from the tweak schedule to the datapath, one additional round already provides enough mixture to reach a combined maximal degree.

When considering plaintext and tweak together, maximum degree tweak monomials are already inserted at the beginning of the processing of the plaintext bits. Hence, we assume that also in this case, enough datapath rounds are available to reach maximum degree in tweak and plaintext bits.

4.7.4 Interpretation of results

In this section, we provide strong arguments that 6 datapath rounds of BipBip do not allow for integral distinguishers. This means that in a single-tweak attack, it has to be possible to overcome at least 6 datapath rounds by other means, e.g., key guessing, before there is the possibility to evaluate a 5-round integral distinguisher. Even then, if those 5 datapath rounds are 5 core datapath rounds or 4 core datapath rounds plus 1 shell datapath round an integral attack is likely to fail. Hence, assuming that adding additional rounds to BipBip do not weaken its resistance against integral attacks, we conclude that BipBip withstands single-tweak integral attacks. Furthermore, we also do not see a benefit from moving to multi-tweak integral attacks. For multi-chosen-tweak integral attacks, we also do not see a promising attack angle, since both datapath and tweak schedule quickly rise in the algebraic degree, while the key guessing becomes much more involved due to the changing tweaks.

4.8 Meet-in-the-Middle Attacks

In this section, we first consider idealized rounds to get a lower-bound on the number of rounds needed for a cipher with block size of 24. Then we look at meet-in-the-middle attacks exploiting the structure of BipBip.

4.8.1 Meet-in-the-middle attack on 6 rounds assuming ideal round functions

In this section, we explore the minimum number of rounds we need for a 24-bit block cipher to be able to reach 96 bits of security. We do this by showing an attack on a 6 round 24-bit block cipher assuming independent data-round keys k^i and an ideal permutation p as round function. The sketch of the attack is shown in Figure 5. We split the cipher into two halves. For the cipher half H_1, we compute from the plaintext P to the meeting point S and for cipher half H_2, we compute from the ciphertext C to S'. The 72-bit data-round key material k^0, k^1, and k^2 are exclusively used in H_1. The 72-bit data-round key material k^4, k^5, and k^6 are exclusively used in H_2.

Attack. For the attack to work, we need 4 known plaintext-ciphertext pairs. In the first step, we encrypt the 4 plaintexts under all possible guesses for the 72-bit data-round key material up to S denoted by S_0, S_1, S_2, S_3. We store the resulting data-round key and value of S_0 in a list W sorted by values of $S_1 + S_0, S_2 + S_0, S_3 + S_0$. On average, in each index of the list, there will be one saved value. Then, we decrypt the 4 ciphertexts back to S', denoted by S'_0, S'_1, S'_2, S'_3, under all possible guesses of the other 72-bit data-round key material and we compute $S'_1 + S'_0, S'_2 + S'_0, S'_3 + S'_0$. Considering a matching between $S_1 + S_0, S_2 + S_0, S_3 + S_0$ and $S'_1 + S'_0, S'_2 + S'_0, S'_3 + S'_0$, we query for entries in the index $S_1 + S_0, S_2 + S_0, S_3 + S_0$ of the list W. This matching indicates k^4, k^5, and k^6 from current guess, k^0, k^1, and k^2 from entry of the list, and k^3 computed by $S_0 + S'_0$ is a candidate for the data-round key with high probability (about $2^{-4 \cdot 24}$). Using another 3 known plaintext-ciphertext pairs, this 168-bit candidate for the data-round key can be checked if it is the right one.

Attack complexity. From a time complexity perspective, first we did 2^{72} half encryptions and 2^{72} half decryptions, together with 2^{72} random memory accesses to find the matching. Then we need $2^{72} \cdot (1 + 2^{-24} + 2^{-48}) \approx 2^{72}$ full encryptions to check if the candidate key is the right one. In total the complexity of attack will be about 2^{73} encryptions and 2^{72} random memory accesses. The memory complexity is roughly $3 \cdot 2^{74}$ bytes and we need 7 known plaintext-ciphertext pairs.

4.8.2 Meet-in-the-middle attack on 8-round BipBip

After describing the attack above, one wonders how far we get when considering the structure of the round function. Let us have a look at an 8-round version called BipBip$_{2,5,1}$ as shown in Figure 6. As we can see in Figure 6, our linear layer allows to compute the three below equations that need six output bits from two S-boxes s and six input bits from two S-boxes of the next layer.

$$x_6 \oplus x_{19} \oplus k^4_6 \oplus k^4_{19} = x'_2 \oplus x'_{23}$$
$$x_8 \oplus x_{20} \oplus k^4_8 \oplus k^4_{20} = x'_{19} \oplus x'_4$$
$$x_{10} \oplus x_{22} \oplus k^4_{10} \oplus k^4_{22} = x'_{21} \oplus x'_1$$

In an attack, the attacker now has to guess the 24-bit data-round keys κ_0, k^1, and k^2, as well as 12 bits of k^3. For the other half of the cipher, an attacker has to guess k^8, k^7, k^6, and 12 bits of k^5. In total, we now have 171 bits of data-round key material, 84 bits on one half, 84 on the other, and 3 in the middle.

Attack. Since the matching point is only 3 bits, an attacker needs to know 57 plaintext-ciphertext pairs to uniquely determine the used key with high probability. In the first step of the attack, we perform the partial decryption of 29 ciphertexts under all 2^{78} data-round

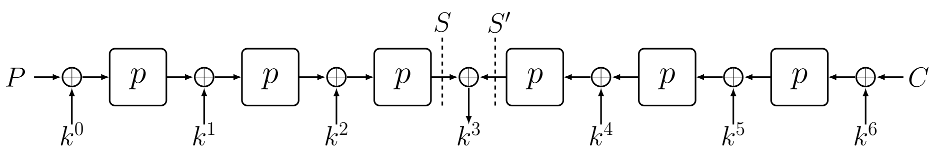

Figure 5: Attack sketch for six round cipher with ideal round function.

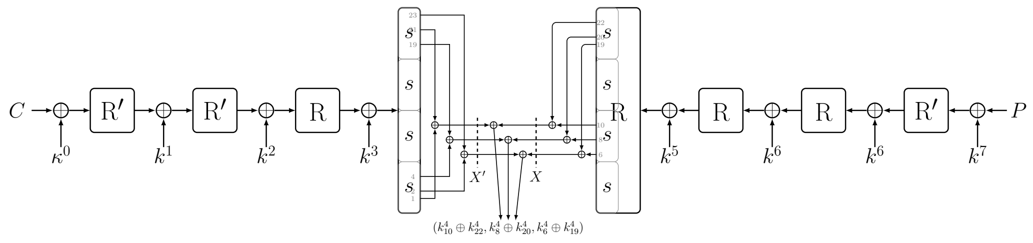
C
P
κ^0
k^1
k^2
k^3
k^5
k^6
k^6
k^7
R′
R′
R
R
R
R
R′
S
X'
X
$(k^4_{10} \oplus k^4_{22}, k^4_8 \oplus k^4_{20}, k^4_6 \oplus k^4_{19})$

Figure 6: Meet-in-the-middle attack on 8 round version of BipBip.

keys and compute the corresponding 3 bits in the middle of cipher. We denote these 3-bit values by $X_0, \ldots, X_{28}$ for each plaintext-ciphertext pair. We store the 84-bit key material together with value of X_0 in the index $X_1 + X_0, \ldots, X_{28} + X_0$ of a sorted list W.

In the second step of the attack, we partially encrypt all 29 plaintexts under all 2^{78} data-round key guesses and search for a matching in the list. That is, we partially encrypt all the 29 plaintexts until X', denoted by $X'_0, \ldots, X'_{28}$, under all possible guesses of the other 78-bit data-round key material and we compute $X'_1 + X'_0, \ldots, X'_{28} + X'_0$. Considering a matching between $X_1 + X_0, \ldots, X_{28} + X_0$ and $X'_1 + X'_0, \ldots, X'_{28} + X'_0$, we query for entries in the index $X_1 + X_0, \ldots, X_{28} + X_0$ of the list W. This matching indicates 78-bit key from current guess, 78-bit key from entry of the list, and 3-bit key computed by $X_0 + X'_0$ is a candidate for the 171-bit data-round key with high probability (about $2^{-3 \cdot 29}$). Using another 28 known plaintext-ciphertext pairs, this 171-bit candidate for the data-round key can be checked if it is the right one. Finally, the remaining $9 \cdot 24 - 171 = 45$ data-round key bits can be found by an exhaustive search.

Attack complexity. In the first step of the attack, we perform $29 \cdot 2^{78}$ half decryptions and create a list of 2^{78} 81-bits. In the second step, we perform $29 \cdot 2^{78}$ half encryptions and 2^{78} table look-ups. Then we need $2^{78} \cdot (1 + 2^{-3} + 2^{-6} + \ldots + 2^{-75}) \approx 1.14 \cdot 2^{72}$ full encryptions to check if the candidate key is the right one.

In total the complexity of attack to recover the whole 216-bit data-round key material is about $2^{82.86}$ encryptions and 2^{78} random memory accesses. The memory complexity is about $10 \cdot 2^{78}$ bytes and we need 57 known plaintext-ciphertext pairs.

It is noteworthy to mention that besides the above-mentioned meet-in-the-middle attack, there are another 3 different ones with the same complexity. Not only for BipBip$_{2,5,1}$, for any BipBip$_{x,5,y}$ with $x + y = 3$, there are four different meet-in-the-middle attacks, with the same complexity. However, still then, we do not expect an attack to cover significantly more than 8 data rounds.

4.9 Generalized Demirci-Selçuk Attacks

In this section, we discuss the application of generalized Demirci-Selçuk attacks [DS08], and their various improvements [DKS10, DFJ13, DF13]. At the heart of the attack lies a distinguishing property first described for AES [GM00], which is also applicable to BipBip. For the 9-round attack, we consider BipBip$_{2,5,2}$ as shown in Figure 7 and the function f of one active 6-bit word (marked in black) and 3 constant words in state x^1 to one word marked in black of state z^5.

4.9.1 Basic principles of the attack

Distinguishing property. If we now look at the sequence $f(0), f(1), \ldots, f(63)$, we see that the values of this sequence are fully determined by 16 6-bit words, namely the value of the 4 words of x^2 when $f(0)$ was calculated, and the 12 words of k^2, k^3, and k^4. Hence,

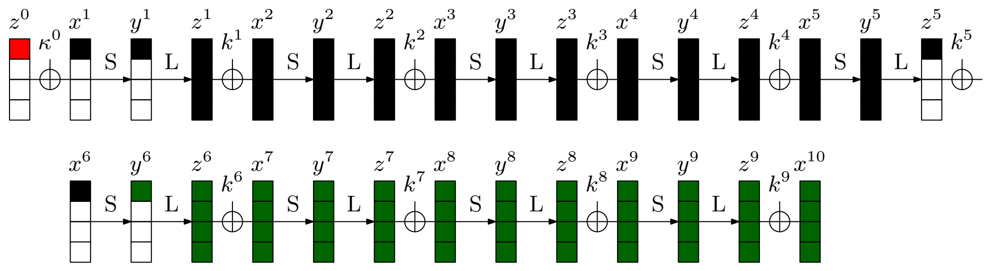

Figure 7: Generalized Demirci-Selçuk attack on 9 round version of BipBip.

we can see at most $2^{4\cdot 24} = 2^{96}$ different sequences out of all possible $(2^6)^{64}$ sequences. This is a property that can be used in a key recovery attack to distinguish wrong key guesses from the correct key guess.

We can further convert this distinguishing property to consider the differences $f(1) \oplus f(0), \ldots, f(63) \oplus f(0)$. Now, we lose a bit of information and have 2^{96} sequences out of $(2^6)^{63} = 2^{378}$ ones. However, the function f extends to state x^6 since the difference cancels k^5. An attack then looks like follows.

Precomputation. At first, we prepare the 2^{96} sequences and store them in an ideal hash table. This means we need around $2^{96} \cdot 2^6 = 2^{102}$ encryptions and at least 2^{96} memory accesses, even if the hash table is ideal. Note that with an ideal hash table, we assume the best-case scenario for an attacker. This is called the offline phase, but precomputation would be more suitable.

Attack complexity. To perform the attack, we need 2^6 ciphertexts that iterate over all values of the word marked in red in z^0 and the corresponding plaintexts. The guess of one word of κ^0 determines the order of sequence of differences, that we calculate from the ciphertexts by guessing full k^7, k^8, k^9, and one word of k^6 for all 2^6 plaintexts. The sequence of differences is contained in the precomputed table with a probability of 1 for the correct key and around $2^{96-378} = 2^{-282}$ for a wrong key guess. The complexity of this part of the attack is around $2^6 \cdot 2^{78} = 2^{84}$ partial encryptions and 2^{84} memory look-ups.

4.9.2 Refinements

As we have seen in the previous section, the complexity of the precomputation phase is higher than 2^{96} BipBip encryptions. Hence, we discuss options to lower the complexity and balance the attack so that the time complexity is smaller than 2^{96} calls to BipBip.

Refined distinguishing property. As pointed out in [DKS10], the effect that the guess of the 6-bit word of key k^1 has is to order the values of the set $f(1) \oplus f(0), f(2) \oplus f(0), \ldots, f(63) \oplus f(0)$ we use as distinguishing property. Hence, we can just not guess this word of key k^1 and use the unordered multiset of differences as distinguishing property. However, we still get at most 2^{96} multisets from z^0 to x^6, but having only 6-bit words, we only have about $\binom{64+63-1}{63} \approx 2^{122}$ different multisets in total. Hence, the probability that a wrong key guess hits one of the multisets contained in the hash table is non-negligible (about $2^{96-102} = 2^{-6}$). A simple way to alleviate this is to use the multiset of 12-bits from x^6 as distinguishing property. Then, we have $\binom{4096+63-1}{63} \approx 2^{466}$ different possible multisets that makes the probability that a wrong key guess hits one of the multisets contained in the hash table to be about $2^{96-466} = 2^{-370}$.

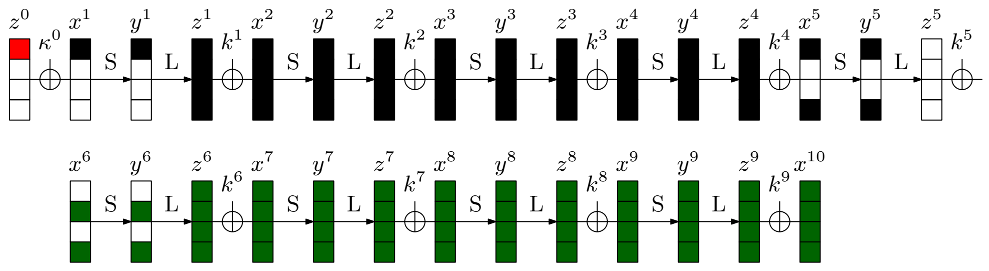

Figure 8: Generalized Demirci-Selçuk attack on 9 round version of BipBip.

Shorter precomputation. Instead of preparing 2^{96} sequences, we just prepare 2^{90} and store them in an ideal hash table. This means we need around $2^{90} \cdot 2^6 = 2^{96}$ encryptions and at least 2^{90} memory accesses, even if the hash table is ideal.

Attack. Due to the different distinguishing property, we do not have to guess κ^0 anymore. However, still we need 2^6 ciphertexts that iterate over one word in z^0, while being constant in all other words. For all corresponding 2^6 plaintexts, we guess full k^7, k^8, k^9, and 12 bits of k^6 and check if the multiset of 63 differences is contained in our table. Since we do not store all possible sequences, there is a chance of roughly 2^{-6} that the sequence produced by the correct key guess is not in our list. That means for the correct guess, it is contained with probability 2^{-6} and for a wrong guess with probability $2^{90-466} = 2^{-376}$. Hence, if we redo the attack on average 2^6 times, we only recover the correct partial key with high probability.

Attack complexity. For the attack, we have a data complexity of $2^6 \cdot 2^6 = 2^{12}$ plaintext ciphertext pairs. In total, we do $2^{12} \cdot 2^{84} = 2^{96}$ partial calls to BipBip plus at least $2^6 \cdot 2^{84} = 2^{90}$ accesses to the ideal hash table.

With the help of an automated search tool [DF16], we were able to improve the attack complexities for $\text{BipBip}_{2,5,2}$ based on the truncated pattern shown in Figure 8.

This is due to the property of the linear layer of the core round function R. We can get knowledge on three bits of the input of the two S-boxes marked in green of state x^6 by combining the information of 6 bits of two S-box outputs of state y^5.

$$x^6_6 \oplus x^6_{19} \oplus k^5_6 \oplus k^5_{19} = y^5_2 \oplus y^5_{23}$$
$$x^6_8 \oplus x^6_{20} \oplus k^5_8 \oplus k^5_{20} = y^5_{19} \oplus y^5_4$$
$$x^6_{10} \oplus x^6_{22} \oplus k^5_{10} \oplus k^5_{22} = y^5_{21} \oplus y^5_1$$

Distinguishing property. This influences now the distinguishing property for the sequence $f(0), f(1), \ldots, f(63)$. This sequence now consist of 3-bit chunks and is fully determined by 14 6-bit words, namely the value of the 4 words of x^2 when $f(0)$ was calculated, the 8 words of k^2 and k^3, and 2 words of k^4. Hence, we can see at most $2^{6 \cdot 14} = 2^{84}$ different sequences out of all possible $(2^3)^{64} = 2^{192}$ sequences. This is a property that can be used as a distinguisher.

Attack complexity. To perform the attack, we need 2^6 ciphertexts that iterate over all values of the word marked in red in z^0 and the corresponding plaintexts. The guess of one word of κ^0 determines the order of sequence of differences, that we calculate from the ciphertexts by guessing full k^7, k^8, k^9, and two words of k^6 for all 2^6 plaintexts. The sequence of differences is contained in the precomputed table with a probability of 1 for the correct key and around $2^{84-192} = 2^{-108}$ for a wrong key guess. The complexity of this attack is around $2^6 \cdot 2^{84} = 2^{90}$ encryptions and 2^{90} memory look-ups.

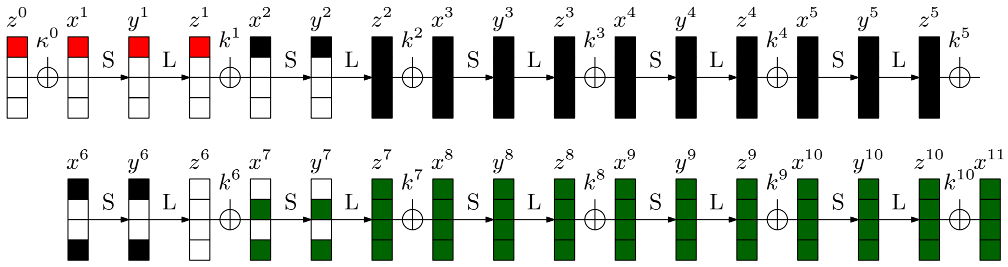

Figure 9: Generalized Demirci-Selçuk attack on 10 round version of BipBip.

4.9.3 Extending to 10 Rounds

We also applied the automated search tool [DF16] on $\text{BipBip}_{2,5,3}$, suggesting the pattern shown in Figure 9 for an attack. It is basically an extension of Figure 8 by one round at the front. This is possible by not considering the iteration over all 6 bits of one word in x^2 but rather we iterate over 2 bits. Now, the 3-bit output is defined by 15 words, so we see at most $2^{6\cdot 15} = 2^{90}$ different sequences of 2^2 3-bit values. However, there are only $(2^3)^4 = 2^{12}$ different ones. So, it seems that this property does not lead to a key recovery attack.

The other way to extend the attack of Figure 8 by one round is to add one round in the middle to the precomputation phase as shown in Figure 10. Here, we see that the function from one word in x^1 to 3 bits in z^6 is determined by 18 words. However, as argued by [DFJ13], if the sequence of function values contains one pair of values that follows the differential in black, we know that the 18 words can only take $2^{15\cdot 6}$ different values. Hence, table size and precomputation reduce to $2^6 \cdot 2^{90}$ partial encryptions and 2^{90} memory accesses. However, to be sure to have at least one pair that follows the truncated differential, we need to have around 2^{12} plaintext ciphertext pairs. This leads to $2^{12} \cdot 2^{90}$ partial decryptions in addition to the 2^{90} memory look-ups.

4.9.4 Verdict

In this section, we explored the application of generalized Demirci-Selçuk attacks. We are able to give attacks on 9 rounds of BipBip. Furthermore, we have investigated approaches given suggested by an automated search tool [DF16] for 10 rounds of BipBip. For 10 rounds of BipBip, we were not able to give attacks below a time complexity of 2^{96} calls to BipBip. Potentially, those attacks can be further improved by, e.g., considering details of the S-box. However, considering that all of these attacks have a very high memory complexity, it seems unlikely that a practical threat to the security of 11 rounds of BipBip emerges from these types of attacks.

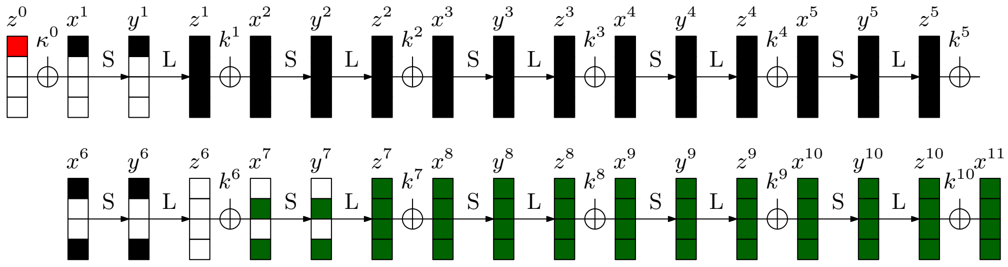

Figure 10: Generalized Demirci-Selçuk attack on 10 round version of BipBip.

4.10 Invariant Attacks, Cycles, and Symmetries

Invariant attacks [LAAZ11, TLS16] exploit the knowledge that if the input to a (keyed) permutation belongs to a certain input set, it will map to a certain output set. With cycle we denote the event that after r applications of a (keyed) round function, an input X maps to X for all r contained in the cycle. With symmetries we denote a possible characteristic of a (keyed) permutation that an input X and a simple bit-permutation of X lead to an output Y and a simple permutation of Y.

All these above-mentioned characteristics have in common that besides a careful selection of linear and non-linear components, typically the round constant plays an important role in the prevention of the mentioned characteristics [BCLR17]. However, notably, we do not use a round constant. The reason for that is rather simple, the purpose of round constants is to differentiate single rounds although the used round key (for a weak-key portion of all keys) stays the same. However, we use a masterkey of 256 bits, while only having data-round keys of 24 bits aiming at 96-bit security. Formulated differently, we come even very close to having independent tweak-round keys in the tweak schedule. Therefore, we do not expect any security benefits from using round constants, since they do not have a big influence on the number of weak keys in our case. Summing up, we assume that the use of randomly chosen 256-bit data-round keys together with the non-linear tweak schedule prohibits invariant attacks and attacks exploiting cycles, or symmetries.

5 Implementation

To assess the goal of designing a low-latency tweakable block cipher, we have to implement BipBip. The first implementation we will discuss is a fully unrolled implementation to evaluate the input to output latency without any intermediate register stages. Therefore, we unrolled all data-block rounds as well as tweak rounds of BipBip as depicted at a high-level in Figure 11. At the input, we have registered the input data, key and tweak. After that, all datapaths are put through back-to-back combinatorial logic. Finally, the output of the last combinatorial logic is registered. So, all combinatorial datapaths of the entire cipher are placed in between the input register and the output register.

Except for the BipBipBox and the non-linear χ function, all other operations of BipBip are linear and hence, rather easy to optimize in implementations. The χ function has already a very compact definition at gate level with three inputs as $\chi : a_i \leftarrow a_i$ XOR (NOT(a_{i+1}) AND a_{i+2}), which limits potential further optimization opportunities. Therefore, to achieve an optimized implementation of the BipBip datapath, we first explored different design options for the BipBipBox. First, we have implemented an optimized NAND-based design, which uses four 2-input NAND gates with 25.9 ps critical path as depicted in Figure 12. Note that all four NAND gates may not have the same delay, since in standard cell technology, the output delay of a gate varies significantly depending on its input transitions and output loads. As a second option, we implemented BipBipBox as a table look-up using case statements in Verilog Hardware Description Language (HDL). The Synopsys Design Compiler optimized this implementation with 3 logic levels (1 OAI, 1 NOR and 1 XOR) resulting in only a 1.9 ps longer critical path. All

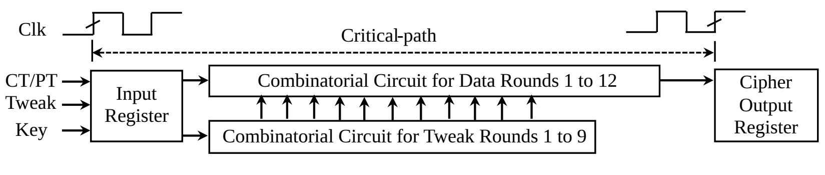
Clk
Critical-path
CT/PT
Tweak
Key
Input Register
Combinatorial Circuit for Data Rounds 1 to 12
Combinatorial Circuit for Tweak Rounds 1 to 9
Cipher Output Register

Figure 11: Unrolled micro-architecture for BipBip that estimates input to output latency.

Table 9: Unrolled implementation results on Intel 10nm with 0.85 V and 100°C.

Cipher	**Critical Path**		**Area**	**Power**
	Gate Levels	Delay [ps]	[GE]	[mW]
Prince Enc/Dec	74	853	7542	42.71
BipBip Dec	**48**	**622**	**5741**	15.91
BipBip Enc	148	1523	10776	19.23

the hardware design experiments reported in this paper are performed on Intel 10nm FinFet technology [AAA+17] with a 0.85 V target voltage and 100°C temperature while we put timing constraints in the synthesis tool. Experiments for the S-box implementations started with 100 ps target clock period and were reduced iteratively to capture the minimum clock period for which the implementation is still capable to meet the timing constraints.

As a next step, we explored the effect of these two S-box implementations on the entire unrolled BipBip datapath (Figure 11). We implemented two instances of the unrolled datapath which differ only in the S-box implementation. Both instances were synthesized multiple times with different target clock periods starting from 1000 ps. Interestingly, both NAND and case-statement based implementations converged to the same timing with 622 ps as lowest result, while they differ in the number of gate levels in the critical path with 54 and 48, respectively. Note that the critical path of our unrolled design is measured in between two rising edges of the clock, which includes the delay of the input register and the delay of the entire combinatorial circuit to implement the whole tweakable block cipher. This experiment ensured that the Synopsys Design Compiler is capable of finding the optimal design from a table look-up implementation in HDL. Moving forward, we implemented an unrolled design of BipBip's encryption with a table look-up based inverse S-box implementation. Similarly, to compare the performance of BipBip against state-of-the-art low-latency block ciphers, we implement Prince [BCG+12] with the same implementation principles on the same technology. Table 9 reports the comparative implementation results.

As we can see, BipBip's encryption datapath provides 27% smaller critical path delay compared to the 64-bit block cipher Prince which is known as one of the lowest-latency block ciphers in the literature. It also requires 24% lower area compared to Prince. Note that both BipBip and Prince process 64 bits and that is why comparing the area and energy costs are reasonable. The implementation of BipBip's encryption computes the unrolled tweak schedule followed by the inverse datapath for the 11 inverse rounds. Our results show that the implementation of BipBip's encryption is 2.45 times slower and requires 1.88 times more logic gates compared to the decryption.

The major use-case for BipBip lies in pipelined implementations. In particular, it shall be possible to compute the pointer decryption for C^3 in three cycles @4GHz on Intel 10nm technology. To achieve this, we break the BipBip unrolled decryption circuit into three parts and incorporate two registers to achieve a depth-3 pipeline. In this implementation, we place rounds 1 to 4 of the datapath and the corresponding rounds 1 to 4 of the tweak

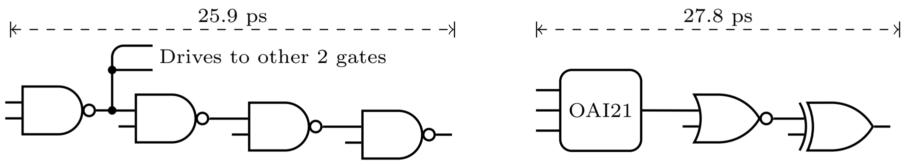

Figure 12: Critical path of the BipBipBox implementation in Verilog-HDL: (left) with optimized NAND logic and (right) with case statement of the substitution function.

schedule within the first stage of the pipeline. The second stage of the pipeline consists of rounds 5 to 7 of the datapath as well as the corresponding tweak schedule. The third stage computes the rest of the cipher. This depth-3 pipelined implementation met all timing constraints with a 223 ps critical path, which ensures that BipBip's decryption can be performed within 3 cycles even up to 4.5 GHz clock frequency with 0.85 V and 100°C operating condition.

Regarding the power consumption shown in Table 9, we see that Prince needs 42.71 mW while BipBip requires 15.91 mW. Focusing on the decrypted bits, we get 0.667 mW per bit for Prince and 0.662 mW per bit for BipBip. However, we want to mention that in addition to the 24 bits of ciphertext, BipBip also processes 40 bits of tweak, which would turn the metric mW per processed bit of data even more in favor of BipBip with 0.249 mW per processed bit. For getting the corresponding energy and energy per bit numbers, the power can be multiplied with the critical path delay. The power consumption is estimated by the Synopsys tool while we compiled the RTL for Intel 10nm technology, operating voltage 0.85V, Max operating Temp 100C and target operating clock 4GHz.

The reason why we mainly focus on the decryption implementation in this section is because decryption is called whenever a pointer is dereferenced. This happens often in a typical program and hence, the latency of the decryption is very performance critical. In the case of multi-core systems that can perform out-of-order execution, many execution ports can do pointer dereferencing to load data from memory and hence, BipBip must be implemented in multiple places. Therefore, the area is also critical for decryption.

On the other hand, encryption is only called when new memory is allocated. This happens comparatively infrequently and adding a few cycles more does not impact performance much. So, it could be implemented as, e.g., instruction set extension, or even in software, since the performance impact where it is used is comparatively small.

6 Conclusion

In this paper, we investigated how to design a secure ultra-low-latency tweakable block cipher with a small block size of 24 bits. The result of this investigation is BipBip that allows for decryption within 3 cycles for up to 4.5 GHz clock frequency on modern CMOS technologies. To support its security claim, we provide extensive preliminary cryptanalysis.

The design of BipBip was a very challenging task, since results for low-latency tweakable block cipher with such a very small domain are scarce. What adds to the challenge is that even the research area of format preserving encryption, which deals with such small dimensions but uses non-low-latency round functions up to the full AES for a single round, deals with many recent breaks of proposed schemes [BHT16, DV17, HTT18, DKLS20, ADK$^+$21, Bey21]. Nevertheless, we think it is the time to take on this challenge, since with C^3 [LRD$^+$21], there is a convincing application that demands for small domain low-latency tweakable block cipher. We hope that BipBip together with C^3 triggers much more research on the design of small domain low-latency primitives, cryptanalysis exploiting such small dimensions, and applications that can benefit from using such primitives.

Acknowledgments This work was partially funded by Intel through the Crypto Frontiers Research Center. Joan Daemen is supported by the European Research Council under the ERC advanced grant agreement under grant ERC-2017-ADG Nr. 788980 ESCADA, and Shahram Rasoolzadeh is supported by the Netherlands Organisation for Scientific Research (NWO) under TOP grant TOP1.18.002 SCALAR.

References

[AAA+17] C. Auth, A. Aliyarukunju, M. Asoro, D. Bergstrom, V. Bhagwat, J. Birdsall, N. Bisnik, M. Buehler, V. Chikarmane, G. Ding, Q. Fu, H. Gomez, W. Han, D. Hanken, M. Haran, M. Hattendorf, R. Heussner, H. Hiramatsu, B. Ho, S. Jaloviar, I. Jin, S. Joshi, S. Kirby, S. Kosaraju, H. Kothari, G. Leatherman, K. Lee, J. Leib, A. Madhavan, K. Marla, H. Meyer, T. Mule, C. Parker, S. Parthasarathy, C. Pelto, L. Pipes, I. Post, M. Prince, A. Rahman, S. Rajamani, A. Saha, J. Dacuna Santos, M. Sharma, V. Sharma, J. Shin, P. Sinha, P. Smith, M. Sprinkle, A. St. Amour, C. Staus, R. Suri, D. Towner, A. Tripathi, A. Tura, C. Ward, and A. Yeoh. A 10nm high performance and low-power CMOS technology featuring 3rd generation FinFET transistors, Self-Aligned Quad Patterning, contact over active gate and cobalt local interconnects. In *2017 IEEE International Electron Devices Meeting (IEDM)*, pages 29.1.1–29.1.4, 2017.

[ADK+21] Ohad Amon, Orr Dunkelman, Nathan Keller, Eyal Ronen, and Adi Shamir. Three Third Generation Attacks on the Format Preserving Encryption Scheme FF3. In Anne Canteaut and François-Xavier Standaert, editors, *EUROCRYPT 2021*, volume 12697 of *LNCS*, pages 127–154. Springer, 2021.

[AJN14] Jean-Philippe Aumasson, Philipp Jovanovic, and Samuel Neves. NORX: Parallel and Scalable AEAD. In Miroslaw Kutylowski and Jaideep Vaidya, editors, *ESORICS 2014*, volume 8713 of *LNCS*, pages 19–36. Springer, 2014.

[AL12] Martin R. Albrecht and Gregor Leander. An All-In-One Approach to Differential Cryptanalysis for Small Block Ciphers. In Lars R. Knudsen and Huapeng Wu, editors, *SAC 2012*, volume 7707 of *LNCS*, pages 1–15. Springer, 2012.

[ARS+15] Martin R. Albrecht, Christian Rechberger, Thomas Schneider, Tyge Tiessen, and Michael Zohner. Ciphers for MPC and FHE. In Elisabeth Oswald and Marc Fischlin, editors, *EUROCRYPT 2015*, volume 9056 of *LNCS*, pages 430–454. Springer, 2015.

[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.

[BCD+99] Carolynn Burwick, Don Coppersmith, Edward D'Avignon, Rosario Gennaro, Shai Halevi, Charanjit Jutla, Stephen M. Matyas Jr., Luke O'Connor, Mohammad Peyravian, David Safford, and Nevenko Zunic. MARS - a candidate cipher for AES. Submission to NIST AES competition, 1999.

[BCD11] Christina Boura, Anne Canteaut, and Christophe De Cannière. Higher-Order Differential Properties of Keccak and *Luffa*. In Antoine Joux, editor, *FSE 2011*, volume 6733 of *LNCS*, pages 252–269. Springer, 2011.

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

[BCLR17] Christof Beierle, Anne Canteaut, Gregor Leander, and Yann Rotella. Proving Resistance Against Invariant Attacks: How to Choose the Round Constants. In Jonathan Katz and Hovav Shacham, editors, *CRYPTO 2017*, volume 10402 of *LNCS*, pages 647–678. Springer, 2017.

[BDMW10] K. A. Browning, J. F. Dillon, M. T. McQuistan, and A. J. Wolfe. An APN Permutation in Dimension Six. *Finite Fields: theory and applications*, 518:33–42, 2010.

[BDPV11a] Guido Bertoni, Joan Daemen, Michaël Peeters, and Gilles Van Assche. Duplexing the Sponge: Single-Pass Authenticated Encryption and Other Applications. In Ali Miri and Serge Vaudenay, editors, *SAC 2011*, volume 7118 of *LNCS*, pages 320–337. Springer, 2011.

[BDPV11b] Guido Bertoni, Joan Daemen, Michaël Peeters, and Gilles Van Assche. Cryptographic sponge functions. https://keccak.team, 2011.

[BDPV11c] Guido Bertoni, Joan Daemen, Michaël Peeters, and Gilles Van Assche. The Keccak reference. SHA-3 competition (round 3), 2011. https://keccak.team/papers.html.

[Ber08] Daniel J. Bernstein. ChaCha, a variant of Salsa20. https://cr.yp.to/chacha.html, 2008.

[Bey21] Tim Beyne. Linear Cryptanalysis of FF3-1 and FEA. In Tal Malkin and Chris Peikert, editors, *CRYPTO 2021*, volume 12825 of *LNCS*, pages 41–69. Springer, 2021.

[BHT16] Mihir Bellare, Viet Tung Hoang, and Stefano Tessaro. Message-Recovery Attacks on Feistel-Based Format Preserving Encryption. In Edgar R. Weippl, Stefan Katzenbeisser, Christopher Kruegel, Andrew C. Myers, and Shai Halevi, editors, *ACM CCS 2016*, pages 444–455. ACM, 2016.

[BJK$^+$16] Christof Beierle, Jérémy Jean, Stefan Kölbl, Gregor Leander, Amir Moradi, Thomas Peyrin, Yu Sasaki, Pascal Sasdrich, and Siang Meng Sim. The SKINNY Family of Block Ciphers and Its Low-Latency Variant MANTIS. In Matthew Robshaw and Jonathan Katz, editors, *CRYPTO 2016*, volume 9815 of *LNCS*, pages 123–153. Springer, 2016.

[BJK$^+$20] Christof Beierle, Jérémy Jean, Stefan Kölbl, Gregor Leander, Amir Moradi, Thomas Peyrin, Yu Sasaki, Pascal Sasdrich, and Siang Meng Sim. SKINNY-AEAD and SKINNY-Hash. *IACR Transactions on Symmetric Cryptology*, 2020(S1):88–131, 2020.

[BR10] Andrey Bogdanov and Christian Rechberger. A 3-Subset Meet-in-the-Middle Attack: Cryptanalysis of the Lightweight Block Cipher KTANTAN. In Alex Biryukov, Guang Gong, and Douglas R. Stinson, editors, *SAC 2010*, volume 6544 of *LNCS*, pages 229–240. Springer, 2010.

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

[CIL+20] Wonseok Choi, Akiko Inoue, ByeongHak Lee, Jooyoung Lee, Eik List, Kazuhiko Minematsu, and Yusuke Naito. Highly secure nonce-based macs from the sum of tweakable block ciphers. *IACR Trans. Symmetric Cryptol.*, 2020(4):39–70, 2020.

[CJRR99] Suresh Chari, Charanjit S. Jutla, Josyula R. Rao, and Pankaj Rohatgi. Towards Sound Approaches to Counteract Power-Analysis Attacks. In Michael J. Wiener, editor, *CRYPTO '99*, volume 1666 of *LNCS*, pages 398–412. Springer, 1999.

[DDK09] Christophe De Cannière, Orr Dunkelman, and Miroslav Knezevic. KATAN and KTANTAN - A Family of Small and Efficient Hardware-Oriented Block Ciphers. In Christophe Clavier and Kris Gaj, editors, *CHES 2009*, volume 5747 of *LNCS*, pages 272–288. Springer, 2009.

[DEG+18] Christoph Dobraunig, Maria Eichlseder, Lorenzo Grassi, Virginie Lallemand, Gregor Leander, Eik List, Florian Mendel, and Christian Rechberger. Rasta: A Cipher with Low AND depth and Few ANDs per Bit. In Hovav Shacham and Alexandra Boldyreva, editors, *CRYPTO 2018*, volume 10991 of *LNCS*, pages 662–692. Springer, 2018.

[DEMS21] Christoph Dobraunig, Maria Eichlseder, Florian Mendel, and Martin Schläffer. Ascon v1.2: Lightweight Authenticated Encryption and Hashing. *J. Cryptol.*, 34(3):33, 2021.

[DF13] Patrick Derbez and Pierre-Alain Fouque. Exhausting Demirci-Selçuk Meet-in-the-Middle Attacks Against Reduced-Round AES. In Shiho Moriai, editor, *Fast Software Encryption - 20th International Workshop, FSE 2013, Singapore, March 11-13, 2013. Revised Selected Papers*, volume 8424 of *Lecture Notes in Computer Science*, pages 541–560. Springer, 2013.

[DF16] Patrick Derbez and Pierre-Alain Fouque. Automatic Search of Meet-in-the-Middle and Impossible Differential Attacks. In Matthew Robshaw and Jonathan Katz, editors, *Advances in Cryptology - CRYPTO 2016 - 36th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 14-18, 2016, Proceedings, Part II*, volume 9815 of *Lecture Notes in Computer Science*, pages 157–184. Springer, 2016.

[DFJ13] Patrick Derbez, Pierre-Alain Fouque, and Jérémy Jean. Improved Key Recovery Attacks on Reduced-Round AES in the Single-Key Setting. In Thomas Johansson and Phong Q. Nguyen, editors, *Advances in Cryptology - EUROCRYPT 2013, 32nd Annual International Conference on the Theory and Applications of Cryptographic Techniques, Athens, Greece, May 26-30, 2013. Proceedings*, volume 7881 of *Lecture Notes in Computer Science*, pages 371–387. Springer, 2013.

[DHP+20] Joan Daemen, Seth Hoffert, Michaël Peeters, Gilles Van Assche, and Ronny Van Keer. Xoodyak, a lightweight cryptographic scheme. *IACR Transactions on Symmetric Cryptology*, 2020(S1):60–87, 2020.

[Din21] Itai Dinur. Cryptanalytic Applications of the Polynomial Method for Solving Multivariate Equation Systems over GF(2). In Anne Canteaut and François-Xavier Standaert, editors, *EUROCRYPT 2021*, volume 12696 of *LNCS*, pages 374–403. Springer, 2021.

[DKLS20] Orr Dunkelman, Abhishek Kumar, Eran Lambooij, and Somitra Kumar Sanadhya. Cryptanalysis of Feistel-Based Format-Preserving Encryption. Cryptology ePrint Archive, Report 2020/1311, 2020. https://ia.cr/2020/1311.

[DKS10] Orr Dunkelman, Nathan Keller, and Adi Shamir. Improved Single-Key Attacks on 8-Round AES-192 and AES-256. In Masayuki Abe, editor, *Advances in Cryptology - ASIACRYPT 2010 - 16th International Conference on the Theory and Application of Cryptology and Information Security, Singapore, December 5-9, 2010. Proceedings*, volume 6477 of *Lecture Notes in Computer Science*, pages 158–176. Springer, 2010.

[DMMR20] Joan Daemen, Pedro Maat Costa Massolino, Alireza Mehrdad, and Yann Rotella. The Subterranean 2.0 Cipher Suite. *IACR Transactions on Symmetric Cryptology*, 2020(S1):262–294, 2020.

[DR07] Joan Daemen and Vincent Rijmen. Probability Distributions of Correlation and Differentials in Block Ciphers. *J. Math. Cryptol.*, 1(3):221–242, 2007.

[DR20] Joan Daemen and Vincent Rijmen. *The Design of Rijndael - The Advanced Encryption Standard (AES), Second Edition*. Information Security and Cryptography. Springer, 2020.

[DS08] Hüseyin Demirci and Ali Aydin Selçuk. A Meet-in-the-Middle Attack on 8-Round AES. In Kaisa Nyberg, editor, *Fast Software Encryption, 15th International Workshop, FSE 2008, Lausanne, Switzerland, February 10-13, 2008, Revised Selected Papers*, volume 5086 of *Lecture Notes in Computer Science*, pages 116–126. Springer, 2008.

[DS09] Itai Dinur and Adi Shamir. Cube Attacks on Tweakable Black Box Polynomials. In Antoine Joux, editor, *EUROCRYPT 2009*, volume 5479 of *LNCS*, pages 278–299. Springer, 2009.

[DV17] F. Betül Durak and Serge Vaudenay. Breaking the FF3 Format-Preserving Encryption Standard over Small Domains. In Jonathan Katz and Hovav Shacham, editors, *CRYPTO 2017*, volume 10402 of *LNCS*, pages 679–707. Springer, 2017.

[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.

[GM00] Henri Gilbert and Marine Minier. A Collision Attack on 7 Rounds of Rijndael. In *The Third Advanced Encryption Standard Candidate Conference, April 13-14, 2000, New York, New York, USA*, pages 230–241. National Institute of Standards and Technology,, 2000.

[GP99] Louis Goubin and Jacques Patarin. DES and Differential Power Analysis (The "Duplication" Method). In Çetin Kaya Koç and Christof Paar, editors, *CHES'99*, volume 1717 of *LNCS*, pages 158–172. Springer, 1999.

[HLLT20] Phil Hebborn, Baptiste Lambin, Gregor Leander, and Yosuke Todo. Lower Bounds on the Degree of Block Ciphers. In Shiho Moriai and Huaxiong Wang, editors, *ASIACRYPT 2020*, volume 12491 of *LNCS*, pages 537–566. Springer, 2020.

[HLLT21] Phil Hebborn, Baptiste Lambin, Gregor Leander, and Yosuke Todo. Strong and Tight Security Guarantees Against Integral Distinguishers. In Mehdi Tibouchi and Huaxiong Wang, editors, *ASIACRYPT 2021*, volume 13090 of *LNCS*, pages 362–391. Springer, 2021.

[HTT18] Viet Tung Hoang, Stefano Tessaro, and Ni Trieu. The Curse of Small Domains: New Attacks on Format-Preserving Encryption. In Hovav Shacham and Alexandra Boldyreva, editors, *CRYPTO 2018*, volume 10991 of *LNCS*, pages 221–251. Springer, 2018.

[JNP14] Jérémy Jean, Ivica Nikolic, and Thomas Peyrin. Tweaks and Keys for Block Ciphers: The TWEAKEY Framework. In Palash Sarkar and Tetsu Iwata, editors, *ASIACRYPT 2014*, volume 8874 of *LNCS*, pages 274–288. Springer, 2014.

[KLPR10] Lars R. Knudsen, Gregor Leander, Axel Poschmann, and Matthew J. B. Robshaw. PRINTcipher: A Block Cipher for IC-Printing. In Stefan Mangard and François-Xavier Standaert, editors, *CHES 2010*, volume 6225 of *LNCS*, pages 16–32. Springer, 2010.

[KR96] Joe Kilian and Phillip Rogaway. How to Protect DES Against Exhaustive Key Search. In Neal Koblitz, editor, *CRYPTO '96*, volume 1109 of *LNCS*, pages 252–267. Springer, 1996.

[KW02] Lars R. Knudsen and David A. Wagner. Integral Cryptanalysis. In Joan Daemen and Vincent Rijmen, editors, *FSE 2002*, volume 2365 of *LNCS*, pages 112–127. Springer, 2002.

[LAAZ11] Gregor Leander, Mohamed Ahmed Abdelraheem, Hoda AlKhzaimi, and Erik Zenner. A Cryptanalysis of PRINTcipher: The Invariant Subspace Attack. In Phillip Rogaway, editor, *CRYPTO 2011*, volume 6841 of *LNCS*, pages 206–221. Springer, 2011.

[Lai94] Xuejia Lai. *Higher Order Derivatives and Differential Cryptanalysis*, pages 227–233. Springer US, Boston, MA, 1994.

[Lev84] Henry M Levy. *Capability-based Computer Systems*. Digital Press, 1984.

[LH94] Susan K. Langford and Martin E. Hellman. Differential-Linear Cryptanalysis. In Yvo Desmedt, editor, *CRYPTO '94*, volume 839 of *LNCS*, pages 17–25. Springer, 1994.

[LMM91] Xuejia Lai, James L. Massey, and Sean Murphy. Markov Ciphers and Differential Cryptanalysis. In Donald W. Davies, editor, *EUROCRYPT '91*, volume 547 of *LNCS*, pages 17–38. Springer, 1991.

[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.

[LRD$^+$21] Michael LeMay, Joydeep Rakshit, Sergej Deutsch, David M. Durham, Santosh Ghosh, Anant Nori, Jayesh Gaur, Andrew Weiler, Salmin Sultana, Karanvir Grewal, and Sreenivas Subramoney. Cryptographic Capability Computing. In *MICRO '21*, pages 253–267. ACM, 2021.

[Mat93] Mitsuru Matsui. Linear Cryptanalysis Method for DES Cipher. In Tor Helleseth, editor, *EUROCRYPT '93*, volume 765 of *LNCS*, pages 386–397. Springer, 1993.

[MDV17] Silvia Mella, Joan Daemen, and Gilles Van Assche. New Techniques for Trail Bounds and Application to Differential Trails in Keccak. *IACR Transactions on Symmetric Cryptology*, 2017(1):329–357, 2017.

[Nai15] Yusuke Naito. Full prf-secure message authentication code based on tweakable block cipher. In Man Ho Au and Atsuko Miyaji, editors, *Provable Security - 9th International Conference, ProvSec 2015, Kanazawa, Japan, November 24-26, 2015, Proceedings*, volume 9451 of *Lecture Notes in Computer Science*, pages 167–182. Springer, 2015.

[O'C93] Luke O'Connor. On the Distribution of Characteristics in Bijective Mappings. In Tor Helleseth, editor, *EUROCRYPT '93*, volume 765 of *LNCS*, pages 360–370. Springer, 1993.

[Ras22] Shahram Rasoolzadeh. Low-latency boolean functions and bijective s-boxes. *IACR Transactions on Symmetric Cryptology*, 2022(3):403–447, Sep. 2022.

[TLS16] Yosuke Todo, Gregor Leander, and Yu Sasaki. Nonlinear Invariant Attack - Practical Attack on Full SCREAM, iSCREAM, and Midori64. In Jung Hee Cheon and Tsuyoshi Takagi, editors, *ASIACRYPT 2016*, volume 10032 of *LNCS*, pages 3–33, 2016.

[Tod15] Yosuke Todo. Structural Evaluation by Generalized Integral Property. In Elisabeth Oswald and Marc Fischlin, editors, *EUROCRYPT 2015*, volume 9056 of *LNCS*, pages 287–314. Springer, 2015.

[Wag99] David A. Wagner. The Boomerang Attack. In Lars R. Knudsen, editor, *FSE '99*, volume 1636 of *LNCS*, pages 156–170. Springer, 1999.

[WWC+14] Jonathan Woodruff, Robert N. M. Watson, David Chisnall, Simon W. Moore, Jonathan Anderson, Brooks Davis, Ben Laurie, Peter G. Neumann, Robert M. Norton, and Michael Roe. The CHERI Capability Model: Revisiting RISC in an age of risk. In *ACM/IEEE ISCA 2014*, pages 457–468. IEEE Computer Society, 2014.

Results for differential distribution and linear correlation

Table 10: Experimental and theoretical distributions of $\mathrm{N}(a,b)$, number of pairs for differential $(a,b) = (\texttt{0x00002e}, \texttt{0x220020})$ for 14×2^{12} randomly chosen key samples. The experimental line contains the histogram of the number of keys for which the differential (a,b) has N pairs and the theoretical line the expected number of keys in the case of a Poisson distribution with $\lambda = 2^{23}\mathrm{EDP} = 0.516122$.

N	0	1	2	3	4	5	6
Experimental	34100	17845	4551	755	83	9	1
Theoretical	34224.64	17664.10	4558.42	784.23	101.19	10.45	0.90

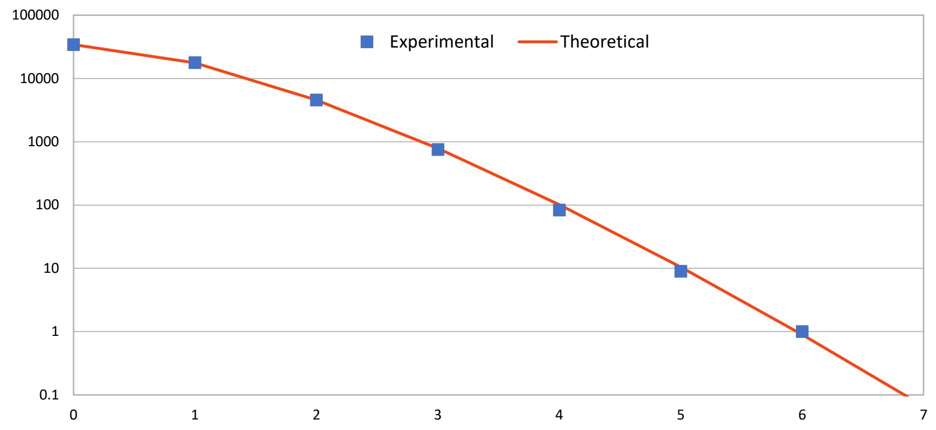

Figure 13: Experimental and theoretical distributions of $\mathrm{N}(a, b)$, corresponding to Table 10.

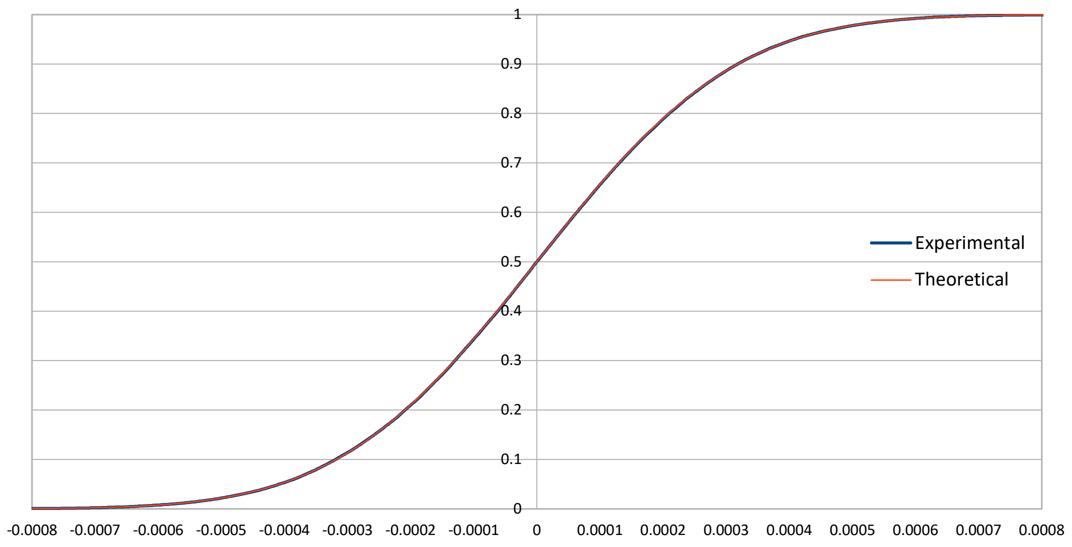

Figure 14: Experimental and theoretical cumulative distributions for $\mathrm{C}(a, b)$, with $(a, b) =$ (`0x0008c0`, `0x1a0974`) using 40×2^{12} randomly chosen key samples. The experimental line contains the cumulative ratio of the keys for which the linear approximation (a, b) has value of x and the theoretical line the expected cumulative ratio of keys in the case of a Normal distribution with $\mu = 0$ and $\sigma^2 = \mathrm{ELP} = 1.034764 \times 2^{-24}$.