

uKNIT: Breaking Round-alignment for Cipher Design

Featuring uKNIT-BC, an Ultra Low-Latency Block Cipher

Kai Hu^{1,4}, Mustafa Khairallah², Thomas Peyrin³ and Quan Quan Tan³

¹ School of Cyber Science and Technology, Shandong University, Qingdao, Shandong, China

`kai.hu@sdu.edu.cn`

² Dept. of Electrical and Information Technology, Lund University, Lund, Sweden

`mustafa.khairallah.1608@eit.lth.se`

³ Nanyang Technological University, Singapore

`thomas.peyrin@ntu.edu.sg, quanquan001@e.ntu.edu.sg`

⁴ Key Laboratory of Cryptologic Technology and Information Security, Ministry of Education, Shandong University, Jinan, China

Abstract. Automated cryptanalysis has seen a lot of attraction and success in the past decade, leading to new distinguishers or key-recovery attacks against various ciphers. We argue that the improved efficiency and usability of these new tools have been undervalued, especially for design processes. In this article, we break for the first time the classical iterative design paradigm for symmetric-key primitives, where constructions are built around the repetition of a round function. We propose instead a new design framework, so-called uKNIT, that allows a round-by-round optimization-led automated construction of the primitives and where each round can be entirely different from the others (the security/performance trade-off actually benefiting from this non-alignment). This new design framework being non-trivial to instantiate, we further propose a method for SPN ciphers using a genetic algorithm and leveraging advances in automated cryptanalysis: given a pool of good cipher candidates on x rounds, our algorithm automatically generates and selects $(x + 1)$ -round candidates by evaluating their security and performance. We emphasize that our design pipeline is also the first to propose a fully automated design process, with completely integrated implementation and security analysis.

We finally exemplify our new design strategy on the important use-case of low-latency cryptography, by proposing the uKNIT-BC block cipher, together with a complete security analysis and benchmarks. Compared to the state-of-the-art in low-latency ciphers (PRINCEv2), uKNIT-BC improves on all crucial security and performance directions at the same time, reducing latency by 10%, while increasing resistance against classical differential/linear cryptanalysis by more than 10%. It also reduces area by 17% and energy consumption by 44% when fixing the latency of both ciphers. As a contribution of independent interest, we discovered a generalization of the Superposition-Tweakey (STK) construction for key schedules, unlocking its application to bit-oriented ciphers.

Keywords: uKNIT, low-latency, block cipher, primitive design.

1 Introduction

Designing symmetric-key cryptography algorithms has always been a delicate balance between achieving robust security and optimizing performances. Historically, both the design of cryptographic primitives and their subsequent cryptanalysis were predominantly manual processes, relying on human ingenuity and expertise. However, in the last decade, the community has witnessed remarkable advancements in automated cryptanalysis techniques. Tools based on Boolean satisfiability problem (SAT) [85,102,109,79], Mixed-Integer Linear Programming (MILP) [86,14,15,81], and Constraint Programming (CP) [60,48,39] solvers have become essential in the cryptanalyst’s arsenal, automating substantial parts of the analysis, even including the key-recovery phase [94]. Their ease of use has allowed researchers to explore more complex and interesting search spaces in the analysis of ciphers.

While cryptanalysis has seen significant automation, cipher design has largely remained a task performed by humans. The core structure of most modern symmetric-key algorithms continues to be crafted by cryptographers, with automation typically limited to selecting internal components such as S-boxes, linear layer matrices, and rotation values (mostly via local brute-force searches). Despite the evident power of automated tools in cryptanalysis, their potential in the design phase remains relatively underexplored (one counter-example being the search for large AES-round-based permutations [68,89,95]). We believe that harnessing these tools for the design process could lead to the discovery of more secure and efficient cryptographic primitives.

In particular, symmetric-key cryptography is almost entirely characterized by the iterative function design paradigm, where the primitive is built by repeating a fixed round function with minor variations along the rounds (constants, counters, or rotation values). This design approach is exemplified in widely adopted standards like AES, SHA-2 or SHA-3. Iterative functions were initially favored because they facilitated both compact implementations (using `for` loops) and ease of human-guided analysis/provable security. However, in many use cases, these two advantages are increasingly becoming less relevant. First, in many scenarios, such as in fast software or low-latency hardware, unrolled implementations do not pose significant challenges (they are actually favored in low-latency hardware). Secondly, automated tools now dominate cryptanalysis, rendering the human-friendly iterative structure less critical (these tools fundamentally do not assume nor care about such a structure).

In this paper, we explore whether *abandoning the classical iterative design paradigm could yield symmetric-key algorithms with better security/performance trade-offs*. A non-iterative design framework offers greater flexibility, potentially leading to more diverse and powerful cryptographic constructions. Maybe even more importantly, by creating primitives via incremental addition of rounds and allowing each to vary in structure, the designer might benefit from a locally guided, greedy approach to optimize security and performance simultaneously. This exploration could provide insights into alternative cipher designs that outperform their iterative counterparts. However, implementing such a framework

presents its own set of challenges, and realizing its potential seems not straightforward.

A natural use case to test this new design strategy is low-latency cryptography, which essentially consists in providing secure cryptographic primitives performing with the lowest possible latency in hardware implementations (typically fully unrolled implementations in ASIC), while maintaining area and energy consumption to reasonable levels. This is notably different from the goal of lightweight cryptography (currently undergoing standardization by the NIST [107]), which does not originally aim to optimize latency. This research field has received much attention very recently due to its numerous critical applications: autonomous vehicles, cloud computing, and financial transactions, require secure, low-latency communication for rapid and essential decision-making. More specifically, low-latency ciphers are already used for RAM memory encryption/authentication [91] or to authenticate pointers for system security [12].

The first published cipher with low-latency as stated target is the block cipher PRINCE [32], later upgraded to PRINCEv2 [36]. Other low-latency block ciphers have been proposed by the community [75,78,114,63], as well as low-latency tweakable block ciphers [11,4,5,37,12,6] and various other low-latency primitives such as PRFs [7,13,3,110,2]. Unfortunately, similarly to lightweight cryptography, the heavy constraints imposed on the performance led to several candidates presenting security issues or little security margin [55,83,34,17]. In fact, most low-latency primitives have reduced security claims compared to their functional sizes (e.g. PRINCE only guarantees security up to 2^{126-n} for 2^n data) or very small block sizes [12,37].

Creating a secure low-latency cipher is a complex task as it represents a dual-objective optimization problem: designers must strike a balance between security and latency. Typically, the approach cryptographers take is to focus on one objective first and then attempt to ‘fit’ the other. One can of course take the classical approach of relying on mathematical constructions to ensure a minimum level of security, such as guaranteeing adequate diffusion in the linear layer or a minimum algebraic degree for an S-box. Most low-latency constructions basically use low-latency sub-components (S-boxes, linear layers) in the hope that this will transfer to the global construction. Then, among the candidates with similar security properties, they select the one with the lowest latency. While this strategy could be sound, it may not yield the best security/latency ratio, mainly because when functions are composed, the interactions between different components become too complex to manage effectively. In addition, latency is very hard to predict [78] (in contrast to area and throughput in lightweight cryptography, which can be quite accurately modelled). Being able to build a cipher round per round, optimizing for latency and security at the same time, might partially solve these issues.

Our contributions. In this paper, we propose uKNIT, a new simple design paradigm for building symmetric-key primitives, leaving behind the classical iterative top-down constructions that prevailed thus far. uKNIT does not enforce each round to be the same, and more importantly allows to build the primi-

tive one round at a time (*knitting* a cipher), optimizing for both security and performance. This optimization-guided search strategy explores a much larger space than previous designs and allows to locally adapt the cipher to meet the designer’s global security/performance targets.

Exploring efficiently such a large design space is not trivial and presents many challenges. Our second contribution is a search technique based on a genetic algorithm for the large class of substitution-permutation network (SPN) primitives. This algorithm is generic and can be configured to many different security and performance goals, but for simplicity we describe it within the scenario of an attacker trying to maximize the resistance of his primitive against classical differential and linear cryptanalysis, while minimizing latency.

The third contribution is the application of this technique to the important and very practical use-case of low-latency cryptography, currently undergoing intense study by the community. More precisely, we propose `uKNIT-BC`, a novel 64-bit block 128-bit key ultra low-latency SPN block cipher. Our candidate largely improves over the current best-performing low-latency block cipher, `PRINCEv2` [36] on all crucial security and performance directions at the same time: reduction of the latency by 10%, while increasing resistance against classical differential/linear cryptanalysis by more than 10%. Besides, it shows area reduction of 10% to 20% and power consumption reduction of 8% to more than 50% at different comparison points normalized at different frequencies. To strengthen the confidence in the security of our design, we performed a very thorough analysis regarding many state-of-the-art attack techniques. This contribution not only validates the interest of exploring new primitives within the `uKNIT` framework, but also represents an important achievement in the field of low-latency cryptography by itself.

We emphasize that low-latency is one clear example that we chose to explore in detail in this paper, but we believe our search strategy will lead to more efficient primitives in many other areas and use cases (lightweight cryptography, fast software encryption, etc.).

As an additional contribution, our design process is supported by an entirely automated security and hardware performance evaluation pipeline. At each step of the search, automated cryptanalysis tools conduct complex security analysis of the pre-candidates, while hardware implementations and benchmarks are automatically produced for a more accurate evaluation of their latency (previous works did not predict latency, or only estimated it via modeling). To the best of our knowledge, this is the first time a cryptographic scheme is being built from such a fully automated security/performance evaluation pipeline.

Finally, as a last contribution of independent interest, we describe a novel method to build strong key schedules, generalizing over the Superposition-Tweakey (STK) construction, which was originally introduced in [69] and later influenced various tweakable block cipher constructions [11]. Our generalization, so-called `gSTK`, is particularly interesting as it can apply to bit-oriented ciphers (in contrary to STK), while ensuring sufficient diffusion within the round keys.

Outline. In Section 2, we present the uKNIT framework and in Section 3 a method to automatically generate uKNIT ciphers instances for SPNs, based on a genetic algorithm. In Section 4, we concentrate on the case of low-latency, explaining the rationale for our low-latency block cipher proposal uKNIT-BC, later fully specified in Section 5. An exhaustive security analysis of uKNIT-BC is conducted in Section 6, while implementation benchmarks of the cipher and comparison with competitors are given in Section 7. Finally, we conclude in Section 8.

2 The uKNIT design framework

In symmetric-key cryptography, most block ciphers and other primitives rely on an iterative design paradigm, where a sequence of rounds R_1, R_2, \dots, R_r is applied to the input and where each round R_i applies the same transformation R (up to a small variation, usually a constant value or even a rotation value, as was used to build the 64-bit ARX-based S-box **Alzette** [9]). For example, an r -round iterative block cipher E taking as input a plaintext P and a key K can be written as:

$$E(P, K) = R_r \circ R_{r-1} \circ \dots \circ R_1(P, K).$$

In the case of an SPN block cipher, we would typically have

$$R_i(X, K) = L \circ S(X \oplus K_i \oplus c_i)$$

where S stands for the substitution layer, L for the (usually linear) permutation layer, K_i for the subkey generated at round i from the master key K , and c_i a round dependent constant.

Iterative primitives are usually created directly via a top-down approach: the designer crafts the function R (S and L in the case of SPNs) according to its security/performance needs and this immediately defines the entire primitive. To ensure good performances, the designers can, of course, choose components S and L that are likely to behave well for the targeted efficiency criterion. For security, they should make sure the components provide strong security when repeatedly iterated. Some structure usually helps in this regard, the wide-trail strategy [45] for the AES block cipher being an excellent example.

To the best of our knowledge, examples of existing symmetric-key cryptographic primitives that are not purely iterative constructions would be the Kaliski-Robshaw block cipher [70] or the MD4/MD5/SHA-1 hash functions. In both cases, the few “rounds” composing the primitives use a different internal component (a different Boolean function), in the hope that this would complicate the attacker’s task. For example, a particular strategy to find a good differential characteristic for the first round might fail for the second round. Yet, these constructions remain very iterative. First, only a small function within the round differs. Secondly, each round is in fact composed of many “steps” that are actually the same. Thus, these constructions remained built classically, the designers did not analyze nor leverage the possible special interactions stemming from the rounds being slightly different.

Another line of work deviating a little from the pure iterative constructions is the arithmetization-oriented family of primitives `LowMC` [1] and `RASTA` [54]. In these SPN designs, the substitution layer S is fixed, but the permutation layer L is chosen pseudo-randomly at each round. However, these designs do not try to precisely analyze and leverage any security or performance gain by having different L at each round (as they are supposed to look randomly chosen). Again, these primitives were created using a top-down approach, where all the rounds were specified at the same time.

Finally, we can mention constructions such as `HADES` [62] and `PRINCE` [32] that are classical iterative structures, but where a special round portion is present at the middle of the cipher only.

2.1 The uKNIT ciphers space

We propose `uKNIT`, a simple method to build ciphers, generalizing the classical iterative constructions. A `uKNIT` primitive is defined by a sequence of rounds R_1, R_2, \dots, R_r applied to the input, but where each round R_i does not necessarily apply the same transformation. Reusing our block cipher example, we would typically have

$$R_i(X, K) = L_i \circ S_i(X \oplus K_i \oplus c_i)$$

where layers L_i and S_i might be completely distinct.

One obvious benefit from the `uKNIT` framework is that the space of the functions considered is larger, iterative constructions being a subset of `uKNIT` ones. A larger search space means candidates with potentially better security/performance trade-offs. In addition, with their very distinct rounds, `uKNIT` ciphers present a natural tendency to avoid strong alignment. More importantly, `uKNIT` ciphers space allows a paradigm shift in how we build symmetric-key primitives. Instead of crafting the scheme directly with a top-down approach, with `uKNIT` one can be much more exotic in how ciphers are created. One could use an incremental process where rounds are added one at a time, both in the forward (appending a round) or backward (pre-pending a round) direction, trying to optimize some fitness function f . This allows to optimize locally the cipher’s design choices, expecting that this improvement will translate on a global scale. Given an existing construction with r rounds, one can build a set of $(r + 1)$ -round candidates (or $(r + s)$ rounds, where s is a relatively small integer) and select the one that maximizes f . One can even consider simply updating a r -round candidate to a new r -round one with a higher value according to the fitness function. Another possibility is to combine smaller ciphers to create larger ones: from a r_1 -round candidate and another r_2 -round candidate, we can produce a $(r_1 + r_2)$ -round one by concatenating the rounds. Note that f can be fully adapted to the use case (in terms of security and performance criterion), which makes `uKNIT` a very generic symmetric-key construction framework.

2.2 The knitting methodology: gradually building uKNIT ciphers

We describe our methodology which we instantiate in Section 3 for the case of SPN ciphers. We assume that the designer has a goal (or a set of goals) for the cipher, represented by the fitness function f . We further consider that the designer has access to performance evaluation and cryptanalysis tools to evaluate the fitness function on any cipher candidate. The methodology is to gradually build the cipher up to i rounds, a process that we refer to as *knitting* a cipher. When building the i -round cipher, we take as input a group of ciphers of i or less rounds, but that does not satisfy our goals. These ciphers are fed to the optimization algorithm which has access to the analysis tools and eventually the optimization algorithm is expected to output an i -round cipher that is better, with respect to the fitness function, than all the input ciphers. This process is repeated until the cipher meets our goals. We can also vary the optimization algorithm at different steps of the process, in order to achieve a good trade-off between computational power and cipher behaviour. One step of this process is summed up in Figure 1.

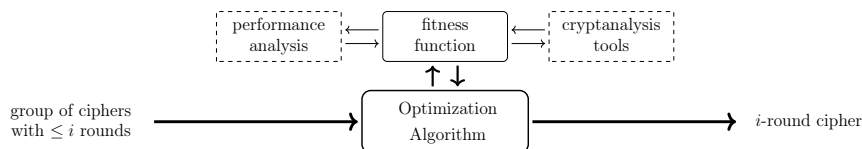


Fig. 1: One step of our knitting methodology.

There are many ways to interpret this methodology. For example, consider the process of designing an r -round cipher starting from a $(r - 2)$ -round cipher. We can consider that we run the methodology twice, once to get an $(r - 1)$ -round cipher, then to get an r -round cipher, or simply run it once to jump from $r - 2$ to r , or even run it many times before we get from $r - 2$ to $r - 1$, it is a matter of different interpretations, as all these descriptions can be referring to the same execution.

3 Application of uKNIT to SPNs

The uKNIT cipher space is extremely large, which naturally triggers the question of how to find a good candidate within this space. We provide in this section one practical solution to knit an SPN cipher, mainly based on a genetic search, as well as a smart brute-force search when necessary.

Genetic search algorithms are evolutionary algorithms that are commonly employed to solve multi-objective optimization problems and offer several advantages, including the ability to handle complex and nonlinear optimization challenges. Before we introduce the genetic search and the smart brute-force algorithms, we describe the global search structure along which we knit the cipher.

3.1 The fitness computation

During the search, we need to evaluate the fitness of the current candidate, in terms of performance and security. Given a set of tools that can automatically evaluate the performance and the security of a candidate, a specific function needs to be defined to assign it a score. This function acts as a compass and depends on the use case considered: what platform, what performance criteria, what security target, etc. It can be computed from exact evaluations or merely approximations. It might even dynamically change during the search process (if some criterion is favored at the beginning of the search and another one towards the end).

Remark. To reduce the number of computations required during the security evaluations, especially when dealing with a large number of rounds, it might be useful to set an upper bound on the number of rounds we are analyzing. An upper bound can be chosen as the number of rounds that an attacker uses as a distinguisher for a key recovery attack for example. In other words, we can choose to only study “windows” of rounds from the cipher candidates, instead of the entire ciphers. As such, we propose the following definition to facilitate the description of the process.

Definition 1 (Window). For a cipher C that has r rounds, the i^{th} l -length window is the l consecutive rounds of C starting from the i^{th} round, where $0 \leq i < r$ and $1 \leq l \leq r - i$. The i^{th} l -length window is denoted by $\mathcal{W}(i, l)$.

Example. The window $\mathcal{W}(2, 8)$ refers to the 8-round cipher obtained by extracting rounds 2 through 9 from the cipher.

3.2 The search - Knitting an SPN cipher

The biggest challenge when knitting an SPN cipher is the huge search space inherent from uKNIT. To control the variability of the search, we can construct the cipher part by part. As a starting point, we generate a set of random x -round ciphers (x being a relatively small value), evaluate and sort these x -round candidates according to their fitness. These candidates are the first generation of the genetic search. Then, we trigger the generation evolution by breeding and mutation.

In each generation, we discard the bad performing ciphers and derive new candidates from the better performing ones. This continues for several loops (or generations). Once we are satisfied with the set of x -round ciphers, we add one round at a time to generate ciphers with a higher number of rounds.

Alternative optimization approach. If the computational power is limited, the search can be divided into two phases if needed. We can employ a genetic algorithm to search for candidate ciphers up to round i . In this phase, the number of ciphers that require evaluation can grow rapidly (in the case of uKNIT-BC, it easily exceeds 100,000 candidates). The computational power constraints are primarily dictated by the time required for the fitness function to evaluate the

candidates. For the next $r - i$ rounds, we can use a smart brute-force algorithm to further extend the number of rounds. However, up to computational power limitations, we recommend using the genetic algorithm as much as possible, as it seems to consistently produce better candidates compared to the brute-force approach. Figure 2 provides an overview of the entire process. In fact, we use the pure genetic approach in the remainder of the paper.

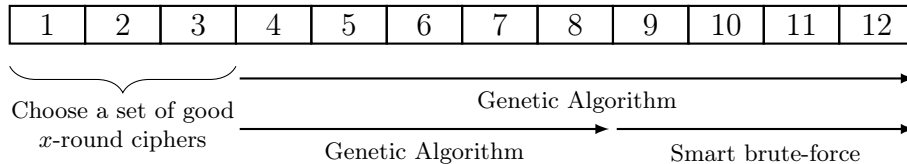


Fig. 2: Overview of the search process to knit an SPN cipher.

3.3 Genetic algorithm

A genetic algorithm is a search heuristic inspired by the principles of evolution and natural selection. It is particularly effective for solving optimization problems where the density of good solutions is limited due to a large solution space, making brute-force methods inefficient or impractical.

In the following paragraphs, we examine each component of our genetic algorithm and explain how they contribute to the overall search process. We provide in the description some precise parameters values that we found to be working well during our experiments. Of course, these depend on the use cases and the cipher properties. The designer should not hesitate to test/adapt them.

Overview. At a high level, Figure 3 illustrates the flow of our genetic algorithm. The process begins with initialization, during which the population is generated. We then evaluate the *fitness* of each candidate in this population to identify two key groups: the *breeding* population (those selected to breed before being eliminated) and the *surviving* population (those that continue to the next round).

The two groups undergo *breeding* and *cloning*, respectively, and may experience a potential *mutation* before moving on to the next generation. This cycle continues for some generations. Afterward, we append two new components, a permutation layer and a substitution layer, to the current cipher candidates. This process repeats until we reach the desired level of security and performance.

Initialization. For the initialization part of the search algorithm, we require a pool of ciphers. We have set the number of candidates to be 200, which turned out to be a good trade-off in our experiments. We construct each candidate cipher C in this manner:

$$C = S^{x-1} \circ L^{x-2} \circ S^{x-2} \circ \dots \circ L^0 \circ S^0$$

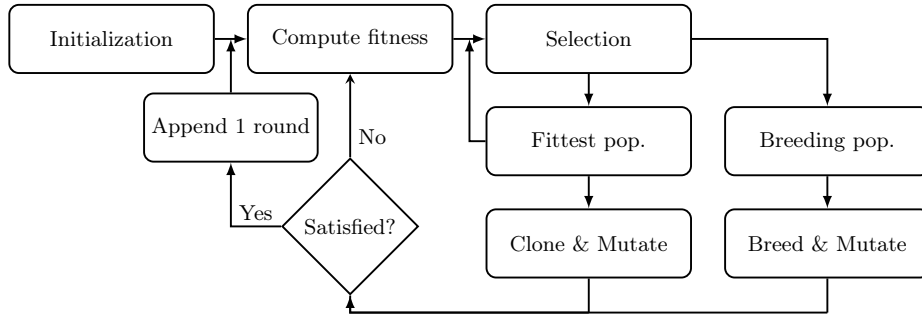


Fig. 3: The flow of the genetic algorithm to search for SPN ciphers.

where S^i and L^i are chosen randomly from sets \mathcal{S} and \mathcal{L} respectively. These are sets of substitution and permutation layers that we have pre-set in the beginning. The choice of these sets for the case of uKNIT-BC is discussed in Section 4. For the lower number of rounds, we set each round to have 100 generations. This value can be adjusted and is dependent on the available computational power and the complexity of the fitness function.

Selection. This process throws away the weaker candidates, following the concept of “survival of the fittest”. However, considering only the *fitness* may cause the population to converge too quickly. Thus, we have to bring in some *diversity*, as well. Overall, the selection process is split into two parts:

- **Selecting the fittest ciphers.** The top 10 ciphers (sorted by on *fitness*) are selected. This set of ciphers is known as the “previous generation” as they are preserved until the next elimination, unless it is the last generation of its respective rounds. To extend the influence on the next generation, these ciphers are cloned and have a mutation introduced with probability 1.
- **Selecting the breeding population.** The set of ciphers we select here will breed the next generation of ciphers. The process of selecting this group is based on both its fitness and a diversity measurement from a new candidate to the current pool of candidates:
 1. Initialize an empty set \mathcal{C}
 2. For each cipher C in the population set: Compute the diversity of C from all $C' \in \mathcal{C}$
 3. Normalize the fitness and diversity scores
 4. Compute the score: $a \cdot (\text{fitness}) + b \cdot (\text{diversity})$ with $a = 1$ and $b = 10$
 5. Select the cipher with the top score and place it in \mathcal{C}
 6. Repeat from Step 2 until we have obtained 40% of the population size.

The distance of a cipher C to another cipher C' is hard to quantify as we neither have a notion of quantifiable distance between S-boxes nor the linear layers that considers their cryptographic properties. Thus, for the substitution layer, we simply count the number of differing S-boxes and for the linear layer, we count the number of different entries in the (binary) matrix. The exact details of the diversity computation are shown in Algorithm 1 of Appendix A.

Breeding. There are three types of breeding we implement: single-point crossover, double-point crossover and uniform crossover. To illustrate a single-point crossover, consider two ciphers C and C' as the parents in a chain form:

$$\begin{aligned} C &: S_0 - P_0 - S_1 - P_1 - \dots - S_n \\ C' &: S'_0 - P'_0 - S'_1 - P'_1 - \dots - S'_n \end{aligned}$$

then a single point crossover chooses one part of the link and swap the remaining parts of the chain with the other parent to create two offspring:

$$\begin{aligned} C'' &: S_0 - P_0 - S_1 - P_1 - S'_2 - P'_2 - \dots - S'_n \\ C''' &: S'_0 - P'_0 - S'_1 - P'_1 - S_2 - P_2 - \dots - S_n \end{aligned}$$

A double point crossover chooses two points instead of one and swap the parts accordingly and a uniform crossover would be a random mixture of S-boxes and linear layers. Note that a uniform crossover is very likely to destroy any cryptographic properties across different rounds and hence, it is assigned a lower probability. The breeding process proceeds as follow:

1. Randomly select randomly with uniform probability two candidates C and C' from the breeding population.
2. Let C and C' undergo either single-point, double-point, or uniform crossover with probability 0.4, 0.4 and 0.2, respectively.
3. With a probability of 0.05, the new ciphers undergo mutation. If the new candidates after breeding are already included in the set of offspring, then this probability is increased to 1. The details of the mutation for the case of uKNIT-BC are explained in Section 4.
4. Repeat this process until we obtain a total of 200 new ciphers.

3.4 Smart brute-force

While genetic algorithm should be favored, we introduce a smart brute-force method in the case where the genetic algorithm is spending an excessive amount of time evaluating the large number of candidates. We emphasize that this method is simply a way to reduce the number of candidates we need to evaluate, thus decreasing the time complexity by a constant factor. This method can be applied after we have obtained fairly decent candidates with high number of rounds. Unlike a naïve brute-force method, this smart brute-force takes into account some specific factors, which we discuss here.

First, note that in the smart brute-force approach, we do not employ the concept of generations, or rather, it can viewed as if there is only 1 generation per round. However, as compensation, we can afford to increase the number of candidates.

The smart brute-force approach basically consists in adding a new round, alternating at the start or the end of the original candidate, such that the fitness function is locally optimized. More precisely, the new round is not generated

randomly, but by choosing carefully its sub-components (S-Box and linear layer) to heuristically increase the fitness function.

For instance, in our case where the target security is the resistance against classical differential cryptanalysis, the smart brute force approach takes in an x -round cipher as well as one of its best differential characteristic. This approach generates many potential linear layers for the additional round and selects one that maximizes the number of active S-boxes in the new round based on the best differential characteristic given. The output is an $x + 1$ -round cipher.

4 Design rationale of the uKNIT-BC block cipher

We apply here our SPN search strategy to the case of low-latency block cipher. We use candidates of $x = 3$ rounds to populate our first generation.

4.1 Fitness computation during the search

Our performance goal is to minimize the total cipher’s latency, while providing enough security against state-of-the-art attacks. Therefore, our fitness or objective function must account for both factors. To achieve this, we rely on open-source tools to evaluate these metrics. While there are various methods to measure a cipher’s security, we chose to preliminarily assess it based on the best differential characteristic and linear trail. To evaluate the best differential characteristic and linear trail, we convert a cipher candidate automatically into SAT models that describe its differential characteristic and linear trail, which is then analyzed by a SAT solver. The modeling method we used follows the techniques described in [102]. For this purpose, we use the SAT solver `CaDiCaL` [19], but our model is compatible with any SAT solver that accepts DIMACS format. For the latency measurement, we use `OpenLane` [97], an open-source automated RTL to GDSII flow that leverages other available open-source CAD tools such as Yosys [111], a set of optimization scripts and synthesis strategies for latency and area and an open-source PDK known as Skywater 130nm⁵, which is a collaboration between Google and SkyWater Technology Foundry. Unlike other open-source PDKs, Skywater 130nm has been silicone-proven both academically and commercially.

After obtaining the probability of the best differential characteristic ($prob_d$), the bias of the best linear trail ($bias_l$), and the latency of the cipher (lat), the fitness of a candidate is computed as:

$$\frac{\max[-\log_2(prob_d), -2 \cdot \log_2(bias_l)]^2}{lat}$$

Remark. While it may look more intuitive to have the above expression to be $\frac{\text{security}}{\text{latency}}$, the security does not scale linearly with the number of rounds (whereas it does for latency). In particular, in the early stages where the number of rounds

⁵ <https://github.com/google/skywater-pdk>

is small, a proportionate increase in security should be prioritized over a proportionate decrease in latency. The above expression is selected during the initial testing phase where we tested a few different expressions and this expression produces candidates that are consistent with our goals for the cipher.

4.2 Structure selection

As we are using a genetic algorithm to automate the search process, we are concerned about having a good starting point and properly defining the set of possible components within our search parameters. Therefore, our initial starting point is somewhat influenced by existing low-latency and low-energy designs such as PRINCE, MANTIS and MIDORI.

Type of construction. For our construction, we opted for the SPN structure with a 64-bit state size. The extensive amount of cryptanalysis conducted on SPN ciphers has given cryptanalysts a deeper understanding of this structure and, naturally, one significant advantage of this choice is the wider availability of automated tools optimized for this task. While we briefly considered the Feistel construction, it proved to be unsuitable for a low-latency context, as only half the state is processed at any given time. Additionally, using a 64-bit structure ensures that the search space will not be too large for automated tools to handle. It is important to note that mutations within our genetic algorithm do not alter the type of structure chosen.

Number of rounds. During the search, a constant evaluation of security and performance is required for the fitness function. One problem we encountered when we attempted to generate ciphers with a short number of rounds was that the construction required much stronger S-boxes and linear layers. This caused the SAT solver to have a hard time solving the model given and thus, slowing down the process significantly. Moreover, ciphers with fewer rounds are also more vulnerable to meet-in-the-middle (MitM) attacks. With the 64-bit SPN structure of uKNIT-BC and its (probably sparse) diffusion layer, we evaluated that at least 10 rounds would be needed to resist MitM attacks. From there, we did not know in advance how many rounds would ensure enough security according to our criterion, but a first rough evaluation indicated a number of rounds around 12 to 14.

4.3 Choice of substitution layers

Initialization. For the substitution layer, we opted for 4-bit S-boxes due to their extensive study and documentation. These S-boxes can be categorized based on affine equivalence classes, with S-boxes within the same class sharing very similar security properties. This categorization allows us to efficiently eliminate entire sets of S-boxes when necessary. This is particularly important as unlike other designs, we do not limit ourselves to using the same S-box throughout the substitution layer, or even within a single layer. Consequently, the search space is

very large. Initially, we considered all possible 4-bit S-boxes, excluding those with undesirable security properties, such as those that contain linear components or probability-one differential transitions. While this approach enabled us to design ciphers with slightly lower latency but stronger security properties, the complexity of these ciphers made them too challenging for our automated tools to analyze, particularly when evaluating a large number of candidates. Eventually, we restricted our selection to the S-box used in MANTIS (which is also MIDORI’s Sb_0) as well as the S-boxes that are simple bit-transpositions of it, affecting both the inputs and outputs. The MANTIS S-box was chosen for its notably lower latency compared to other S-boxes, and applying bit transpositions will not change its latency. Surprisingly, this decision significantly sped up the performance of the automated tool.

Mutation. For the S-boxes, we have identified four mutations meant to reflect an increasing distance in terms of the S-box properties:

1. A bit permutation on the output
2. A linear permutation on the output
3. An affine permutation on the output
4. A swap of indices: $S[i] \longleftrightarrow S[j]$ for some $i, j \in \{0, \dots, 15\}$

For the first three cases, we allow the S-boxes to be in the same affine equivalence class, which help to preserve most of the security properties. However, our initial empirical results showed that the increase in security is not worth the increase in latency, thus, for uKNIT-BC, we eventually decided to set the mutation to only allow bit permutations only.

4.4 Choice of linear layers

For our low-latency cipher design, the depth of the linear circuit is a critical factor, just as it is in other ciphers targeting minimal latency. To ensure maximum diffusion for each output bit, we restrict the set of possible circuits to be a set where each output bit is computed through a series of p -XOR gates. This configuration corresponds to having $p + 1$ ‘1’s in each row of the linear transformation matrix. To prevent excessive fan-out where a single signal is used in too many places, each input bit is also used in p -XOR gates. This is equivalent to having $p + 1$ ‘1’s in each column of the matrix. Under these conditions, p cannot be odd if we want the linear circuit to be invertible. Therefore, if p is odd, we allow one row or column to have $(p - 1)$ -XOR gates instead. In our design, we considered cases where $p = 0, 1,$ and 2 . We did not explore $p \geq 3$ as this would increase the depth of the circuit by one, which would negatively impact latency when taking the key addition into account (refer to Figure 4 for an illustration of XOR depth).

Almost-MDS matrices. Among the set of 2-XORs invertible matrices, there exists a subset that exhibits a critical mathematical property known as the almost-MDS property. This property is significant because it offers a strong level

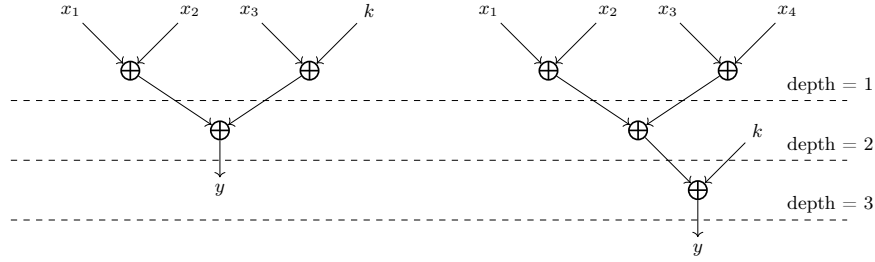


Fig. 4: An illustration of the depth with 4 and 5 variables (including the keys).

of diffusion while maintaining relatively low complexity in terms of the number of XOR gates required. To the best of our knowledge, two low-latency cipher designs have utilized this type of linear layers: MIDORI and PRINCE. We give the values of the MIDORI and PRINCE matrices in Appendix C.

We aim to leverage these almost-MDS matrices to achieve an optimal balance between security (via strong diffusion) and efficiency (via minimal latency). However, we only use them as a starting search point, and apply several transformations as part of the genetic algorithm. The almost-MDS matrices are defined over nibbles, and by performing a row-wise or column-wise rotation at this block matrix level, the almost-MDS property can be preserved. However, in our case, we seek greater variation, even if it means deviating from the almost-MDS property. Based on our experiments, as long as the deviation is not significant, the overall quality of the linear layer remains strong. Thus, for the initialization phase, we consider the case that we are allowed to do elementary row swap (ERS) and column swap operations (ECS) at the bit-level. More specifically, we follow these steps to generate our initial linear layer:

1. Randomly select either $\tilde{M}_{prince}^{(0)}$ or M_{midori}
2. Apply a random number of ECS operations
3. Apply a random number of ERS operations
4. Repeat Step 1 to 3 to obtain four of them for each of the 16-bit columns
5. Verify that it is invertible, if not, repeat the process from Step 1

Although the ERS and ECS mentioned above allow for some diffusion to other columns (not exactly a MixColumns), to ensure the various cells are mixed thoroughly, we also implemented the ShiftRows operation identical to that of PRINCE after the matrix multiplication operation for all the candidates:

$$\begin{bmatrix} s_0 & s_4 & s_8 & s_{12} \\ s_1 & s_5 & s_9 & s_{13} \\ s_2 & s_6 & s_{10} & s_{14} \\ s_3 & s_7 & s_{11} & s_{15} \end{bmatrix} \rightarrow \begin{bmatrix} s_0 & s_4 & s_8 & s_{12} \\ s_5 & s_9 & s_{13} & s_1 \\ s_{10} & s_{14} & s_2 & s_6 \\ s_{15} & s_3 & s_7 & s_{11} \end{bmatrix}$$

Mutation. In the linear layer, represented by a 64×64 binary matrix, the mutation function is similar to the operation we perform in the initialization. It

operates by randomly selecting two rows and swapping them. The same process is then applied to the columns. This is equivalent to rewiring two input and output bits in the linear layer. While this operation carries a high probability of disrupting the almost-MDS property further, our experiments show that as long as the number of swaps is kept low, the resulting candidates often demonstrate enhanced security, particularly in terms of their best differential characteristics.

4.5 The key schedule

Jean *et al.* [69] proposed a generalized construction of building key schedules for key alternating ciphers, dubbed as the Superposition-Tweak-key (STK) construction. For a block cipher with block size n , the (twea)key is divided into p blocks, each of size n . Each key block is processed independently. Each round, a function h' permutes the nibbles of each block. Then, an LFSR is applied to each nibble. In the same block, the same LFSR is applied to all nibbles, while different blocks must have different LFSRs. Last but not least, the key blocks are XORed together to generate the round key.

The goal of this construction is to ensure that when the cipher is used with two related keys with a known difference, the number of nibbles where the key blocks have non-zero difference but the corresponding round key has 0 difference is limited. Indeed, the construction achieves this goal efficiently, and has been used in several TBC designs. In [41], Cogliati *et al.* studied extending the STK framework to more than 3 branches (as in the first generation of STK-based ciphers), correcting an error another proposal [88]. In [72], Khairallah *et al.* studied how smart mode designs coupled with STK-based TBCs can lead to efficient implementations and cheap side-channel countermeasures, which is relevant in the domain of low-latency ciphers, since expensive masking is not an option. However, the original presentation of the STK construction has two drawbacks for our use case:

1. The STK construction was particular to AES-like ciphers, where the whole cipher is nibble- or byte-oriented. Thus, it may not be suitable for bit-oriented ciphers.
2. The number of rounds in our proposal is small. Thus, having a key schedule where the nibbles do not interact with each other can lead to slow diffusion.

For these reasons, we need a generalization of the STK construction that can work for bit-oriented ciphers, and has sufficient diffusion in the round keys. In order to do so, we have to view the round keys as bit vectors. In Definition 2, we give a mathematical representation of any linear key schedule, viewed over binary vector spaces.

Definition 2. *Let p, n, k be three positive integers, where n is the block size, k is the (twea)key size, and $k = np$. The master key K_m is a vector from the vector space \mathbb{F}_2^k . For $i \in \{0, 1, \dots, r\}$, where r is the number of rounds, $K_i \in \mathbb{F}_2^n$ are the round (twea)keys. Then, any linear key schedule can be represented as a sequence of $n \times k$ matrices over \mathbb{F}_2 : G_0, G_1, \dots, G_r , such that $K_i = G_i K_m$. We call the set of matrices $\mathbf{G} = \{G_0, G_1, \dots, G_r\}$ a linear key schedule.*

Given Definition 2, we can now give a stronger goal for the key schedule, but does not only work for nibble-oriented ciphers.

Definition 3. Let \mathbf{G} be an r -round linear key schedule according to Definition 2. The key schedule is a generalized STK (gSTK) key schedule if for any p indices $0 \leq i_0 < i_1 < \dots < i_{p-1} \leq r$, $[G_{i_0}^T \ G_{i_1}^T \ \dots \ G_{i_{p-1}}^T]$ is a full rank matrix, where G^T is the transpose matrix of G .

It is easy to see why the property in Definition 3 captures the essence of the STK construction. Note that if we select $p' < p$ round keys, it must hold that there are many keys K_m that lead to the same set of p' round keys. Thus, an adversary with full control over the key difference can choose a set of p' round keys such that $K_m \neq K'_m$, but the round keys are equal. We want to make sure that this does not hold for $p' \geq p$. Thus, we want that the relation between any set of p round keys and the input key to be bijective. Definition 3 is a more formal way of writing this. For small r , it is easy to check whether this property holds. It boils down to checking the determinants of $\binom{r}{p}$ matrices of size $k \times k$.

Another interesting observation is that a powerful adversary can always choose a key such that a specific set of p rounds keys take a specific value. However, this determines all the other round keys. Thus, we can set G_0 to G_{p-1} such that the first p rounds keys are simply blocks of the input keys. This allows us to reduce the latency in unrolled key schedule as for the first p key additions, the round keys are readily available, and the p^{th} round key is only used after p rounds, so we can afford to use a circuit with somewhat higher latency. Since the linear key schedule is typically lighter than the cipher, the key schedule has minimal effect on the overall latency. The challenge now becomes how to select the matrices. In the remainder of this section, we shall focus on the case of $n = 64$, $k = 128$ and $p = 2$, and discuss possible options of selecting the matrices in efficient and smart ways.

Based on our observations, we assign $G_0 = [I \ 0]$ and $G_1 = [0 \ I]$. Then, the design problem becomes: Find $(r - 1)$ 64×128 matrices G_2, \dots, G_r such that

$$\begin{bmatrix} G_i \\ G_j \end{bmatrix}, \begin{bmatrix} I & 0 \\ & G_i \end{bmatrix} \text{ and } \begin{bmatrix} 0 & I \\ G_i & \end{bmatrix} \text{ are all invertible } \forall 2 \leq i < j \leq r.$$

We now re-introduce some simplifications from the STK construction to reduce our search space.

Method I: We consider the case where each block of the key is treated independently, and then the different blocks are XORed to get each round key. This translates to choosing $2r - 2$ 64×64 matrices A_2, \dots, A_r and B_2, \dots, B_r , s.t.

$$A_i, B_i \text{ and } \begin{bmatrix} A_i & B_i \\ A_j & B_j \end{bmatrix} \text{ are all invertible } \forall 2 \leq i < j \leq r.$$

Method II: We refine the analysis further by considering an iterative structure. We choose 2 64×64 invertible matrices A and B such that

$$\begin{bmatrix} A_i & B_i \\ A_j & B_j \end{bmatrix} \text{ is invertible } \forall 1 \leq i < j \leq r - 1.$$

This allows us to simplify the condition to check to a condition on the differences between repeated iterations of A and B . In particular, A and B satisfy the gSTK requirement if

$$0 \neq \det \begin{pmatrix} A^i & B^i \\ A^j & B^j \end{pmatrix} = \det(A^i) \det(B^j - A^j A^{-i} B^i) = \\ \det(A^i) \det(B^j - A^{j-i} B^i) = \det(A^i) \det(B^{j-i} - A^{j-i}) \det(B^i) \quad (1)$$

which translates to A and B being invertible, and $B^l - A^l$ is invertible for $1 \leq l \leq r - 2$.

Method III: This method, and Method IV, use a nibble structure, but we have faster diffusion compared to just nibble permutations. Instead of defining A and B as binary matrices, we define them as matrices over the finite field $\text{GF}(2^b)$, where b is typically 4 or 8. We also define $B = AC$, where $C = cI$, for $c \in \text{GF}(2^b)$ and $c \neq 0$ and $c^i \neq 1 \forall 1 \leq i \leq r$. We can show that this is sufficient since C commutes with any matrix: $AC = AcI = cAI = cIA = CA$. Thus,

$$\det((AC)^r - A^r) = \det(A^r C^r - A^r) = \det(A^r) \det(C^r - I) = \det(A^r) \det((c^r - 1)I)$$

Incidentally, if A consists of elements 0 and 1 only, and each row has only one non-zero element, then this method coincides with the original STK construction.

If c is a primitive element of a field defined by a primitive polynomial, then the multiplication can be implemented by a Linear Feedback Shift Register (LFSR) of maximal length $2^b - 1$ and the condition is satisfied as long as $r \geq 2^b - 1$. For $b = 4$, multiple LFSRs can suffice, such as the Galois LFSR defined by the primitive polynomial $x^4 + x + 1$ and the Fibonacci LFSR defined by the primitive polynomial $x^4 + x^3 + 1$. In fact, the two are defined over two isomorphic fields and we use the latter in our design. See [106, Section 7.4] on how to implement LFSRs from primitive polynomials using both methods, labelled in the book as Method I (Fibonacci LFSR) and II (Galois LFSR).

Method IV: We could use different matrices for different rounds on the form $B_i = c^{i-2} A_i$ and the analysis follows.

An interesting observation in methods III and IV is that as long as the matrix/matrices A are invertible and c is of a sufficiently large order, then the choice of A does not affect the gSTK property and we can use any matrix, from nibble permutations, all the way up to MDS matrices or very dense matrices.

Rationale of the key schedule of uKNIT-BC. Although we do not claim any security for uKNIT-BC in the related-key setting, we still want a well-designed key schedule as it enhances the security against key-recovery attacks by introducing non-trivial relations between round keys.

We expect to achieve two goals for the key schedule:

- It should not be too heavy such that it increases the overall area and latency;

- It should be easy to model by automatic search tools, as it will facilitate the search and provide lower bounds against the related-key differential attacks.

The two goals require the design of key schedule be simple, thus the gSTK framework discussed above is ideal for us. As such, we instantiate a particular instance of the gSTK, using Method III.

According to Method III, the first two round keys use the master keys directly. For the remaining round keys, we only need to choose proper $A \in \text{GF}(2^4)^{4 \times 4}$ and $C = cI$. First, to avoid the heavy field multiplication, we choose an LFSR used by SKINNY-64 to play the role of c . Second, for the choice of A , we define $A = P \circ M$ where P is a nibble-permutation matrix, while $M \in F_{2^4}^{16 \times 16}$ is a matrix with some XORs that provide a stronger diffusion. To make the key schedule simple, we restrict that all the entries of M are either 0 or 1. which is inline with the rationale that some previous tweakable block cipher designs have considered. Using some XORs in M would speed up the diffusion, but too many XORs would also significantly complicate the related-key differential lower bounds search. Therefore, we chose M that contains one XOR per 4 rows (columns) for uKNIT-BC key schedule (see Section 4.5). As for P , we use a brute-force search to search among all cyclic permutations such that for every four rounds, each key position must be in position 3, 6, 9 or 12 where an XOR operation is applied. The criteria for a good choice of P is one that has a decent amount of diffusion. Eventually, we chose one that after 11 iterations of $G = [P \circ M, P \circ M \circ L]$ (L is a matrix corresponding to applying an LFSR to each nibble), every nibble of K_{12} is a function of at least 10 nibbles of K_m . P is given in Section 4.5.

Such a design achieves a good balance between the diffusion speed and the search efficiency for the security lower bounds. With the SAT tools, we check the lower bounds for the related-key differential probability for all windows. The data is shown in Table 9 of Appendix F.

5 Specifications of the low-latency block cipher uKNIT-BC

Since each component of the cipher is distinct, its description is more complex compared to other conventional ciphers. As far as possible, we try to give a compact specification of our cipher. We also provide a GitHub repository containing the implementation and test vectors here:

<https://anonymous.4open.science/r/uKNIT-implementations-7ABA>

5.1 Overview

uKNIT-BC is a 64-bit block, 128-bit key block cipher that contains 12 rounds (a total of 12 substitution layers and 11 linear layers). It also has 13 round-key additions. Each round, except the final one, consists of two layers: substitution layer and linear layer. The final round contains only the substitution layer. Figure 5 describes the overview of uKNIT-BC. In our description, each n -bit variable is a bit column vector indexed $0, 1, \dots, n$ from top to bottom. When we refer to a nibble vector, the indexing order is the same and nibble i refers to the bits $4i$ to $4i + 3$, from top to bottom.

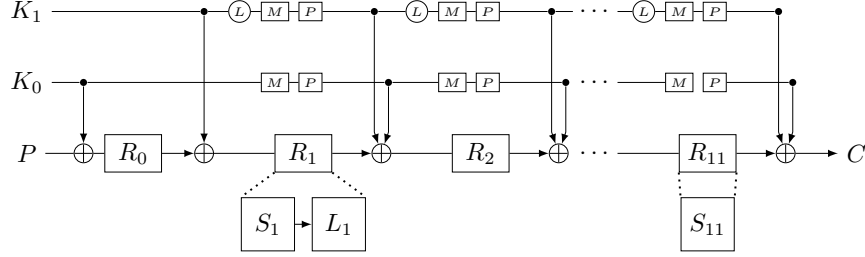


Fig. 5: The structure of uKNIT-BC including the key schedule.

5.2 Substitution layers

The 4-bit S-boxes applied to each nibble of the internal state in uKNIT-BC can be represented by $(D \circ S_{\text{MANTIS}} \circ B)(x)$ where S_{MANTIS} is the MANTIS S-box ($S_{\text{MANTIS}}[x] = [c, a, d, 3, e, b, f, 7, 8, 9, 1, 5, 0, 2, 4, 6]$) and D and B are transposition matrices given in Table 5 of Appendix B.1.

5.3 Linear layers

Our linear layers L_i are very sparse 64×64 binary matrices applied on the entire internal state. Since we have exactly 3 indices in each row having “1” with the rest being “0”, we can describe the matrices solely based on these indices. Table 7 and 8 of Appendix B.2 provide the list of these indices.

5.4 Key schedule

The key schedule of uKNIT-BC is depicted in Figure 5. Given the 128-bit master key $K_m = K_0 || K_1$, K_0 and K_1 are used as the round keys of the first two rounds. For $i \geq 2$, the round key K_i is computed by

$$\begin{aligned} K_0 &= P \circ M(K_0), & K_1 &= P \circ M \circ L(K_1), \\ K_i &= K_0 \oplus K_1, \end{aligned}$$

where L is a linear operation that applies the SKINNY-64 LFSR to each nibble of K_1 . $L : (x_3 || x_2 || x_1 || x_0) \rightarrow (x_2 || x_1 || x_0 || x_3 \oplus x_2)$. The permutation P is defined as $P = [3, 8, 13, 2, 15, 9, 11, 5, 0, 6, 14, 10, 7, 12, 4, 1]$. More precisely, the 64-bit input key K is split into 16 nibbles, and the output is $K'[i] \leftarrow K[P[i]]$, where $K[i]$ is the i^{th} nibble.

Finally, M is a matrix defined as $M = \begin{bmatrix} M' & & & \\ & M' & & \\ & & M' & \\ & & & M' \end{bmatrix}$ where $M' = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 1 \end{bmatrix}$.

6 Security analysis of uKNIT-BC

In this section, we perform an extensive security analysis of uKNIT-BC with state-of-the-art cryptanalysis. We give in Table 1 the best key-recovery attacks we obtained. Due to page limitation, we only provide here a summary of our analysis. All the details and more analysis are provided in Appendix G.

Security claims and attack model. We claim that there is no attack that can break the full uKNIT-BC with less than 2^{112} time complexity and 2^{47} (chosen / adaptively-chosen) plaintext-ciphertext pairs (2^{50} bytes). These two numbers consider the NIST lightweight cryptography requirement [90], and they are similar to what is claimed for PRINCEv2. We do not claim any security regarding related-key, known-key or chosen-key attacks. We did a complete cryptanalysis on uKNIT-BC and did not find any attack that threatens its security. We encourage third-party cryptanalysis on any (consecutive) windows, even surpassing the complexity limitations we set, as they are beneficial to explore the security margin of uKNIT-BC.

Notations. We first introduce a few more notations to facilitate the descriptions of our attacks. The S-boxes and linear layers used in the i^{th} -round uKNIT-BC are denoted by S_i and L_i respectively, where $0 \leq i \leq 11$ (the L_{11} is omitted in uKNIT-BC). For $0 \leq i \leq 10$, the input of the i^{th} -round is x_i , and after the XORing with the round key k_i , it becomes x'_i , i.e., $x'_i = x_i \oplus k_i$. Then, we apply S_i to x'_i to get y_i and apply L_i to y_i to get x_{i+1} . For the last round, i.e., $i = 11$, we have $x_{12} = S_{11}(x_{11}) \oplus k_{12}$.

In differential cryptanalysis, given x_i, x'_i, y_i , their differences are denoted by $\Delta(x_i), \Delta(x'_i)$ and $\Delta(y_i)$, respectively. Given a state $z \in \{x, x', y, k\}$, $z[t]_b$ is the t -th bit of z , while $z[t_0, t_1, t_2, \dots, t_{n-1}]_b$ are the n bits of z as $z[t_i]_b$, $0 \leq t \leq n-1$. Similarly, $z[t]_n$ is the t -th nibble of z , while $z[t_0, t_1, \dots, t_{n-1}]_n$ are the n nibbles of $z[t_i]_n$, $0 \leq i \leq n-1$. When t_0, \dots, t_{n-1} are consecutive, we also write these n bits and nibbles as $z[t_0 \sim t_{n-1}]_b$ and $z[t_0 \sim t_{n-1}]_n$, respectively. As for the discussions on linear cryptanalysis, the linear masks of a state $z \in \{x, x', y, k\}$ are denoted by $\Gamma(z)$.

6.1 Differential cryptanalysis

In this subsection, we analyze the security of uKNIT-BC against differential attacks [24]. Unlike typical ciphers, uKNIT-BC is fully non-aligned as all rounds are very different, so we should check the security of all windows for $\mathcal{W}(i, r)$,

Table 1: Overview of our main key-recovery attacks on round-reduced uKNIT-BC, for which the full number of rounds is 12.

Attack	Rounds	Targeted Win.	Time	Data	Reference
Differential	10	$\mathcal{W}(0, 10)$	2^{78} Enc.	2^{56} CP	Section 6.1
Impossible differential	10	$\mathcal{W}(1, 10)$	2^{93} Enc.	2^{55} CP	Appendix G.2
Demirci-Selguk MitM	9	$\mathcal{W}(0, 9)$	2^{87} Enc.	2^{61} CP	Appendix G.5

$1 \leq r \leq 12$ and $0 \leq i \leq 12 - r$. With our SAT tool, we search for differential characteristics (DCs) for all windows and the results are shown in Table 2 (left).

Table 2: Differential probabilities (left) and linear correlations (right), in $-\log_2$, for $\mathcal{W}(i, r)$, $1 \leq r \leq 12, 0 \leq i \leq 12 - r$. The last column gives the value for the r -round PRINCE (both v1 and v2) where these r rounds contain $\lfloor (r - 2)/2 \rfloor$ forward rounds, $\lceil (r - 2)/2 \rceil$ backward rounds and the middle 2 rounds.

$r \backslash i$	0	1	2	3	4	5	6	7	8	9	10	11	PRINCE
1	2	2	2	2	2	2	2	2	2	2	2	2	-
2	8	8	6	6	8	8	6	8	8	6	8	-	-
3	14	12	12	12	14	14	12	14	12	12	-	-	-
4	25	23	24	26	30	26	26	24	24	-	-	-	32
5	40	40	39	40	40	39	37	37	-	-	-	-	39
6	49	48	46	46	50	47	49	-	-	-	-	-	44
7	60	58	52	61	60	59	-	-	-	-	-	-	56
8	71	70	68	71	72	-	-	-	-	-	-	-	66
9	81	82	80	82	-	-	-	-	-	-	-	-	74
10	94	87	92	-	-	-	-	-	-	-	-	-	80
11	101	99	-	-	-	-	-	-	-	-	-	-	89
12	113	-	-	-	-	-	-	-	-	-	-	-	99

$r \backslash i$	0	1	2	3	4	5	6	7	8	9	10	11	PRINCE
1	1	1	1	1	1	1	1	1	1	1	1	1	-
2	4	4	3	3	4	4	3	4	4	3	4	-	-
3	7	6	6	6	7	6	6	7	6	6	-	-	-
4	13	10	11	13	14	12	12	11	12	-	-	-	16
5	19	18	19	19	19	18	17	17	-	-	-	-	19
6	24	23	22	23	25	23	21	-	-	-	-	-	22
7	29	26	26	30	29	27	-	-	-	-	-	-	27
8	35	34	34	34	34	-	-	-	-	-	-	-	32
9	39	38	37	39	-	-	-	-	-	-	-	-	34
10	45	44	43	-	-	-	-	-	-	-	-	-	38
11	49	50	-	-	-	-	-	-	-	-	-	-	41
12	55	-	-	-	-	-	-	-	-	-	-	-	49

Remark. When $r \leq 5$, the differential probabilities of $\mathcal{W}(i, r)$ are higher than the r -round PRINCE ones. However, for $r \geq 6$, the differential probabilities of all r -length windows are lower. This shows that the components of uKNIT-BC are weaker in security than those of PRINCE, but for a higher number of rounds, the freedom to combine different components allowed uKNIT-BC to achieve stronger differential security. Generally speaking, a weaker component goes along with a better latency performance, which helps to explain to some extent why uKNIT-BC is better than PRINCE in both the security and latency criteria.

For 8-length windows, all probabilities are lower than 2^{-64} , so we expect that 8-round uKNIT-BC does not have any useful DCs. Thus, we choose to use a DC of $\mathcal{W}(2, 7)$ to mount a differential key-recovery attack. With an automatic search model that can consider both the distinguisher and the key-recovery process (see Appendix D), we check $\mathcal{W}(0 \leq i \leq 1, 11)$ with 7-round DC and 2-round backward and forward propagations. However, no effective key-recovery attack was detected. We then check $\mathcal{W}(0 \leq i \leq 2, 10)$ with 7-round DC with r_0 -round backward propagation and r_2 -round forward propagation, where $(r_0, r_2) \in \{(2, 1), (1, 2)\}$ and manage to detect some useful key-recovery attacks. The best key-recovery attack is depicted in Figure 6 which works for $\mathcal{W}(0, 10)$ with 7-round DC, 2-round backward propagation and 1-round forward propagation. The key-recovery process is as follows:

1. A total of 120 key bits (consisting of $k_0[0 \sim 11]_n$, $k_1[0 \sim 2, 4, 5, 7, 8, 10, 11]_n$, and $k_{10}[0, 2, 3, 7 \sim 10, 12, 15]_n$) are involved in the backward and forward propagations. Thus, we initialize a counter \mathbb{H} with 2^{120} entries.

2. One plaintext structure contains 2^{48} plaintexts. Encrypt these plaintexts and combine them to 2^{95} pairs. Since the ciphertext difference has 7 inactive S-boxes, thus only $2^{95-28} = 2^{67}$ pairs survive.
3. For each surviving pairs, we guess all 2^{48} values of $\Delta(x_1[0 \sim 2, 4, 5, 7, 8, 10, 11]_n)$. The possible keys of $k_1[0 \sim 2, 4, 5, 7, 8, 10, 11]$ and $k_0[0 \sim 11]$ can be extracted with some table-lookups for the active S-boxes. With $\Delta(x_{10})$ and $\Delta(x_9)$, the 36 bits of $k_{10}[0, 2, 3, 7 \sim 10, 12, 15]$ can be obtained by table-lookups, too.
4. k_0 and k_1 are directly related to the master key. Thus, the remaining 44 bits of the master key can be searched exhaustively.

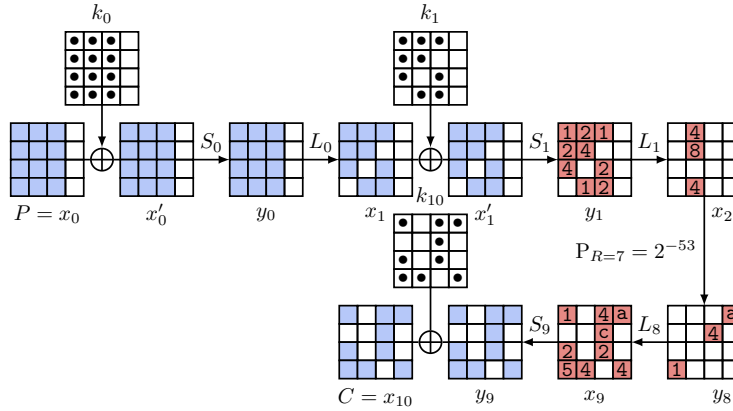


Fig. 6: The differential key-recovery attack on $\mathcal{W}(0, 10)$ based on a DC of $\mathcal{W}(2, 7)$ (red color) whose probability is 2^{-53} .

Complexity. For each plaintexts pair described above, the probability that it can propagate to $\Delta(y_1)$ is 2^{-48} . Thus, to make one *right pair* to appear, we need about $2^{(48+53)} = 2^{101}$ pairs due to the 2^{-53} -probability DC. In other words, $2^{101-95} = 2^6$ structures yield one right pair. To use 4 right pairs, we need 2^8 structures. Since the counter is as large as 2^{120} rows, the average number of times where the wrong keys are hit is significantly smaller than 1. Thus, the signal-to-noise rate is overwhelmingly large. Therefore, the data complexity is $2^{48+8} = 2^{56}$ plaintexts. The time complexity is to process the $2^{67+8} = 2^{75}$ pairs. For each pair, we need to do 30 table-lookups for extracting the keys, thus the time complexity is about 2^{80} memory accesses. We regard each memory access as one S-box evaluation, so 2^{80} memory access can be scaled to $2^{72.7}$ 10-round uKNIT-BC encryptions. Besides, for each pair, on average $2^{120-64-53} = 2^3$ pairs would be suggested, so we need to access \mathbb{H} 8 times. Considering \mathbb{H} is a large table, we regard one access to \mathbb{H} as one encryption, thus the time complexity of accessing \mathbb{H} is about 2^{78} encryptions, which dominates the time complexity. The memory complexity is dominated by the counter with 2^{120} rows. Since we expect

4 right pairs, 4 bits for each row should be sufficient. The memory complexity can be scaled to 2^{116} 64-bit blocks.

We also considered the effect of clustering and multiple differential cryptanalysis in Appendix G.1. We show that they should not be a threat for uKNIT-BC.

6.2 Linear (hull) cryptanalysis

Linear cryptanalysis [84] is another fundamental attack to check the security of a cipher. Akin to the differential attack, we use our SAT models to check for the linear correlations of all possible windows of uKNIT-BC. The results for all linear characteristics (LCs) are shown in Table 2 (right). Similar observations are found here, when $r \leq 7$, the correlation of $\mathcal{W}(i, r), 0 \leq i \leq 12 - r$ can be larger than those of PRINCE. While for $r \geq 8$, the correlations of all $\mathcal{W}(i, r)$ will be smaller than PRINCE, which again shows the effectiveness of uKNIT strategy – weaker components can be combined to design ciphers that have strong security without sacrificing the latency.

We use a model to search for good linear key-recovery attacks similarly to the differential key-recovery process. The best attack with a single LC works for $\mathcal{W}(0, 10)$ consisting of a distinguisher of correlation 2^{-30} for $\mathcal{W}(2, 7)$ and a 2-round backward propagation involving 96-bit keys and 1-round forward propagation involving 24-bit keys. Since the squared correlation of its distinguisher is significantly smaller than the 7-round DC we used in Section 6.1, we believe the linear attacks based on one LC cannot be stronger than the differential attack.

To examine how much uKNIT-BC suffers from the linear hull effect, we use a similar method from Appendix G.1. The main target is still $\mathcal{W}(2, 8)$ as it is in the middle which is the most crucial chunk in a possible key-recovery attack. We searched for 100 LCs for $\mathcal{W}(2, 8)$ whose correlation is exactly 2^{-34} . These LCs belong to 7 different linear truncated patterns (LTPs). The LTPs share a similar definition with DTPs but it treats linear masks rather than the differences. Also, when clustering LCs, we use the squared correlations to build the matrix M^{S_i} to avoid the cancellations of positive and negative correlations. Technically, we consider the expected linear potential of a linear hull.

Definition 4 (Expected linear potential (ELP) [92]). *Let $\Gamma(x_i)$ and $\Gamma(x_{i+r})$ be the input and output masks of an r -round key-alternating cipher (note that uKNIT-BC is key-alternating). The expected linear potential is defined as*

$$\text{ELP}(\Gamma(x_i), \Gamma(x_{i+r})) = \sum_{\Gamma(x_{i+1}), \dots, \Gamma(x_{i+r-1})} \text{C}^2((\Gamma(x_i), \Gamma(x_{i+1}), \dots, \Gamma(x_{i+r})))$$

where $\text{C}(\Gamma(x_i), \Gamma(x_{i+1}), \dots, \Gamma(x_{i+r}))$ is the correlation of a LC.

Among the 7 LTPs, the ELP of the best linear hull for $\mathcal{W}(2, 8)$ we find is $2^{-62.4}$, denoted by

$$(0, 0, 0, 0, \mathbf{b}, 4, 0, \mathbf{e}, 0, 0, 0, 0, 0, 0, 0, 0) \xrightarrow{\mathcal{W}(2, 8)} (0, 2, 0, 4, 2, 0, 2, 0, 0, 0, 0, 1, 8, 0, 0, 0)$$

This linear hull might be used in a key-recovery attack on $\mathcal{W}(0,10)$. However, due to the extremely high ELP, we believe it should be difficult to be used in the key-recovery attack considering the data limit we put on `uKNIT-BC`.

7 Implementations and benchmarks

General Comparison. We synthesized the different ciphers for the TSMC 65nm technology library using design compiler. For each cipher, we derived two measurements: the first one is a combinational unconstrained circuit as a baseline. In the second, For the second result, we place each cipher between input and output registers and set the target clock period to *1ns*. For all ciphers, this goal is unattainable. However, the synthesis algorithm outputs the closest netlist it could reach to this goal, as the expense of larger area. In both cases, we use the `compile_ultra` command. For `uKNIT-BC`, we also included the results of the circuit synthesized for the maximum achievable frequency of `PRINCEv2`. While `uKNIT-BC` is competitive even at its own maximum achievable frequency, it does not provide a fair comparison as `PRINCEv2` cannot even reach that frequency. Thus, it is a more fair comparison to also include the area gain when only the maximum achievable frequency of `PRINCEv2` is targeted. We also include the results with key schedule and with precomputed round keys (key side loading). The results are given in Table 3. In the comparison, we included most of the ciphers that claim to be designed with low latency in mind. However, they satisfy different security goals. `KoalaP` is a public permutation used in a Variable-Input-Length PRF (VIL-PRF), which takes many blocks and the latency is for how fast each block is absorbed. If it is used with a fixed 64-bit input length, it requires two calls to the permutation, doubling the latency, making it slower than `uKNIT-BC`. `Gleeok128` and `Orthros` are Fixed-Input-Length PRFs (FIL-PRFs). They are not designed to resist the same type of attacks as `uKNIT-BC`, so they can tolerate having lower latency for 128-bit blocks. That being said, from a pure latency point of view, we do in fact have lower latency, and in Appendix H we discuss how to build a PRF from `uKNIT-BC`, not even considering using less rounds. `BipBip` has a 24-bit block, yet its latency is not that much better than `uKNIT-BC` (less than 3%). `SPEEDY` and `Qarmav1` are TBCs with large block size and are both slower than `uKNIT-BC`. Last but not least, the candidate that is closest to `uKNIT-BC` in terms of security model and block size is `PRINCEv2`. `uKNIT-BC` has $\sim 10\%$ lower latency, and for the lowest latency of `PRINCEv2`, we need only half the area with precomputed round keys of $\sim 18\%$ less area with key schedule. For the remainder of this section, we provide more in depth benchmarking compared exclusively to `PRINCEv2`.

Power Consumption. We believe a straightforward comparison of different circuits synthesized for different targets, then setting the frequency to a nominal value, *e.g.* 10 MHz, is not very informative. This is especially true if the circuit used in the comparison is synthesized with maximum achievable frequency in mind. The synthesizer can allocate a very large area to achieve such frequency

Table 3: Comparison of different low latency primitives using TSMC 65nm. The first row of each cipher is the unconstrained combinational circuit, and the second row is the single cycle circuit with target latency of $1ns$. For **KoalaP**, the first row is just the permutation, while the second row is the latency is the time the PRF takes to absorb 1 64-bit block per cycle. For the VIL-PRF latency of **KoalaP**, we need to multiply the permutation latency by $1 + n_b$ where n_b is the number of input 64-bit blocks per 257-bit output block.

	Name	Block Size	Latency (ns)	Area (μm^2)
FIL-PRF	Gleeok128 [3]	128	3.45	73,078.92
		128	1.61	133,343.99
	Orthros [7]	128	2.66	40,932.36
		128	1.59	77,437.08
TBC	BipBip [12]	24	4.03	39,278.52
		24	1.45	60,630.12
	SPEEDY 7 rnds [78]	192	3.75	46,826.64
		192	1.79	88,331.04
	Qarmav1 9 rnds [4]	128	4.84	42,787.08
		128	2.74	94,944.23
Public Perm.	KoalaP [2]	64	1.46	24,104.88
		64	1.16	52,965.36
BC	PRINCEv2 [36]	64	2.90	12,006.72
		64	1.65	27,564.12
	uKNIT-BC (with side loading)	64	2.58	10,685.88
		64	1.64	14,587.92
		64	1.49	21,779.27
	uKNIT-BC	64	2.53	15,859.80
		64	1.64	22,963.67
		64	1.48	30,436.20

which increases power consumption. Thus, estimating the power of this particular circuit at a much lower frequency gives an inaccurate picture of how the cipher is used. Rather, if we target a low frequency, we should estimate the power using a circuit that is optimized either for area or for power consumption, not for frequency. Besides, comparing the power consumption of ciphers with different block sizes is also not very informative. Thus, we compare our proposal to PRINCEv2 and we follow a different approach. We synthesize each cipher for specific frequencies, and use the circuit corresponding to each frequency to estimate the power consumption. In case of the maximum achievable frequency for PRINCEv2, we synthesize uKNIT-BC for the same frequency and set the circuit of PRINCEv2 at that frequency. In case of the maximum achievable frequency for uKNIT-BC, we set the frequency of that circuit as such. Besides, we set the switching activity of the key bits to 0. The key is loaded once and used to encrypt many blocks. Thus, it is unreasonable to assume that the key values, and key schedule gates, are toggling all the time. It is a standard practice to simulate real operating conditions to estimate and optimize power consumption. The

results are presented in Table 4 (top). The results indeed show the subtleties of comparing power consumption.

Table 4: Power consumption and area at different frequencies.

Power consumption (in mW) using TSMC 65nm

Frequency	675 MHz	606 MHz	500 MHz	250 MHz	10 MHz
PRINCEv2	Infeasible	17.60	7.49	3.02	0.43
uKNIT-BC (SL)	13.73	8.02	4.92	2.22	0.34
Gain	∞	54.4%	34.3%	26.4%	20.9%
uKNIT-BC	18.15	9.83	6.08	2.76	0.50
Gain	∞	44.1%	18.8%	8.6%	-16.3%

Power consumption (in mW) using FDSOI 28nm 1.1V

Frequency	1.07 GHz	1 GHz	961 MHz	500 MHz	250 MHz	10 MHz
PRINCEv2	Infeasible	Infeasible	15.47	3.41	1.75	0.18
uKNIT-BC (SL)	13.95	7.19	6.07	2.58	1.33	0.14
Gain	∞	∞	60.8%	24.3%	24.0%	22.2%
uKNIT-BC	14.39	8.39	6.72	2.93	1.53	0.21
Gain	∞	∞	56.6%	14.1%	12.6%	-16.6%

Area (in μm^2) using FDSOI 28nm 1.1V

Frequency	1.07 GHz	1 GHz	961 MHz	10-500 MHz
PRINCEv2	Infeasible	Infeasible	9968.42	4777.51
uKNIT-BC (SL)	8015.40	4750.10	4297.06	3744.29
uKNIT-BC	10939.79	7635.64	6905.32	6078.71

In the medium range of frequency (250 ~ 500 MHz), uKNIT-BC experiences approximately 25% ~ 35% improvement without the key schedule and approximately 8% ~ 19% with key schedule. At PRINCEv2 maximum achievable frequency, uKNIT-BC reaches 44% and 55% improvement over PRINCEv2 with and without the key schedule, respectively. This massive gain is attributed to the fact that uKNIT-BC reaches this frequency without extensive optimization, while PRINCEv2 requires very high optimization effort and enlarged area. This is also why we observe a widening gap between the two ciphers, until we reach a range of frequencies for which PRINCEv2 is unsynthesizable. The power consumption of uKNIT-BC at maximum achievable frequency with key schedule is only marginally higher than PRINCEv2 at the maximum achievable frequency. Due to the frequency difference, this implies uKNIT-BC consumes 7.5% less energy. At low frequencies, uKNIT-BC consumes 20.9% less power without key schedule, but has higher consumption with key schedule due to the effects of leakage power consumption in the gates of the key schedule.

FDSOI 28nm. As further verification of our analysis, we replicated our results for PRINCEv2 and uKNIT-BC (with and without key schedule) using the FDSOI

28nm 1.1V standard cell library. Table 4 (mid) includes the power results, while Table 4 (bottom) includes the area results. We observe the same trends, and even higher gains in terms of power consumption as the latency is increased. In fact, uKNIT-BC can reach the GHz range, while PRINCEv2 cannot.

8 Conclusion

In this paper, we introduced a new design framework called uKNIT, aimed at providing guidelines for generating ciphers with various objectives and breaking for the first time the round alignment that prevailed. To showcase the strength of this framework, we instantiated a specific example focused on creating an ultra-low latency cipher, uKNIT-BC. It outperforms the state-of-the-art low-latency cipher (PRINCEv2) in both security and performance. Additionally, we generalized the STK construction for key scheduling, enabling its application to bit-oriented ciphers. To the best of our knowledge, this is also the first attempt at fully automating the cipher design process. Automation enables us to explore uncharted territories more effectively, paving the way for innovative cryptographic designs.

Acknowledgments

We thank the anonymous reviewers for the relevant comments on the first version of our paper. This research was supported by the Seagate Technology LLC grant “Low-Latency Lightweight Cryptography” and the Singapore NRF Investigatorship grant NRF-NRFI08-2022-0013. The second author is funded by the Wallenberg-NTU Presidential Postdoctoral Fellowship.

References

1. Albrecht, M.R., Rechberger, C., Schneider, T., Tiessen, T., Zohner, M.: Ciphers for MPC and FHE. In: Oswald, E., Fischlin, M. (eds.) *Advances in Cryptology - EUROCRYPT 2015*. LNCS, vol. 9056, pp. 430–454. Springer (2015), https://doi.org/10.1007/978-3-662-46800-5_17
2. Amiri-Eliasi, P., Belkheyar, Y., Daemen, J., Ghosh, S., Kuijsters, D., Mehrdad, A., Mella, S., Rasoolzadeh, S., Assche, G.V.: Koala: A Low-Latency Pseudorandom Function. *IACR Cryptol. ePrint Arch.* p. 1249 (2024), <https://eprint.iacr.org/2024/1249>
3. Anand, R., Banik, S., Caforio, A., Ishikawa, T., Isobe, T., Liu, F., Minematsu, K., Rahman, M., Sakamoto, K.: Gleeok: A Family of Low-Latency PRFs and its Applications to Authenticated Encryption. *IACR Trans. Cryptogr. Hardw. Embed. Syst.* 2024(2), 545–587 (2024), <https://doi.org/10.46586/tches.v2024.i2.545-587>
4. Avanzi, R.: The QARMA Block Cipher Family. Almost MDS Matrices Over Rings With Zero Divisors, Nearly Symmetric Even-Mansour Constructions With Non-Involutory Central Rounds, and Search Heuristics for Low-Latency S-Boxes. *IACR Trans. Symmetric Cryptol.* 2017(1), 4–44 (2017), <https://doi.org/10.13154/tosc.v2017.i1.4-44>

5. Avanzi, R., Banik, S., Dunkelman, O., Eichlseder, M., Ghosh, S., Nageler, M., Regazzoni, F.: The QARMAv2 Family of Tweakable Block Ciphers. *IACR Trans. Symmetric Cryptol.* 2023(3), 25–73 (2023), <https://doi.org/10.46586/tosc.v2023.i3.25-73>
6. Avanzi, R., Dunkelman, O., Minematsu, K.: MATTER: A Wide-Block Tweakable Block Cipher. *IACR Cryptol. ePrint Arch.* p. 1186 (2024), <https://eprint.iacr.org/2024/1186>
7. Banik, S., Isobe, T., Liu, F., Minematsu, K., Sakamoto, K.: Orthros: A Low-Latency PRF. *IACR Cryptol. ePrint Arch.* p. 390 (2021), <https://eprint.iacr.org/2021/390>
8. Bar-On, A., Dunkelman, O., Keller, N., Weizman, A.: DLCT: A New Tool for Differential-Linear Cryptanalysis. In: Ishai, Y., Rijmen, V. (eds.) *Advances in Cryptology - EUROCRYPT 2019*. LNCS, vol. 11476, pp. 313–342. Springer (2019), https://doi.org/10.1007/978-3-030-17653-2_11
9. Beierle, C., Biryukov, A., dos Santos, L.C., Großschädl, J., Perrin, L., Udovenko, A., Velichkov, V., Wang, Q.: Alzette: A 64-Bit ARX-box - (Feat. CRAX and TRAX). In: Micciancio, D., Ristenpart, T. (eds.) *Advances in Cryptology - CRYPTO 2020*. LNCS, vol. 12172, pp. 419–448. Springer (2020), https://doi.org/10.1007/978-3-030-56877-1_15
10. Beierle, C., Canteaut, A., Leander, G., Rotella, Y.: Proving Resistance Against Invariant Attacks: How to Choose the Round Constants. In: Katz, J., Shacham, H. (eds.) *Advances in Cryptology - CRYPTO 2017*. LNCS, vol. 10402, pp. 647–678. Springer (2017), https://doi.org/10.1007/978-3-319-63715-0_22
11. Beierle, C., Jean, J., Kölbl, S., Leander, G., Moradi, A., Peyrin, T., Sasaki, Y., Sasdrich, P., Sim, S.M.: The SKINNY Family of Block Ciphers and Its Low-Latency Variant MANTIS. In: Robshaw, M., Katz, J. (eds.) *Advances in Cryptology - CRYPTO 2016*. LNCS, vol. 9815, pp. 123–153. Springer (2016), https://doi.org/10.1007/978-3-662-53008-5_5
12. Belkheyyar, Y., Daemen, J., Dobraunig, C., Ghosh, S., Rasoolzadeh, S.: BipBip: A Low-Latency Tweakable Block Cipher with Small Dimensions. *IACR Trans. Cryptogr. Hardw. Embed. Syst.* 2023(1), 326–368 (2023), <https://doi.org/10.46586/tches.v2023.i1.326-368>
13. Belkheyyar, Y., Daemen, J., Dobraunig, C., Ghosh, S., Rasoolzadeh, S.: Introducing two Low-Latency Cipher Families: Sonic and SuperSonic. *IACR Cryptol. ePrint Arch.* p. 878 (2023), <https://eprint.iacr.org/2023/878>
14. Bellini, E., Gérard, D., Grados, J., Makarim, R.H., Peyrin, T.: Boosting Differential-Linear Cryptanalysis of ChaCha7 with MILP. *IACR Trans. Symmetric Cryptol.* 2023(2), 189–223 (2023), <https://doi.org/10.46586/tosc.v2023.i2.189-223>
15. Bellini, E., Gérard, D., Grados, J., Makarim, R.H., Peyrin, T.: Fully Automated Differential-Linear Attacks Against ARX Ciphers. In: Rosulek, M. (ed.) *Topics in Cryptology - CT-RSA 2023*. LNCS, vol. 13871, pp. 252–276. Springer (2023), https://doi.org/10.1007/978-3-031-30872-7_10
16. Beyne, T.: Block Cipher Invariants as Eigenvectors of Correlation Matrices. In: Peyrin, T., Galbraith, S.D. (eds.) *Advances in Cryptology - ASIACRYPT 2018*. LNCS, vol. 11272, pp. 3–31. Springer (2018), https://doi.org/10.1007/978-3-030-03326-2_1
17. Beyne, T., Neyt, A.: Note on the cryptanalysis of Speedy. *Cryptology ePrint Archive, Paper 2024/262* (2024), <https://eprint.iacr.org/2024/262>

18. Beyne, T., Verbauwhede, M.: Integral Cryptanalysis Using Algebraic Transition Matrices. *IACR Trans. Symmetric Cryptol.* 2023(4), 244–269 (2023), <https://doi.org/10.46586/tosc.v2023.i4.244-269>
19. Biere, A., Faller, T., Fazekas, K., Fleury, M., Froyloks, N., Pollitt, F.: CaDiCaL 2.0. In: Gurfinkel, A., Ganesh, V. (eds.) *Computer Aided Verification - CAV 2024*. LNCS, vol. 14681, pp. 133–152. Springer (2024)
20. Biham, E., Biryukov, A., Shamir, A.: Cryptanalysis of Skipjack Reduced to 31 Rounds Using Impossible Differentials. In: Stern, J. (ed.) *Advances in Cryptology - EUROCRYPT '99*. LNCS, vol. 1592, pp. 12–23. Springer (1999), https://doi.org/10.1007/3-540-48910-X_2
21. Biham, E., Dunkelman, O., Keller, N.: The Rectangle Attack - Rectangling the Serpent. In: Pfitzmann, B. (ed.) *Advances in Cryptology - EUROCRYPT 2001*. LNCS, vol. 2045, pp. 340–357. Springer (2001), https://doi.org/10.1007/3-540-44987-6_21
22. Biham, E., Dunkelman, O., Keller, N.: New Results on Boomerang and Rectangle Attacks. In: Daemen, J., Rijmen, V. (eds.) *Fast Software Encryption - FSE 2002*. LNCS, vol. 2365, pp. 1–16. Springer (2002), https://doi.org/10.1007/3-540-45661-9_1
23. Biham, E., Dunkelman, O., Keller, N.: New Combined Attacks on Block Ciphers. In: Gilbert, H., Handschuh, H. (eds.) *Fast Software Encryption - FSE 2005*. LNCS, vol. 3557, pp. 126–144. Springer (2005), https://doi.org/10.1007/11502760_9
24. Biham, E., Shamir, A.: Differential Cryptanalysis of DES-like Cryptosystems. In: Menezes, A., Vanstone, S.A. (eds.) *Advances in Cryptology - CRYPTO '90*. LNCS, vol. 537, pp. 2–21. Springer (1990), https://doi.org/10.1007/3-540-38424-3_1
25. Biryukov, A., Shamir, A.: Structural Cryptanalysis of SASAS. In: Pfitzmann, B. (ed.) *Advances in Cryptology - EUROCRYPT 2001*. LNCS, vol. 2045, pp. 394–405. Springer (2001), https://doi.org/10.1007/3-540-44987-6_24
26. Biryukov, A., Wagner, D.A.: Slide Attacks. In: Knudsen, L.R. (ed.) *Fast Software Encryption - FSE '99*. LNCS, vol. 1636, pp. 245–259. Springer (1999), https://doi.org/10.1007/3-540-48519-8_18
27. Biryukov, A., Wagner, D.A.: Advanced Slide Attacks. In: Preneel, B. (ed.) *Advances in Cryptology - EUROCRYPT 2000*. LNCS, vol. 1807, pp. 589–606. Springer (2000), https://doi.org/10.1007/3-540-45539-6_41
28. Blondeau, C., Bogdanov, A., Wang, M.: On the (In)Equivalence of Impossible Differential and Zero-Correlation Distinguishers for Feistel- and Skipjack-Type Ciphers. In: Boureanu, I., Owesarski, P., Vaudenay, S. (eds.) *Applied Cryptography and Network Security - ACNS 2014*. LNCS, vol. 8479, pp. 271–288. Springer (2014), https://doi.org/10.1007/978-3-319-07536-5_17
29. Bogdanov, A., Leander, G., Nyberg, K., Wang, M.: Integral and Multidimensional Linear Distinguishers with Correlation Zero. In: Wang, X., Sako, K. (eds.) *Advances in Cryptology - ASIACRYPT 2012*. LNCS, vol. 7658, pp. 244–261. Springer (2012), https://doi.org/10.1007/978-3-642-34961-4_16
30. Bogdanov, A., Rijmen, V.: Linear hulls with correlation zero and linear cryptanalysis of block ciphers. *Des. Codes Cryptogr.* 70(3), 369–383 (2014), <https://doi.org/10.1007/s10623-012-9697-z>
31. Bogdanov, A., Wang, M.: Zero Correlation Linear Cryptanalysis with Reduced Data Complexity. In: Canteaut, A. (ed.) *Fast Software Encryption - FSE 2012*. LNCS, vol. 7549, pp. 29–48. Springer (2012), https://doi.org/10.1007/978-3-642-34047-5_3

32. Borghoff, J., Canteaut, A., Güneysu, T., Kavun, E.B., Knezevic, M., Knudsen, L.R., Leander, G., Nikov, V., Paar, C., Rechberger, C., Rombouts, P., Thomsen, S.S., Yalçin, T.: PRINCE - A Low-Latency Block Cipher for Pervasive Computing Applications - Extended Abstract. In: Wang, X., Sako, K. (eds.) *Advances in Cryptology - ASIACRYPT 2012*. LNCS, vol. 7658, pp. 208–225. Springer (2012), https://doi.org/10.1007/978-3-642-34961-4_14
33. Boura, C., Canteaut, A.: Another View of the Division Property. In: Robshaw, M., Katz, J. (eds.) *Advances in Cryptology - CRYPTO 2016*. LNCS, vol. 9814, pp. 654–682. Springer (2016), https://doi.org/10.1007/978-3-662-53018-4_24
34. Boura, C., David, N., Boissier, R.H., Naya-Plasencia, M.: Better Steady than Speedy: Full Break of SPEEDY-7-192. In: Hazay, C., Stam, M. (eds.) *Advances in Cryptology - EUROCRYPT 2023*. LNCS, vol. 14007, pp. 36–66. Springer (2023), https://doi.org/10.1007/978-3-031-30634-1_2
35. Boura, C., Naya-Plasencia, M.: *Symmetric Cryptography, Volume 2: Cryptanalysis and Future Directions*. John Wiley & Sons (2024)
36. Bozilov, D., Eichlseder, M., Knezevic, M., Lambin, B., Leander, G., Moos, T., Nikov, V., Rasoolzadeh, S., Todo, Y., Wiemer, F.: PRINCEv2 - More Security for (Almost) No Overhead. In: Dunkelman, O., Jr., M.J.J., O’Flynn, C. (eds.) *Selected Areas in Cryptography - SAC 2020*. LNCS, vol. 12804, pp. 483–511. Springer (2020), https://doi.org/10.1007/978-3-030-81652-0_19
37. Canale, F., Güneysu, T., Leander, G., Thoma, J.P., Todo, Y., Ueno, R.: SCARF - A Low-Latency Block Cipher for Secure Cache-Randomization. In: Calandrino, J.A., Troncoso, C. (eds.) *USENIX Security Symposium, 2023*. pp. 1937–1954. USENIX Association (2023), <https://www.usenix.org/conference/usenixsecurity23/presentation/canale>
38. Canteaut, A., Fuhr, T., Gilbert, H., Naya-Plasencia, M., Reinhard, J.: Multiple Differential Cryptanalysis of Round-Reduced PRINCE. In: Cid, C., Rechberger, C. (eds.) *Fast Software Encryption - FSE 2014*. LNCS, vol. 8540, pp. 591–610. Springer (2014), https://doi.org/10.1007/978-3-662-46706-0_30
39. Chakraborty, D., Hadipour, H., Nguyen, P.H., Eichlseder, M.: Finding Complete Impossible Differential Attacks on AndRX Ciphers and Efficient Distinguishers for ARX Designs. *IACR Trans. Symmetric Cryptol.* 2024(3), 84–176 (2024), <https://doi.org/10.46586/tosc.v2024.i3.84-176>
40. Cid, C., Huang, T., Peyrin, T., Sasaki, Y., Song, L.: Boomerang Connectivity Table: A New Cryptanalysis Tool. In: Nielsen, J.B., Rijmen, V. (eds.) *Advances in Cryptology - EUROCRYPT 2018*. LNCS, vol. 10821, pp. 683–714. Springer (2018), https://doi.org/10.1007/978-3-319-78375-8_22
41. Cogliati, B., Jean, J., Peyrin, T., Seurin, Y.: A Long Tweak Goes a Long Way: High Multi-user Security Authenticated Encryption from Tweakable Block Ciphers. *IACR Communications in Cryptology* 1(2) (2024)
42. Cui, T., Jia, K., Fu, K., Chen, S., Wang, M.: New Automatic Search Tool for Impossible Differentials and Zero-Correlation Linear Approximations. *IACR Cryptol. ePrint Arch.* p. 689 (2016), <http://eprint.iacr.org/2016/689>
43. Daemen, J., Govaerts, R., Vandewalle, J.: Correlation Matrices. In: Preneel, B. (ed.) *Fast Software Encryption - FSE 1994*. LNCS, vol. 1008, pp. 275–285. Springer (1994), https://doi.org/10.1007/3-540-60590-8_21
44. Daemen, J., Knudsen, L.R., Rijmen, V.: The Block Cipher Square. In: Biham, E. (ed.) *Fast Software Encryption - FSE ’97*. LNCS, vol. 1267, pp. 149–165. Springer (1997), <https://doi.org/10.1007/BFb0052343>

45. Daemen, J., Rijmen, V.: The Wide Trail Design Strategy. In: Honary, B. (ed.) *Cryptography and Coding, 8th IMA International Conference*. LNCS, vol. 2260, pp. 222–238. Springer (2001), https://doi.org/10.1007/3-540-45325-3_20
46. Daemen, J., Rijmen, V.: *The Design of Rijndael: AES - The Advanced Encryption Standard*. Information Security and Cryptography, Springer (2002), <https://doi.org/10.1007/978-3-662-04722-4>
47. Dai, W., Hoang, V.T., Tessaro, S.: Information-Theoretic Indistinguishability via the Chi-Squared Method. In: Katz, J., Shacham, H. (eds.) *Advances in Cryptology - CRYPTO 2017*. LNCS, vol. 10403, pp. 497–523. Springer (2017), https://doi.org/10.1007/978-3-319-63697-9_17
48. Delobel, F., Derbez, P., Gontier, A., Rouquette, L., Solnon, C.: A CP-Based Automatic Tool for Instantiating Truncated Differential Characteristics. In: Chattopadhyay, A., Bhasin, S., Picek, S., Rebeiro, C. (eds.) *Progress in Cryptology - INDOCRYPT 2023*. LNCS, vol. 14459, pp. 247–268. Springer (2023), https://doi.org/10.1007/978-3-031-56232-7_12
49. Demirci, H., Selçuk, A.A.: A Meet-in-the-Middle Attack on 8-Round AES. In: Nyberg, K. (ed.) *Fast Software Encryption - FSE 2008*. LNCS, vol. 5086, pp. 116–126. Springer (2008), https://doi.org/10.1007/978-3-540-71039-4_7
50. Derbez, P., Fouque, P.: Exhausting Demirci-Selçuk Meet-in-the-Middle Attacks Against Reduced-Round AES. In: Moriai, S. (ed.) *Fast Software Encryption - FSE 2013*. LNCS, vol. 8424, pp. 541–560. Springer (2013), https://doi.org/10.1007/978-3-662-43933-3_28
51. Derbez, P., Fouque, P.: Automatic Search of Meet-in-the-Middle and Impossible Differential Attacks. In: Robshaw, M., Katz, J. (eds.) *Advances in Cryptology - CRYPTO 2016*. LNCS, vol. 9815, pp. 157–184. Springer (2016), https://doi.org/10.1007/978-3-662-53008-5_6
52. Derbez, P., Fouque, P., Jean, J.: Improved Key Recovery Attacks on Reduced-Round AES in the Single-Key Setting. In: Johansson, T., Nguyen, P.Q. (eds.) *Advances in Cryptology - EUROCRYPT 2013*. LNCS, vol. 7881, pp. 371–387. Springer (2013), https://doi.org/10.1007/978-3-642-38348-9_23
53. Dinur, I., Dunkelman, O., Shamir, A.: Collision Attacks on Up to 5 Rounds of SHA-3 Using Generalized Internal Differentials. In: Moriai, S. (ed.) *Fast Software Encryption - FSE 2013*. LNCS, vol. 8424, pp. 219–240. Springer (2013), https://doi.org/10.1007/978-3-662-43933-3_12
54. Dobraunig, C., Eichlseder, M., Grassi, L., Lallemand, V., Leander, G., List, E., Mendel, F., Rechberger, C.: Rasta: A cipher with low ANDdepth and few ANDs per bit. *IACR Cryptol. ePrint Arch.* p. 181 (2018), <http://eprint.iacr.org/2018/181>
55. Dobraunig, C., Eichlseder, M., Kales, D., Mendel, F.: Practical Key-Recovery Attack on MANTIS5. *IACR Trans. Symmetric Cryptol.* 2016(2), 248–260 (2016), <https://doi.org/10.13154/tosc.v2016.i2.248-260>
56. Dunkelman, O., Keller, N., Shamir, A.: Improved Single-Key Attacks on 8-Round AES-192 and AES-256. In: Abe, M. (ed.) *Advances in Cryptology - ASIACRYPT 2010*. LNCS, vol. 6477, pp. 158–176. Springer (2010), https://doi.org/10.1007/978-3-642-17373-8_10
57. Dunkelman, O., Keller, N., Shamir, A.: A Practical-Time Related-Key Attack on the KASUMI Cryptosystem Used in GSM and 3G Telephony. In: Rabin, T. (ed.) *Advances in Cryptology - CRYPTO 2010*. LNCS, vol. 6223, pp. 393–410. Springer (2010), https://doi.org/10.1007/978-3-642-14623-7_21

58. Dunkelman, O., Keller, N., Shamir, A.: A Practical-Time Related-Key Attack on the KASUMI Cryptosystem Used in GSM and 3G Telephony. *J. Cryptol.* 27(4), 824–849 (2014), <https://doi.org/10.1007/s00145-013-9154-9>
59. Dunkelman, O., Keller, N., Shamir, A.: Slidex Attacks on the Even-Mansour Encryption Scheme. *J. Cryptol.* 28(1), 1–28 (2015), <https://doi.org/10.1007/s00145-013-9164-7>
60. Gérard, D., Lafourcade, P., Minier, M., Solnon, C.: Computing AES related-key differential characteristics with constraint programming. *Artif. Intell.* 278 (2020), <https://doi.org/10.1016/j.artint.2019.103183>
61. Gorski, M., Lucks, S., Peyrin, T.: Slide Attacks on a Class of Hash Functions. In: Pieprzyk, J. (ed.) *Advances in Cryptology - ASIACRYPT 2008*. LNCS, vol. 5350, pp. 143–160. Springer (2008), https://doi.org/10.1007/978-3-540-89255-7_10
62. Grassi, L., Lüftenegger, R., Rechberger, C., Rotaru, D., Schafneger, M.: On a Generalization of Substitution-Permutation Networks: The HADES Design Strategy. In: Canteaut, A., Ishai, Y. (eds.) *Advances in Cryptology - EUROCRYPT 2020*. LNCS, vol. 12106, pp. 674–704. Springer (2020), https://doi.org/10.1007/978-3-030-45724-2_23
63. Greene, P., Motley, M., Weeks, B.: ARADI and LLAMA: Low-Latency Cryptography for Memory Encryption. *IACR Cryptol. ePrint Arch.* p. 1240 (2024), <https://eprint.iacr.org/2024/1240>
64. Hadipour, H., Eichlseder, M.: Integral Cryptanalysis of WARP based on Monomial Prediction. *IACR Trans. Symmetric Cryptol.* 2022(2), 92–112 (2022), <https://doi.org/10.46586/tosc.v2022.i2.92-112>
65. Hall, C., Wagner, D.A., Kelsey, J., Schneier, B.: Building PRFs from PRPs. In: Krawczyk, H. (ed.) *Advances in Cryptology - CRYPTO '98*. LNCS, vol. 1462, pp. 370–389. Springer (1998), <https://doi.org/10.1007/BFb0055742>
66. Hu, K., Peyrin, T., Tan, Q.Q., Yap, T.: Revisiting Higher-Order Differential-Linear Attacks from an Algebraic Perspective. In: Guo, J., Steinfeld, R. (eds.) *Advances in Cryptology - ASIACRYPT 2023*. LNCS, vol. 14440, pp. 405–435. Springer (2023), https://doi.org/10.1007/978-981-99-8727-6_14
67. Hu, K., Sun, S., Wang, M., Wang, Q.: An Algebraic Formulation of the Division Property: Revisiting Degree Evaluations, Cube Attacks, and Key-Independent Sums. In: Moriai, S., Wang, H. (eds.) *Advances in Cryptology - ASIACRYPT 2020*. LNCS, vol. 12491, pp. 446–476. Springer (2020), https://doi.org/10.1007/978-3-030-64837-4_15
68. Jean, J., Nikolic, I.: Efficient Design Strategies Based on the AES Round Function. In: Peyrin, T. (ed.) *Fast Software Encryption - FSE 2016*. LNCS, vol. 9783, pp. 334–353. Springer (2016), https://doi.org/10.1007/978-3-662-52993-5_17
69. Jean, J., Nikolić, I., Peyrin, T.: Tweaks and Keys for Block Ciphers: The TWEAKEY Framework. In: Sarkar, P., Iwata, T. (eds.) *Advances in Cryptology - ASIACRYPT 2014*. pp. 274–288. Springer Berlin Heidelberg, Berlin, Heidelberg (2014)
70. Jr., B.S.K., Robshaw, M.J.B.: Fast Block Cipher Proposal. In: Anderson, R.J. (ed.) *Fast Software Encryption - 1993*. LNCS, vol. 809, pp. 33–40. Springer (1993), https://doi.org/10.1007/3-540-58108-1_3
71. Kelsey, J., Kohno, T., Schneier, B.: Amplified Boomerang Attacks Against Reduced-Round MARS and Serpent. In: Schneier, B. (ed.) *Fast Software Encryption - FSE 2000*. LNCS, vol. 1978, pp. 75–93. Springer (2000), https://doi.org/10.1007/3-540-44706-7_6

72. Khairallah, M., Yadhunathan, S., Bhasin, S.: Lightweight Leakage-Resilient PRNG from TBCs Using Superposition. In: Wacquez, R., Homma, N. (eds.) *Constructive Side-Channel Analysis and Secure Design - COSADE 2024*. LNCS, vol. 14595, pp. 197–217. Springer (2024), https://doi.org/10.1007/978-3-031-57543-3_11
73. Knudsen, L.: DEAL-a 128-bit block cipher. *complexity* 258(2), 216 (1998)
74. Knudsen, L.R., Wagner, D.A.: Integral Cryptanalysis. In: Daemen, J., Rijmen, V. (eds.) *Fast Software Encryption - FSE 2002*. LNCS, vol. 2365, pp. 112–127. Springer (2002), https://doi.org/10.1007/3-540-45661-9_9
75. Kounavis, M.E., Deutsch, S., Ghosh, S., Durham, D.: K-Cipher: A Low Latency, Bit Length Parameterizable Cipher. In: *IEEE Symposium on Computers and Communications - ISCC 2020*. pp. 1–7. IEEE (2020), <https://doi.org/10.1109/ISCC50000.2020.9219582>
76. Langford, S.K., Hellman, M.E.: Differential-Linear Cryptanalysis. In: Desmedt, Y. (ed.) *Advances in Cryptology - CRYPTO '94*. LNCS, vol. 839, pp. 17–25. Springer (1994), https://doi.org/10.1007/3-540-48658-5_3
77. Leander, G., Abdelraheem, M.A., Alkhzaimi, H., Zenner, E.: A Cryptanalysis of PRINTcipher: The Invariant Subspace Attack. In: Rogaway, P. (ed.) *Advances in Cryptology - CRYPTO 2011*. LNCS, vol. 6841, pp. 206–221. Springer (2011), https://doi.org/10.1007/978-3-642-22792-9_12
78. Leander, G., Moos, T., Moradi, A., Rasoolzadeh, S.: The SPEEDY Family of Block Ciphers Engineering an Ultra Low-Latency Cipher from Gate Level for Secure Processor Architectures. *IACR Trans. Cryptogr. Hardw. Embed. Syst.* 2021(4), 510–545 (2021), <https://doi.org/10.46586/tches.v2021.i4.510-545>
79. Li, H., Zhang, H., Hu, K., Liu, G., Qiu, W.: AlgSAT - A SAT Method for Verification of Differential Trails from an Algebraic Perspective. In: Zhu, T., Li, Y. (eds.) *Information Security and Privacy - ACISP 2024*. LNCS, vol. 14895, pp. 450–471. Springer (2024), https://doi.org/10.1007/978-981-97-5025-2_23
80. Liu, M., Lu, X., Lin, D.: Differential-Linear Cryptanalysis from an Algebraic Perspective. In: Malkin, T., Peikert, C. (eds.) *Advances in Cryptology - CRYPTO 2021*. LNCS, vol. 12827, pp. 247–277. Springer (2021), https://doi.org/10.1007/978-3-030-84252-9_9
81. Liu, Y., Xiang, Z., Chen, S., Zhang, S., Zeng, X.: A Novel Automatic Technique Based on MILP to Search for Impossible Differentials. In: Tibouchi, M., Wang, X. (eds.) *Applied Cryptography and Network Security - ACNS 2023*. LNCS, vol. 13905, pp. 119–148. Springer (2023), https://doi.org/10.1007/978-3-031-33488-7_5
82. Lucks, S.: The Saturation Attack - A Bait for Twofish. In: Matsui, M. (ed.) *Fast Software Encryption - FSE 2001*. LNCS, vol. 2355, pp. 1–15. Springer (2001), https://doi.org/10.1007/3-540-45473-X_1
83. Mahzoun, M., Kraveva, L., Posteuca, R., Ashur, T.: Differential Cryptanalysis of K-Cipher. In: *IEEE Symposium on Computers and Communications - ISCC 2022*. pp. 1–7. IEEE (2022), <https://doi.org/10.1109/ISCC55528.2022.9912926>
84. Matsui, M.: Linear Cryptanalysis Method for DES Cipher. In: Helleseht, T. (ed.) *Advances in Cryptology - EUROCRYPT '93*. LNCS, vol. 765, pp. 386–397. Springer (1993), https://doi.org/10.1007/3-540-48285-7_33
85. Mouha, N., Preneel, B.: Towards Finding Optimal Differential Characteristics for ARX: Application to Salsa20. *Cryptology ePrint Archive, Paper 2013/328* (2013), <https://eprint.iacr.org/2013/328>

86. Mouha, N., Wang, Q., Gu, D., Preneel, B.: Differential and Linear Cryptanalysis Using Mixed-Integer Linear Programming. In: Wu, C., Yung, M., Lin, D. (eds.) Information Security and Cryptology - Inscrypt 2011. LNCS, vol. 7537, pp. 57–76. Springer (2011), https://doi.org/10.1007/978-3-642-34704-7_5
87. Murphy, S.: The Return of the Cryptographic Boomerang. *IEEE Trans. Inf. Theory* 57(4), 2517–2521 (2011), <https://doi.org/10.1109/TIT.2011.2111091>
88. Naito, Y., Sasaki, Y., Sugawara, T.: Lightweight Authenticated Encryption Mode Suitable for Threshold Implementation. In: Canteaut, A., Ishai, Y. (eds.) Advances in Cryptology - EUROCRYPT 2020. Lecture Notes in Computer Science, vol. 12106, pp. 705–735. Springer (2020), https://doi.org/10.1007/978-3-030-45724-2_24
89. Nikolic, I.: How to Use Metaheuristics for Design of Symmetric-Key Primitives. In: Takagi, T., Peyrin, T. (eds.) Advances in Cryptology - ASIACRYPT 2017. LNCS, vol. 10626, pp. 369–391. Springer (2017), https://doi.org/10.1007/978-3-319-70700-6_13
90. NIST: Submission requirements and evaluation criteria for the lightweight cryptography standardization process (2018), <https://csrc.nist.gov/CSRC/media/Projects/Lightweight-Cryptography/documents/final-lwc-submission-requirements-august2018.pdf>
91. NXP Semiconductors: AN12527 - LPC55Sxx PRINCE Real-Time Data Encryption. Application note (2023), <https://www.nxp.com/docs/en/application-note/AN12527.pdf>
92. Nyberg, K.: Linear Approximation of Block Ciphers. In: Santis, A.D. (ed.) Advances in Cryptology - EUROCRYPT '94. LNCS, vol. 950, pp. 439–444. Springer (1994), <https://doi.org/10.1007/BFb0053460>
93. Peyrin, T.: Improved Differential Attacks for ECHO and Grøstl. In: Rabin, T. (ed.) Advances in Cryptology - CRYPTO 2010. LNCS, vol. 6223, pp. 370–392. Springer (2010), https://doi.org/10.1007/978-3-642-14623-7_20
94. Qin, L., Dong, X., Wang, X., Jia, K., Liu, Y.: Automated Search Oriented to Key Recovery on Ciphers with Linear Key Schedule Applications to Boomerangs in SKINNY and ForkSkinny. *IACR Trans. Symmetric Cryptol.* 2021(2), 249–291 (2021), <https://doi.org/10.46586/tosc.v2021.i2.249-291>
95. Sakamoto, K., Liu, F., Nakano, Y., Kiyomoto, S., Isobe, T.: Rocca: An Efficient AES-based Encryption Scheme for Beyond 5G (Full version). *IACR Cryptol. ePrint Arch.* p. 116 (2022), <https://eprint.iacr.org/2022/116>
96. Sasaki, Y., Todo, Y.: New Impossible Differential Search Tool from Design and Cryptanalysis Aspects - Revealing Structural Properties of Several Ciphers. In: Coron, J., Nielsen, J.B. (eds.) Advances in Cryptology - EUROCRYPT 2017. LNCS, vol. 10212, pp. 185–215 (2017), https://doi.org/10.1007/978-3-319-56617-7_7
97. Shalan, M., Edwards, T.: Building OpenLANE: A 130nm OpenROAD-based Tapeout- Proven Flow : Invited Paper. In: 2020 IEEE/ACM International Conference On Computer Aided Design (ICCAD). pp. 1–6 (2020)
98. Shi, D., Sun, S., Derbez, P., Todo, Y., Sun, B., Hu, L.: Programming the Demirci-Selçuk Meet-in-the-Middle Attack with Constraints. In: Peyrin, T., Galbraith, S.D. (eds.) Advances in Cryptology - ASIACRYPT 2018. LNCS, vol. 11273, pp. 3–34. Springer (2018), https://doi.org/10.1007/978-3-030-03329-3_1
99. Shi, D., Sun, S., Song, L., Hu, L., Yang, Q.: Exploiting Non-full Key Additions: Full-Fledged Automatic Demirci-Selçuk Meet-in-the-Middle Cryptanalysis

- of SKINNY. In: Hazay, C., Stam, M. (eds.) *Advances in Cryptology - EUROCRYPT 2023*. LNCS, vol. 14007, pp. 67–97. Springer (2023), https://doi.org/10.1007/978-3-031-30634-1_3
100. Song, L., Yang, Q., Chen, Y., Hu, L., Weng, J.: Probabilistic Extensions: A One-Step Framework for Finding Rectangle Attacks and Beyond. In: Joye, M., Leander, G. (eds.) *Advances in Cryptology - EUROCRYPT 2024*. LNCS, vol. 14651, pp. 339–367. Springer (2024), https://doi.org/10.1007/978-3-031-58716-0_12
 101. Sun, B., Liu, Z., Rijmen, V., Li, R., Cheng, L., Wang, Q., Alkhzaimi, H., Li, C.: Links among Impossible Differential, Integral and Zero Correlation Linear Cryptanalysis. *IACR Cryptol. ePrint Arch.* p. 181 (2015), <http://eprint.iacr.org/2015/181>
 102. Sun, L., Wang, W., Wang, M.: Accelerating the Search of Differential and Linear Characteristics with the SAT Method. *IACR Trans. Symmetric Cryptol.* 2021(1), 269–315 (2021), <https://doi.org/10.46586/tosc.v2021.i1.269-315>
 103. Todo, Y.: Structural Evaluation by Generalized Integral Property. In: Oswald, E., Fischlin, M. (eds.) *Advances in Cryptology - EUROCRYPT 2015*. LNCS, vol. 9056, pp. 287–314. Springer (2015), https://doi.org/10.1007/978-3-662-46800-5_12
 104. Todo, Y., Leander, G., Sasaki, Y.: Nonlinear Invariant Attack - Practical Attack on Full SCREAM, iSCREAM, and Midori64. In: Cheon, J.H., Takagi, T. (eds.) *Advances in Cryptology - ASIACRYPT 2016*. LNCS, vol. 10032, pp. 3–33 (2016), https://doi.org/10.1007/978-3-662-53890-6_1
 105. Todo, Y., Morii, M.: Bit-Based Division Property and Application to Simon Family. In: Peyrin, T. (ed.) *Fast Software Encryption - FSE 2016*. LNCS, vol. 9783, pp. 357–377. Springer (2016), https://doi.org/10.1007/978-3-662-52993-5_18
 106. Tribbey, W.: *Numerical Recipes: The Art of Scientific Computing (3rd Edition)* is written by William H. Press, Saul A. Teukolsky, William T. Vetterling, and Brian P. Flannery, and published by Cambridge University Press, (c) 2007, hardback, ISBN 978-0-521-88068-8, 1235 pp. *ACM SIGSOFT Softw. Eng. Notes* 35(6), 30–31 (2010), <https://doi.org/10.1145/1874391.187410>
 107. Turan, M.S., McKay, K., Chang, D., Bassham, L., Kang, J., Waller, N., Kelsey, J., Hong, D.: Status Report on the Final Round of the NIST Lightweight Cryptography Standardization Process. NIST IR 8454 (2023), <https://doi.org/10.6028/NIST.IR.8454>
 108. Wagner, D.A.: The Boomerang Attack. In: Knudsen, L.R. (ed.) *Fast Software Encryption - FSE '99*. LNCS, vol. 1636, pp. 156–170. Springer (1999), https://doi.org/10.1007/3-540-48519-8_12
 109. Wang, D., Wang, B., Sun, S.: SAT-aided Automatic Search of Boomerang Distinguishers for ARX Ciphers. *IACR Trans. Symmetric Cryptol.* 2023(1), 152–191 (2023), <https://doi.org/10.46586/tosc.v2023.i1.152-191>
 110. Wang, J., Huang, T., Wu, S., Liu, Z.: Twinkle: A family of Low-latency Schemes for Authenticated Encryption and Pointer Authentication. *IACR Communications in Cryptology* 1(2) (2024)
 111. Wolf, C.: Yosys Open SYnthesis Suite. <https://yosyshq.net/yosys/>
 112. Xiang, Z., Zhang, W., Bao, Z., Lin, D.: Applying MILP Method to Searching Integral Distinguishers Based on Division Property for 6 Lightweight Block Ciphers. In: Cheon, J.H., Takagi, T. (eds.) *Advances in Cryptology - ASIACRYPT 2016*. LNCS, vol. 10031, pp. 648–678 (2016), https://doi.org/10.1007/978-3-662-53887-6_24

113. Yang, Q., Song, L., Zhang, N., Shi, D., Wang, L., Zhao, J., Hu, L., Weng, J.: Optimizing Rectangle and Boomerang Attacks: A Unified and Generic Framework for Key Recovery. *J. Cryptol.* 37(2), 19 (2024), <https://doi.org/10.1007/s00145-024-09499-1>
114. Zhang, L., Wu, R., Zhang, Y., Zheng, Y., Wu, W.: LLLWBC: A New Low-Latency Light-Weight Block Cipher. In: Deng, Y., Yung, M. (eds.) *Information Security and Cryptology - Inscrypt 2022*. LNCS, vol. 13837, pp. 23–42. Springer (2022), https://doi.org/10.1007/978-3-031-26553-2_2
115. Zhang, Z., Hou, C., Liu, M.: Collision Attacks on Round-Reduced SHA-3 Using Conditional Internal Differentials. In: Hazay, C., Stam, M. (eds.) *Advances in Cryptology - EUROCRYPT 2023*. LNCS, vol. 14007, pp. 220–251. Springer (2023), https://doi.org/10.1007/978-3-031-30634-1_8
116. Zhang, Z., Hou, C., Liu, M.: Probabilistic Linearization: Internal Differential Collisions in up to 6 Rounds of SHA-3. In: Reyzin, L., Stebila, D. (eds.) *Advances in Cryptology - CRYPTO 2024*. LNCS, vol. 14923, pp. 241–272. Springer (2024), https://doi.org/10.1007/978-3-031-68385-5_8

Supplementary Materials

A Algorithm for diversity computation between two ciphers

Algorithm 1 Diversity computation of two ciphers C and C'

Inputs:

$$C = S_0, P_0, S_1, P_1, \dots, P_{n-1}, S_n$$

$$C' = S'_0, P'_0, S'_1, P'_1, \dots, P'_{n-1}, S'_n$$

Outputs:

A value representing the difference between C and C'

$$SL \leftarrow S_0, S_1, \dots, S_n$$

$$SL' \leftarrow S'_0, S'_1, \dots, S'_n$$

$$\text{sCounter} \leftarrow [0, \dots, 0] \quad \triangleright \text{initialize the diversity for each subst layer}$$

for each substLayer index i from 0 to n **do**

for each S-box index j from 0 to 15 **do**

if $SL[i][j] \neq SL'[i][j]$ **then** $\text{sCounter}[i] = \text{sCounter}[i] + 1$

end if

end for

$$\text{sCounter}[i] = \text{sCounter}[i]/16 \quad \triangleright \text{Normalization}$$

end for

$$PL \leftarrow P_0, P_1, \dots, P_{n-1}$$

$$PL' \leftarrow P'_0, P'_1, \dots, P'_{n-1}$$

$$\text{pCounter} \leftarrow [0, \dots, 0] \quad \triangleright \text{initialize the diversity for each permutation layer}$$

for each linearLayer index i from 0 to $n - 1$ **do**

for each row index j from 0 to 63 **do**

for each col index k from 0 to 63 **do**

if $PL[i][j][k] \neq PL'[i][j][k]$ **then** $\text{pCounter}[i] = \text{pCounter}[i] + 1$

end if

end for

end for

$$\text{pCounter}[i] \leftarrow \text{pCounter}[i]/(64 \times 64) \quad \triangleright \text{Normalization}$$

end for

return $\text{sum}(\text{sCounter}) + \text{sum}(\text{pCounter})$

B uKNIT-BC components

B.1 uKNIT-BC substitution layers

The 4-bit S-boxes applied to each nibble of the internal state in uKNIT-BC can be represented by $(D \circ S_{\text{MANTIS}} \circ B)(x)$ where S_{MANTIS} is the MANTIS S-box ($S_{\text{MANTIS}}[x] = [c, a, d, 3, e, b, f, 7, 8, 9, 1, 5, 0, 2, 4, 6]$) and D and B are transposition matrices.

To represent the S-boxes succinctly, we use cycles. For example, $\begin{bmatrix} 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix}$ can be represented as (03)(12). The transposition matrices D and B are given in Table 5.

Table 5: The matrices D (top) and B (bottom) for all the S-boxes in each round.

S-box	Round											
	0	1	2	3	4	5	6	7	8	9	10	11
0	(021)(3) (0213)	(0)(132) (02)(1)(3)	(0231) (0)(132)	(0321) (0)(123)	(0231) (0)(12)(3)	(021)(3) (0)(12)(3)	(012)(3) (0)(1)(23)	(03)(1)(2) (02)(1)(3)	(021)(3) (0312)	(0123) (0)(13)(2)	(03)(1)(2) (0321)	(02)(13) (012)(3)
1	(0312) (012)(3)	(032)(1) (0312)	(0132) (0321)	(03)(12) (0)(1)(23)	(0)(1)(2)(3) (0132)	(01)(23) (023)(1)	(03)(12) (0)(132)	(0)(123) (013)(2)	(0231) (0123)	(0321) (03)(1)(2)	(012)(3) (023)(1)	(0132) (023)(1)
2	(023)(1) (013)(2)	(01)(23) (021)(3)	(03)(12) (0231)	(021)(3) (0)(123)	(032)(1) (02)(1)(3)	(0)(123) (01)(23)	(0132) (03)(12)	(0123) (0)(123)	(012)(3) (0)(132)	(02)(1)(3) (0)(1)(2)(3)	(0)(12)(3) (01)(2)(3)	(03)(1)(2) (03)(1)(2)
3	(0123) (0)(132)	(0)(132) (0213)	(0)(13)(2) (023)(1)	(023)(1) (0)(1)(2)(3)	(03)(1)(2) (0)(12)(3)	(0)(1)(23) (013)(2)	(01)(2)(3) (0321)	(02)(1)(3) (0123)	(03)(1)(2) (0213)	(021)(3) (0)(12)(3)	(03)(12) (0123)	(0132) (03)(1)(2)
4	(032)(1) (0)(1)(23)	(0312) (0213)	(0)(123) (0)(123)	(0)(1)(2)(3) (02)(1)(3)	(0)(13)(2) (02)(1)(3)	(02)(13) (0)(1)(23)	(03)(1)(2) (0132)	(0312) (03)(1)(2)	(031)(2) (0)(1)(23)	(031)(2) (0213)	(0)(132) (021)(3)	(0)(132) (0213)
5	(032)(1) (0231)	(0321) (013)(2)	(021)(3) (03)(12)	(02)(13) (031)(2)	(032)(1) (01)(2)(3)	(031)(2) (03)(12)	(0231) (03)(1)(2)	(02)(13) (0)(123)	(023)(1) (03)(1)(2)	(0123) (01)(2)(3)	(0132) (0)(123)	(0)(123) (031)(2)
6	(0132) (012)(3)	(03)(12) (0321)	(0321) (02)(1)(3)	(01)(23) (013)(2)	(0213) (0)(1)(2)(3)	(021)(3) (032)(1)	(02)(1)(3) (0)(123)	(03)(12) (0)(12)(3)	(012)(3) (031)(2)	(0)(123) (032)(1)	(032)(1) (012)(3)	(0)(13)(2) (02)(1)(3)
7	(032)(1) (0)(123)	(0312) (0)(13)(2)	(02)(1)(3) (0)(1)(23)	(01)(23) (013)(2)	(0)(123) (023)(1)	(0)(12)(3) (0)(123)	(02)(13) (013)(2)	(02)(13) (0123)	(0)(123) (0312)	(031)(2) (0132)	(02)(13) (013)(2)	(0)(12)(3) (0)(13)(2)
8	(0312) (0312)	(02)(1)(3) (02)(13)	(01)(2)(3) (03)(12)	(031)(2) (03)(12)	(0)(132) (0)(12)(3)	(013)(2) (0123)	(0321) (0)(1)(23)	(02)(13) (02)(1)(3)	(023)(1) (012)(3)	(012)(3) (0)(12)(3)	(0)(1)(2)(3) (0321)	(0231) (02)(13)
9	(0)(1)(23) (032)(1)	(0213) (03)(1)(2)	(0321) (0)(13)(2)	(031)(2) (013)(2)	(023)(1) (0231)	(02)(1)(3) (032)(1)	(01)(23) (01)(2)(3)	(01)(23) (023)(1)	(013)(2) (032)(1)	(02)(1)(3) (031)(2)	(0312) (031)(2)	(03)(12) (0123)
10	(0)(1)(2)(3) (01)(23)	(03)(12) (0)(132)	(0312) (0231)	(012)(3) (03)(12)	(02)(13) (0)(123)	(0213) (0)(1)(23)	(032)(1) (013)(2)	(021)(3) (03)(12)	(0231) (03)(1)(2)	(0231) (0)(12)(3)	(02)(1)(3) (0312)	(012)(3) (0)(1)(23)
11	(0321) (0)(13)(2)	(0231) (032)(1)	(0312) (0)(123)	(012)(3) (0)(132)	(0132) (032)(1)	(03)(12) (0231)	(0)(1)(2)(3) (03)(12)	(0123) (03)(12)	(0)(12)(3) (03)(12)	(0)(13)(2) (023)(1)	(0)(13)(2) (0321)	(013)(2) (0132)
12	(0123) (013)(2)	(02)(13) (0)(132)	(012)(3) (021)(3)	(02)(13) (0213)	(03)(12) (02)(13)	(0)(1)(23) (032)(1)	(03)(1)(2) (0123)	(01)(2)(3) (02)(1)(3)	(0)(132) (021)(3)	(012)(3) (0123)	(031)(2) (01)(2)(3)	(02)(13) (0)(12)(3)
13	(0)(1)(23) (0)(12)(3)	(031)(2) (0)(12)(3)	(032)(1) (02)(1)(3)	(0)(12)(3) (0312)	(0123) (012)(3)	(0)(132) (032)(1)	(032)(1) (03)(1)(2)	(012)(3) (0)(132)	(0312) (032)(1)	(03)(12) (0231)	(03)(1)(2) (0)(123)	(0312) (02)(13)
14	(0123) (0)(132)	(0123) (0)(123)	(0132) (01)(2)(3)	(0231) (0)(13)(2)	(0)(12)(3) (031)(2)	(0231) (023)(1)	(02)(1)(3) (0)(1)(23)	(0123) (0312)	(0)(1)(23) (0)(13)(2)	(0321) (0321)	(0)(12)(3) (0)(132)	(0)(13)(2) (0231)
15	(02)(13) (0)(1)(23)	(0)(13)(2) (031)(2)	(0132) (0132)	(0)(1)(23) (032)(1)	(0312) (0)(1)(23)	(02)(1)(3) (03)(1)(2)	(0231) (032)(1)	(0321) (0312)	(0)(123) (0123)	(0)(132) (02)(13)	(012)(3) (01)(2)(3)	(0)(132) (01)(23)

B.2 uKNIT-BC linear layers

The linear layers L_i of uKNIT-BC are very sparse 64×64 binary matrices applied on the entire internal state. Since we have exactly 3 indices in each row having “1” with the rest being “0”, we can describe the matrices solely based on these indices. Table 7 and 8 of Appendix B.2 provide the list of these indices.

We give here a miniature example of how to read these linear layer tables. The matrix $L_{ex} = \begin{bmatrix} 0 & 1 & 1 & 1 \\ 1 & 0 & 1 & 1 \\ 1 & 1 & 0 & 1 \\ 1 & 1 & 1 & 0 \end{bmatrix}$ can be represented as in Table 6.

Table 6: Example of how we represent the matrix L_{ex} using a table.

Index	L_{ex}		
0	1	2	3
1	0	2	3
2	0	1	3
3	0	1	2

Table 7: The linear layer of uKNIT-BC (row index 0 to 31).

Index	L_0	L_1	L_2	L_3	L_4	L_5	L_6	L_7	L_8	L_9	L_{10}
0	0 10 14	3 6 14	0 8 14	0 6 9	3 6 8	3 5 11	3 6 7	7 11 15	5 8 14	4 11 14	1 8 13
1	1 8 15	0 5 15	7 11 15	2 4 10	2 7 10	0 7 8	2 5 13	0 6 9	4 11 12	6 8 60	2 10 12
2	2 9 12	1 7 13	3 5 27	1 7 37	0 5 11	2 6 9	0 4 15	3 10 12	7 9 13	5 9 13	3 11 15
3	3 11 13	2 4 12	2 4 9	3 5 11	1 4 9	1 4 10	1 12 14	1 4 14	6 10 15	7 10 15	0 9 14
4	18 23 25	16 23 30	12 18 21	16 21 24	19 27 28	18 23 30	6 7 10	16 20 24	17 25 31	18 23 31	17 23 26
5	16 22 26	22 27 29	19 23 24	19 23 25	18 21 24	21 26 29	17 22 24	18 23 27	18 21 29	17 24 30	16 24 29
6	19 21 24	17 20 25	17 20 25	18 20 26	16 22 29	17 20 24	18 23 27	17 21 25	16 23 24	19 22 29	22 27 30
7	17 20 27	18 26 31	16 22 26	17 22 27	20 25 30	16 25 31	35 39 47	19 22 26	22 26 30	16 20 28	18 20 31
8	35 36 42	38 42 46	39 42 46	35 39 43	35 41 44	36 40 47	32 36 41	34 39 43	39 40 46	39 41 46	33 43 47
9	32 38 47	32 39 44	36 43 44	35 39 46	32 43 47	39 43 45	33 40 44	33 36 40	38 42 45	36 42 44	37 41 45
10	37 41 45	34 37 41	38 40 45	33 41 44	33 40 45	38 42 44	37 43 45	35 37 41	37 41 47	37 40 45	34 36 40
11	33 40 44	33 43 45	37 41 47	36 40 47	34 42 46	37 41 46	16 20 25	32 38 42	36 43 44	34 39 41	35 39 46
12	55 58 62	52 57 62	48 56 60	50 54 62	51 52 61	53 57 61	53 56 63	53 59 61	49 56 62	55 57 62	53 59 61
13	49 56 61	48 55 56	49 52 61	52 58 60	54 58 60	51 52 56	54 58 61	52 58 60	50 57 60	48 53 58	50 58 62
14	50 52 57	49 59 63	50 54 58	49 53 63	48 56 63	48 59 60	55 57 60	54 57 63	51 59 61	51 59 63	48 54 63
15	51 53 63	50 53 60	55 59 63	51 55 61	49 53 59	49 55 63	52 59 62	55 56 62	48 58 63	12 49 54	51 55 56
16	23 25 30	19 27 29	12 21 28	23 25 30	16 22 26	16 22 31	21 26 31	23 27 29	17 20 31	18 25 31	18 20 25
17	21 24 28	18 21 31	23 24 30	20 26 29	21 24 31	18 23 27	20 25 30	22 26 30	18 21 27	19 27 29	23 26 28
18	22 26 29	20 25 28	20 25 29	22 27 28	19 23 28	19 26 29	23 27 29	20 24 31	19 26 30	16 21 28	16 21 29
19	20 27 31	16 23 24	22 26 31	21 24 31	17 25 30	20 24 28	22 24 28	21 25 28	23 24 28	17 26 30	19 27 30
20	32 38 43	35 42 46	34 40 45	8 42 45	34 37 46	32 40 47	35 39 42	32 38 46	34 41 47	32 36 42	36 40 44
21	36 42 46	32 39 40	35 41 47	8 34 42	35 36 44	34 42 44	33 38 44	35 37 47	35 42 45	38 43 47	32 41 45
22	33 39 44	33 36 45	32 42 46	32 40 47	32 38 47	35 43 45	34 43 45	33 36 45	32 40 46	35 38 43	33 38 47
23	34 41 45	37 41 47	33 43 44	33 38 44	33 39 45	33 41 46	36 41 46	34 39 44	33 43 44	33 37 40	35 39 42
24	49 54 61	49 54 63	51 55 63	52 56 58	48 55 63	49 55 58	50 54 58	48 54 57	53 58 63	53 58 61	55 56 60
25	52 57 60	50 53 58	49 52 57	49 53 59	50 58 60	52 56 62	49 52 59	49 53 59	52 56 62	49 54 56	50 52 62
26	48 58 62	55 56 61	54 58 62	48 51 55	53 59 62	48 54 60	51 55 57	50 55 56	55 57 60	50 57 62	48 54 57
27	51 53 59	51 57 62	51 59 63	50 54 57	51 52 57	50 57 61	48 53 56	51 52 58	54 59 61	51 52 63	49 59 61
28	7 11 13	0 5 8	0 6 14	1 7 12	2 7 14	5 11 12	3 6 10	6 9 13	2 8 14	3 6 8	4 11 15
29	4 8 15	3 6 9	4 9 13	0 6 13	1 4 13	6 9 14	1 8 12	1 4 8	3 10 15	0 7 10	6 9 14
30	6 10 14	1 7 11	3 5 10	3 5 15	3 6 12	4 10 13	0 4 11	3 5 12	0 9 13	1 5 9	5 10 12
31	5 9 12	2 4 10	1 7 15	2 4 14	0 5 15	7 8 15	2 5 9	2 11 15	1 11 12	2 4 11	7 8 13

Table 8: The linear layer of uKNIT-BC (row index 32 to 63).

Index	L_0	L_1	L_2	L_3	L_4	L_5	L_6	L_7	L_8	L_9	L_{10}
32	38 43 47	33 36 43	33 36 43	34 42 45	35 36 41	33 37 46	39 42 47	34 43 44	32 39 46	33 37 45	39 42 46
33	34 37 45	34 41 47	35 37 47	33 38 41	33 39 40	35 39 45	32 41 46	35 41 47	34 37 47	35 38 47	32 37 45
34	33 39 40	35 38 46	34 38 45	39 43 46	34 37 42	34 38 44	34 37 45	32 42 46	33 36 44	34 39 46	33 38 43
35	35 42 46	39 40 44	32 39 46	32 36 47	32 38 43	32 36 47	33 38 40	33 40 45	35 38 45	32 36 44	34 40 44
36	50 57 60	49 54 59	48 53 56	53 59 63	52 57 61	48 54 59	50 54 61	48 54 63	48 53 58	48 58 61	50 52 58
37	53 59 63	51 52 62	52 57 61	48 55 61	48 55 56	51 56 62	49 52 62	50 55 62	50 55 57	51 52 59	49 53 61
38	48 55 62	53 58 60	50 58 62	54 57 62	49 59 62	50 53 61	48 53 63	51 52 60	51 54 59	50 55 62	51 56 60
39	49 54 56	48 56 61	48 53 60	52 56 60	50 54 60	55 58 63	51 55 60	49 53 61	49 52 56	12 54 56	54 57 63
40	2 5 9	6 9 14	1 11 15	0 9 13	2 10 14	2 9 14	19 21 26	3 5 10	2 5 14	1 5 13	2 5 10
41	0 6 10	5 8 15	5 10 27	1 12 37	0 11 15	3 11 12	5 9 13	2 7 15	1 4 12	22 27 29	3 4 11
42	3 7 11	4 10 12	2 9 13	16 24 31	1 9 13	0 8 15	4 11 15	4 8 14	3 6 15	2 4 14	0 6 9
43	1 4 8	7 11 13	0 6 8	2 10 14	3 8 12	1 10 13	8 12 14	0 9 13	0 7 13	3 6 60	1 7 8
44	17 27 31	17 25 28	19 24 30	3 11 15	17 20 30	19 21 29	17 24 28	17 25 28	17 20 25	23 25 31	19 22 30
45	19 24 28	18 21 26	12 18 28	18 26 29	18 24 31	17 24 28	19 26 31	16 24 31	21 27 29	20 21 28	17 26 28
46	18 25 30	23 24 30	17 25 29	17 27 28	22 26 29	16 22 25	18 27 29	19 26 30	16 24 28	24 26 30	16 21 24
47	16 26 29	19 22 29	16 26 31	19 25 30	19 23 27	23 27 30	16 25 30	18 27 29	19 22 30	0 7 15	20 25 31
48	48 55 58	50 58 60	53 56 60	56 58 60	55 56 63	54 59 60	48 56 63	50 56 62	48 53 63	50 55 57	48 57 63
49	51 59 63	48 55 61	49 57 61	49 59 63	51 57 61	49 58 63	50 58 61	48 57 63	49 52 62	12 49 56	52 58 62
50	50 52 60	54 59 63	50 54 62	48 51 61	49 53 62	50 53 57	49 59 62	51 58 60	50 55 60	52 59 63	49 53 59
51	54 56 61	51 52 57	51 55 59	50 57 62	50 54 58	51 52 62	51 57 60	49 59 61	51 54 61	48 53 61	51 55 60
52	3 7 13	2 10 12	3 10 27	5 11 15	4 9 13	0 7 15	3 7 10	0 6 13	0 7 9	1 9 13	2 5 12
53	1 4 15	0 8 15	2 4 13	7 12 37	6 8 12	1 4 13	0 11 15	1 8 14	3 6 10	2 11 14	1 7 13
54	2 5 12	1 11 13	1 7 11	4 10 14	5 11 15	3 5 12	2 9 13	2 7 11	1 4 11	0 10 15	3 4 15
55	0 6 14	3 9 14	6 8 14	6 9 13	7 10 14	2 6 14	1 8 14	5 10 12	2 5 8	3 8 60	0 6 14
56	17 20 31	19 22 27	16 22 31	18 20 29	17 20 25	17 20 28	17 22 28	17 21 28	19 22 26	19 22 27	21 24 29
57	19 21 28	21 26 31	18 21 28	17 22 28	16 26 29	19 21 26	18 23 29	19 22 30	20 25 31	18 23 25	17 23 28
58	18 23 30	17 20 28	17 20 29	16 21 31	18 21 31	22 25 31	16 20 30	18 23 29	16 23 28	17 24 26	19 22 27
59	16 22 29	16 24 30	19 23 30	19 23 30	23 27 28	18 27 30	19 21 31	16 20 31	18 27 29	16 20 21	18 25 31
60	39 40 44	36 43 45	34 38 40	8 34 45	36 41 44	32 36 40	32 36 46	39 43 44	35 38 42	32 42 44	38 43 47
61	34 37 41	35 38 42	32 39 42	35 43 46	39 40 45	33 37 41	38 40 44	37 41 47	34 37 41	34 41 46	34 36 44
62	32 43 47	34 37 47	35 37 41	38 41 44	38 43 47	34 38 42	34 37 43	38 42 46	33 36 43	33 40 45	35 42 46
63	35 36 46	32 40 44	33 36 44	32 36 40	37 42 46	35 39 43	35 42 47	36 40 45	32 39 40	35 43 47	32 37 41

C The matrices of MIDORI and PRINCE

Let $I_4 \in \mathbb{F}_2^{4 \times 4}$ be the 4×4 identity matrix in and $O_4 \in \mathbb{F}_2^{4 \times 4}$ be the 4×4 all zero matrix, and let $M_i \in \mathbb{F}_2^{4 \times 4}$ be a matrix that has zero everywhere except for diagonal entries (j, j) where $j \neq i$. Then, the MixColumns matrices of PRINCE and MIDORI are

$$\tilde{M}_{\text{PRINCE}}^{(0)} = \begin{bmatrix} M_0 & M_1 & M_2 & M_3 \\ M_1 & M_2 & M_3 & M_0 \\ M_2 & M_3 & M_0 & M_1 \\ M_3 & M_0 & M_1 & M_2 \end{bmatrix} \tilde{M}_{\text{PRINCE}}^{(1)} = \begin{bmatrix} M_1 & M_2 & M_3 & M_0 \\ M_2 & M_3 & M_0 & M_1 \\ M_3 & M_0 & M_1 & M_2 \\ M_0 & M_1 & M_2 & M_3 \end{bmatrix} M_{\text{MIDORI}} = \begin{bmatrix} O_4 & I_4 & I_4 & I_4 \\ I_4 & O_4 & I_4 & I_4 \\ I_4 & I_4 & O_4 & I_4 \\ I_4 & I_4 & I_4 & O_4 \end{bmatrix}$$

One may realise that $\tilde{M}_{\text{PRINCE}}^{(0)}$ is actually a rotation of $\tilde{M}_{\text{PRINCE}}^{(1)}$.

D Automatic search for differential key-recovery attacks

The complexity of a full key-recovery attack is influenced by both the distinguisher and the key-guessing processes. To search for a good key-recovery-friendly DC, we add the key-guessing part within our search as follows.

1. **Distinguisher.** The SAT model M_d used in the search for the distinguisher is produced.
2. **Backward propagation.** Assume that we add r_0 rounds before the DC, whose input difference is denoted by α . We need to describe the backward propagation of α to the plaintext with probability 1. Thus, we add the following constraints to ensure that:
 - the input and output differences of each S-box are either active or inactive simultaneously
 - for each linear layer, if one output difference bit b_o is active, then all input difference bits that are needed to calculate b_o must be activeWe call this model the backward propagation, denoted by M_b .
3. **Forward propagation.** Similar to the backward propagation, assume we append r_2 rounds to the DC, whose output difference is β . We construct a forward propagation that requires:
 - the input and output differences of each S-box to be either active or inactive simultaneously
 - for each linear layer, if one input difference bit b_i is active, then all output difference bits that are required for calculating b_i must be activeThis model is called forward propagation, denoted by M_f .
4. **Objective function.** To search for a key-recovery-friendly DC, we use two objective functions as follows,

$$\mathcal{A}(M_b) + \mathcal{A}(M_f) \leq O_1, \quad \text{P}(M_d) \leq O_2$$

where $\mathcal{A}(M_b)$ and $\mathcal{A}(M_f)$ are the number of active S-boxes in the two-end key-guess processes, $\text{P}(M_d)$ is the probability of the DC, and O_1 and O_2 are two positive numbers for the objective functions. In the search, starting from small values, we try all possible combinations of (O_1, O_2) , until we find one that returns a solution. Note that this model is rather intuitive in how it searches for good key-recovery attacks based on a DC, but it does not provide a guarantee that the key-recovery attack found is optimal.

E Example of a differential truncated path for uKNIT-BC

We provide in Figure 7 an illustration of how to generate a DTP given a DC.

$\Delta(x_2)$	0000be0700000000 00007d0e00000000	a_2	0000110100000000 0000110100000000
$\Delta(x_3)$	0000000000000070 00000000000000a0	a_3	0000000000000010 0000000000000010
$\Delta(x_4)$	0004000001000000 0004000008000000	a_4	0001000001000000 0001000001000000
$\Delta(x_5)$	0000040480200808 00000808a0a00802	a_5	0000010110100101 0000010110100101
$\Delta(x_6)$	02a4000034040000 0444000024040000	a_6	0111000011010000 0111000011010000
$\Delta(x_7)$	0000000020400000 0000000080200000	a_7	0000000010100000 0000000010100000
$\Delta(x_8)$	0000080000000002 0000080000000008	a_8	0000010000000001 0000010000000001
$\Delta(x_9)$	0004802000082040 000c204000028010	a_9	0001101000011010 0001101000011010

Fig. 7: Generation of a DTP from a DC of $\mathcal{W}(2, 8)$. For convenience, we also give the differences and patterns of $\Delta(y_i)$.

F Related-key differentials for uKNIT-BC

We provide in Table 9 the related-key differentials probabilities we found on uKNIT-BC.

Table 9: The negative log-2 of probabilities ($-\log_2(p)$) of the related-key differentials of uKNIT-BC for $\mathcal{W}(i, r)$ for $1 \leq r \leq 12$ and $0 \leq i \leq 12 - r$. The numbers in brackets mean that it is a lower bound.

$r \backslash i$	0	1	2	3	4	5	6	7	8	9	10	11
1	0	0	0	0	0	0	0	0	0	0	0	0
2	0	0	0	0	0	0	0	0	0	0	0	–
3	2	2	2	2	2	2	2	2	2	2	–	–
4	6	7	6	6	6	6	6	6	6	–	–	–
5	14	12	14	12	11	13	13	13	–	–	–	–
6	25	21	22	24	19	23	22	–	–	–	–	–
7	(29)	(29)	(29)	(30)	(30)	(31)	–	–	–	–	–	–
8	(32)	(33)	(33)	(32)	(33)	–	–	–	–	–	–	–
9	(35)	(35)	(35)	(35)	–	–	–	–	–	–	–	–
10	(41)	(42)	(41)	–	–	–	–	–	–	–	–	–
11	(45)	(45)	–	–	–	–	–	–	–	–	–	–
12	(49)	–	–	–	–	–	–	–	–	–	–	–

G Other security analysis of uKNIT-BC

We provide in this section a broader range of security analysis of uKNIT-BC.

G.1 Multiple differential cryptanalysis

Although uKNIT-BC is secure against differential attacks built from differential characteristics (Section 6.1), we note that $\mathcal{W}(2, 8)$ is the weakest among all 8-length windows, with a differential probability of 2^{-68} . We need to check if there are any differentials with clustering DCs for $\mathcal{W}(2, 8)$ whose probability could reach 2^{-64} . With our SAT model, we searched for 100 different DCs whose probabilities are exactly 2^{-68} . The 100 DCs belong to 72 distinct differentials, and all of them belong to 3 similar differential truncated patterns.

Definition 5 (Differential truncated pattern (DTP)). A differential truncated pattern for uKNIT-BC is a sequence of bit-strings as

$$T = (a_i, a_{i+1}, \dots, a_{i+r}) \in (\mathbb{F}_2^{16})^{r+1}.$$

We say that a DC $(x_i, x_{i+1}, \dots, x_{i+r})$ belongs to T if

$$x_i[t]_n = \begin{cases} 0 & \text{if } a_i[t]_b = 0; \\ \text{any} & \text{otherwise.} \end{cases}$$

In [38], Canteaut *et al.* applied a multiple differential attack to 10-round PRINCE, and later the same attack was applied to 11-round PRINCEv2 by the designers [36]. In the language of DTP, this attack traces all DCs belonging to some special DTPs where the Hamming weights of all a_i are 4. In this paper, we generalize their attack based on uKNIT-BC linear layers.

Figure 7 in Appendix E illustrates how to generate a DTP given a DC. By adapting Canteaut *et al.*'s method, we capture all DCs that belong to the DTP. First, for $2 \leq i < 9$, we enumerate all possible $\Delta(y_i)$ that satisfies the active pattern of a_i while $L_i(\Delta(y_i))$ satisfies the pattern a_{i+1} . Denote the Hamming weight of a_i by w_i and $h_i = \min\{w_i, w_{i+1}\}$, there are 2^{h_i} possible values for $\Delta(y_i)$. Then, we can generate a matrix to describe the propagation of the differences from $\Delta(y_i)$ to $\Delta(x_{i+1})$. Let $M \in \mathbb{F}_2^{h_i \times h_i}$ and the entry at the cross of the u^{th} column and v^{th} row, denoted by $M_{v,u}^{L_i}$, is defined as

$$M_{v,u}^{L_i} = \begin{cases} 1 & u = \text{sup}(\delta_{\Delta(y_i)}) \text{ and } v = \text{sup}(\delta_{\Delta(x_{i+1})}); \\ 0 & \text{otherwise.} \end{cases}$$

where $\text{sup}(b)$ is the support of a bit vector b and δ_x is a bit unit vector with the entry at x being 1. Thus, $\delta_{\Delta(x_{i+1})} = M_{v,u} \delta_{\Delta(y_i)}$. Then, according to a_i , we can generate all 2^{w_i} possible $\Delta(x_i)$. For all $\Delta(x_i)$ and $\Delta(y_i)$ we can also generate a matrix $M^{S_i} \in \mathbb{F}_2^{w_{i+1} \times w_i}$ with

$$M_{v,u}^{S_i} = \text{P}(\Delta(x_i) \xrightarrow{S_i} \Delta(y_i)) \text{ where } u = \text{sup}(\delta_{\Delta(x_i)}) \text{ and } v = \text{sup}(\delta_{\Delta(y_i)}).$$

Therefore, using the same principles mentioned in [38], the (k, l) -entry of

$$M^{\mathcal{W}(2,8)} = M^{S_9} \times \prod_{8 \geq i \geq 2} (M^{L_i} \times M^{S_i})$$

encodes the probability of the differential where the input difference is represented by the index k of the support for a_2 and the output difference is represented by index l of the support for a_9 . With the DTP in Figure 7 in Appendix E, we compute $M^{\mathcal{W}(2,8)}$ and the best differential of this DTP has a probability of $2^{-63.3}$. We also found that the other two DTPs and the probability of their respective best differential are also approximately $2^{-63.3}$. These differentials have probabilities larger than but very close to 2^{-64} . Key-recovery attacks based on these differentials would therefore admit extremely high data and time complexities. Considering our security claim, we do not think they will be a threat to uKNIT-BC.

G.2 Impossible differential cryptanalysis

Impossible differential (ID) cryptanalysis was proposed independently by Knudsen [73] and Biham *et al.* [20], remaining one of the most powerful attacks on block ciphers. The automatic search methods for ID distinguishers were developed from the DC searches [96,42]. By implementing these automatic models with our SAT solvers, we detected the following two impossible truncated differentials over $\mathcal{W}(3, 7)$. Denote one active S-box by 1 and inactive S-box by 0, the 2 impossible differentials are

$$\begin{aligned} (0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1) &\stackrel{7R}{\not\rightarrow} (1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0) \\ (0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1) &\stackrel{7R}{\not\rightarrow} (0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0) \end{aligned} \quad (2)$$

We use the second impossible differentials (Equation (2)) in a key-recovery attack on $\mathcal{W}(1, 11)$. Two rounds are added before the distinguisher and two rounds after. The key-recovery attack process is shown in Figure 8. The ID of $\mathcal{W}(3, 7)$ is from $\Delta(x_3)$ to $\Delta(y_9)$. $\Delta(x_3)$ propagates backward to the plaintext $\Delta(x_1)$ with probability 1 and $\Delta(y_9)$ propagates forward to the ciphertext with probability 1. The active cells are shown in Figure 8. The key-recovery process is as follows.

1. Initialize a table \mathbb{H} with the 92-bit $k_1[1 \sim 15]_n, k_2[8 \sim 11]_n, k_{11}[7, 10, 11, 13]_n$ as the index and 0 for all the values.
2. As shown in Figure 8, one plaintext structure contains 2^{60} plaintexts, yielding a total of 2^{119} pairs. Encrypt all plaintexts of one structure, combine them into pairs with a hash table. Since there are 10 inactive cells in the ciphertext side, $2^{119-40} = 2^{79}$ pairs survive. For 2^t structures, there are 2^{t+79} pairs.
3. For each pair, we guess the 2^{16} possibilities of $\Delta(x_2)$ differences and extract the 60-bit $k_1[1 \sim 15]_n$ and 16-bit $k_2[8 \sim 11]_n$ with some look-up tables. At the side of the ciphertexts, 16-bit $k_{11}[7, 10, 11, 13]_n$ can also be extracted with table look-ups.

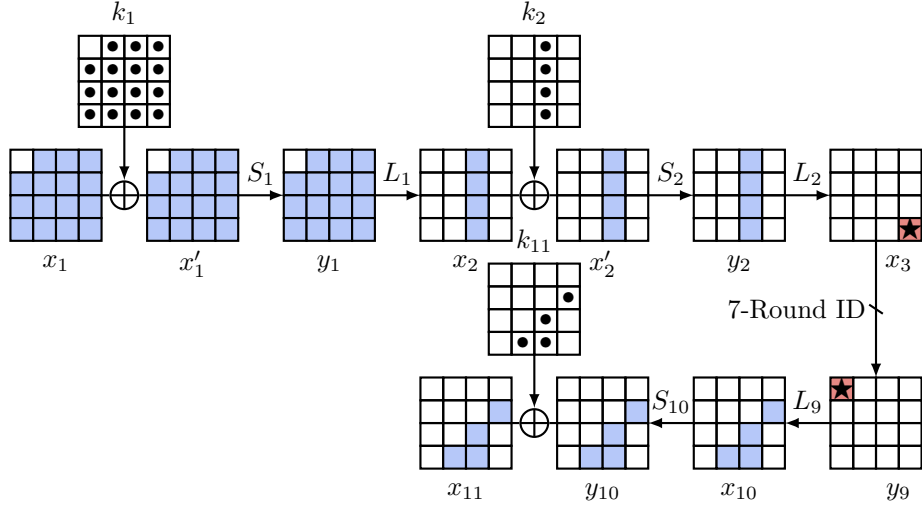


Fig. 8: The key-recovery process for $\mathcal{W}(1, 10)$ based on the second ID of $\mathcal{W}(3, 7)$ (Equation (2)).

- On average, a wrong pair can reach the input and output differences of the ID with a probability of $2^{-56-12} = 2^{-68}$. Thus, with 2^{t+79} pairs, the probability that one wrong key cannot be excluded by all pairs is $(1 - 2^{-68})^{2^{t+79}} = (1 - 2^{-68})^{2^{68} \times 2^{-68} \times 2^{t+79}} \approx e^{-2^{t+11}}$. As we want the number of remaining wrong keys to be less than 1, we have

$$2^{92} \times e^{-2^{t+11}} < 1,$$

which means $t \geq -5.0$. In other words, we only need 2^{55} plaintexts and process $2^{55 \times 2 - 1 - 40} = 2^{69}$ pairs.

Complexity. The data complexity is 2^{55} chosen plaintexts. The time complexity is dominated by processing each pair and updating \mathbb{H} for labelling those impossible keys. For each pair, on average there are $2^{92-68} = 2^{24}$ keys that can pass the filters. Thus, we need to access \mathbb{H} 2^{24} times for each pair. Since \mathbb{H} is a large table, we regard each access to \mathbb{H} as one encryption. The time complexity is eventually $2^{69+24} = 2^{93}$ encryptions. The memory complexity is dominated by \mathbb{H} , which is 2^{92} bits that is equivalent to 2^{86} 64-bit blocks.

G.3 Zero-correlation cryptanalysis

Zero-correlation (ZC) attacks are an extension of linear cryptanalysis based on linear hull with exact zero correlation (ZC linear hull) [30,29,31]. The automatic search methods are similar to those for ID, which was first proposed in [42].

With our SAT tools, we detected the following two ZC linear hulls over $\mathcal{W}(3, 7)$. Denote one active S-box by 1 and inactive S-box by 0, the 2 ZC linear

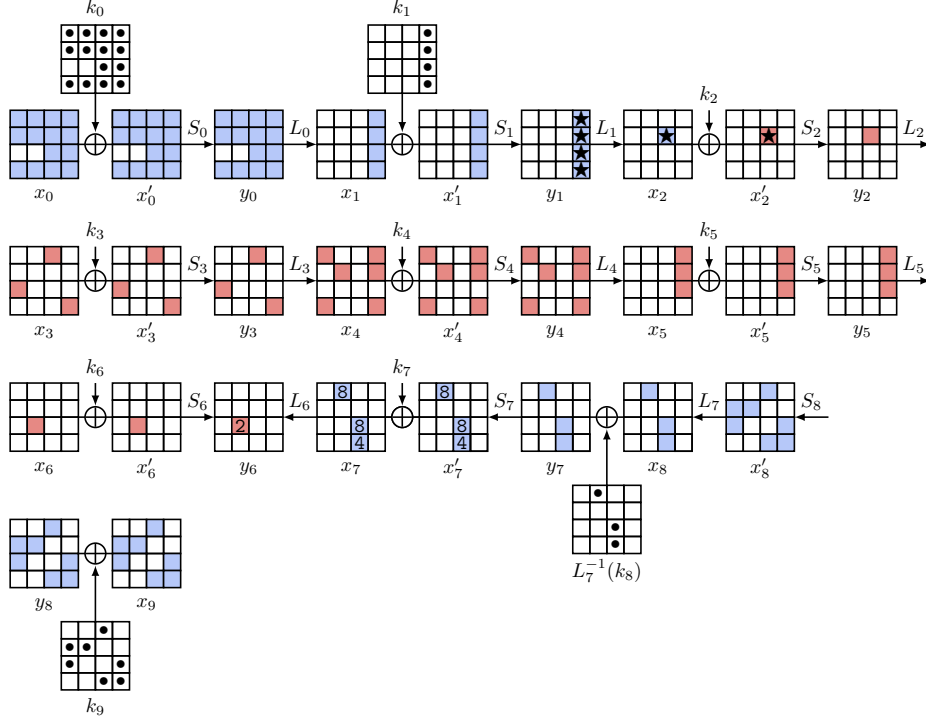


Fig. 9: The DS-MITM attack process for $\mathcal{W}(0,9)$. The distinguisher is applied to $\mathcal{W}(2,5)$ and labeled in red, while the online phase is labeled in blue.

give the definition of δ -set [49] in our context, which is a core notion in DS-MitM attacks.

Definition 6 (δ -set for uKNIT-BC). *A δ -set of size 2^b is a set of internal states such that b bits of the states take up all their possible values while other bits remain constant.*

The distinguisher in this attack covers from x'_2 to y_6 . We choose a 2^4 -size δ -set for x'_2 such that $x'_2[9]_n$ takes up all 16 possible values. Suppose that the vectorial Boolean function that sends $x'_2[9]_n$ to $y_6[6]_n$ is f , then we have the following observations.

Observation 1 *Suppose that the 16 elements in the δ -set are X_0, X_1, \dots, X_{15} . The total number of possible 60-bit ordered sequence*

$$f(X_0) \oplus f(X_1), f(X_0) \oplus f(X_2), \dots, f(X_0) \oplus f(X_{15}) \quad (3)$$

is fully determined by the 56-bit parameters,

$$x'_2[9]_n, x'_3[2, 8, 15]_n, x'_4[0, 3, 5, 12, 13, 15]_n, x'_5[12 \sim 14]_n, x'_6[6]_n \quad (4)$$

where these parameters correspond to one element of the δ -set.

With the *differential enumeration technique* [56,52], the number of parameter bits in Equation (4) can be reduced to 32 bits.

Observation 2 (Differential enumeration technique [56,52]) *Suppose that we have a truncated differential $\Delta(x'_2) \rightarrow \Delta(y_6)$ where $\Delta(x'_2)[9]_n = \star$ (any value except 0) and $\Delta(y_2)[6]_n = 2$. The 56-bit parameters can be determined by the following 36-bit differences,*

$$\Delta(x_2)[9]_n, \Delta(y_2)[9]_n, \Delta(y_3)[2, 8, 15]_n, \Delta(x_5)[12 \sim 14], \Delta(x_6)[6]_n.$$

As we have 2^{60} possibilities of Equation 3 in the uniformly random situation, Observations 1 and 2 can be used as a distinguisher. The key-recovery attack process is described as follows.

Offline phase. We compute 2^{32} 60-bit sequences of Equation 3 and store them into a hash table \mathbb{H} .

Online phase. In the online phase, we find one right pair that satisfies the truncated differential in Observation 2 and recover the corresponding keys. Then, we use the keys to compute the sequence in Equation 3.

1. As shown in Figure 9, we propagate $\Delta(x'_2)$ backward to $\Delta(x_0)$, and propagate $\Delta(y_6)$ forward to $\Delta(x_9)$. By doing so, we constructed a plaintext structure containing 2^{56} plaintexts, yielding 2^{111} pairs. Since the ciphertext difference has 9 inactive nibbles, $2^{111-36} = 2^{75}$ pairs will survive.
2. For each pair,
 - $\Delta(x_0)$ reaches $\Delta(x_2)$ (with $\Delta(x_2)$ being any non-zero difference) with a probability of 2^{-52} ;
 - $\Delta(x_9)$ propagates to $\Delta(y_6) = 2$ is with a probability of 2^{-28} .
 In total, the probability to obtain a right pair is $2^{-52-28} = 2^{-80}$. Therefore, the number of structures required to get a right pair is $2^{80-75} = 2^5$.
3. For each of these 2^{80} right pair candidates, at the plaintext side, we guess all possible 2^{16} $\Delta(x_1)$ and 2^4 $\Delta(x_2)$, then extract on average one candidate of $k_0[0, 1, 3 \sim 5, 7 \sim 15]_n$ and $k_1[12 \sim 14]_n$. At the ciphertext side, we enumerate all possible 2^{12} $\Delta(y_7)[4, 10, 11]_n$, and we can extract on average one $L_7^{-1}(k_8)[4, 10, 11]_n$ and $k_9[1, 2, 5, 8, 14, 15]_n$.
4. Suppose that we select $x'_2[9]_n = k_2[9]_n$, then we can get the δ -set at x_2 . Decrypting them to x_0 with the keys we obtain, we get all 16 plaintexts for the δ -set. Encrypt the 16 plaintexts to ciphertexts. Using the guessed k_8 and k_9 bits to decrypt these 16 ciphertexts to y_6 , compute the sequence of Equation (3).
5. Check if the calculated sequence is in \mathbb{H} built in the offline phase. Since the number of possibilities of Equation (3) is 2^{56} (note that 4 bits have been determined by the truncated differential), only $2^{80-(56-36)} = 2^{60}$ pairs can survive. For each surviving pair, we have on average 15 values for $k_0[0, 1, 3 \sim 5, 7 \sim 15]_n$, $k_1[12 \sim 14]_n$, $L_7^{-1}(k_8)[4, 10, 11]_n$ and $k_9[1, 2, 5, 8, 14, 15]_n$ according to the guessed $\Delta(x_2)$, or equivalently, we have about 2^{62} key candidates with its corresponding plaintext, ciphertext and intermediate pairs.

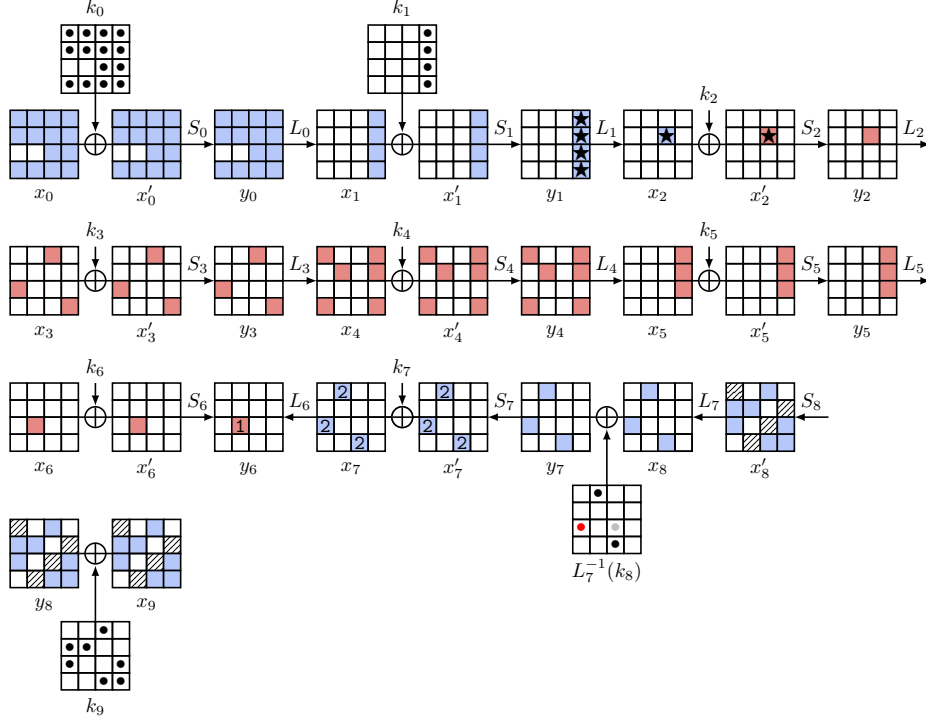


Fig. 10: The second DS-MitM key-recovery on $\mathcal{W}(0,9)$.

6. To further determine the keys, we can play the DS-MitM with a slightly modified distinguisher shown in Figure 10. As the probability that a pair of the 2^{60} being a right pair is 2^{-44} , with 2^{44} surviving pairs, we can find a right pair for the second DS-MitM attack. So the same key candidates with the first attack except that $L_7^{-1}(k_8)[10]_n$ is replaced with $L_7^{-1}(k_8)[2]_n$ can be recovered. The filtering effect is 2^{-20} again. Thus only $2^{62-20+2} = 2^{44}$ key candidates survives. The time complexity of the second attack has been negligible compared to the first one. We can repeat this process several times, until we recover the only one key candidates.
7. Finally, we can exhaustively search the unknown 56 bits of k_0 and k_1 , and use the 60-bit values of k_8, k_9 , as well as the internal keys derived from the right pairs as filters, Finally, we expect only correct key.

Complexity. The complexities are all dominated by the first attack. The data complexity basically consists in encrypting the 2^{61} plaintexts. The memory complexity represents the storage of the 2^{36} 60-bit sequences, which is 2^{40} bits or 2^{34} 64-bit blocks. The time complexity of the offline phase is to computing all sequences, which is $2^{36+4} = 2^{40}$ 5-round encryptions. For the online phase, the time complexity is dominated by processing all 2^{80} pairs. For each pair, we need to guess 15 $\Delta(x_2)$, and the remaining time complexity is dominated by partially

encrypting the δ -set, which is $2^{80+4+4} = 2^{88}$ 4-round encryptions, approximately, 2^{87} 9-round encryptions.

G.6 Boomerang and rectangle attacks

The boomerang attack was proposed in 1999 as a variant of differential cryptanalysis [108]. In this attack, we first throw a plaintext pair (p_0, p_1) with $p_0 \oplus p_1 = \alpha$ and obtain their corresponding ciphertexts (c_0, c_1) . Then, we choose c_2 and c_3 such that $c_0 \oplus c_2 = \beta$ and $c_1 \oplus c_3 = \beta$, and decrypt c_2, c_3 to get corresponding p_2, p_3 . We observe if $p_2 \oplus p_3 = \alpha$. If so, we say that the boomerang returns and (p_0, p_1, p_2, p_3) is called a right quartet. The boomerang attack is an adaptively-chosen-ciphertext attack, so in many cases, we prefer its chosen-plaintext version, called rectangle attack [71,21]. The rectangle attack generates plaintext quartets (p_0, p_1, p_2, p_3) satisfying $p_0 \oplus p_1 = \alpha, p_2 \oplus p_3 = \alpha$ and observe after encryption if $c_0 \oplus c_2 = \beta, c_1 \oplus c_3 = \beta$. If so, (p_0, p_1, p_2, p_3) is called a right quartet. We apply the rectangle attack to uKNIT-BC in this subsection.

For a random permutation with block size n , a right quartet appears with a probability of 2^{-2n} . To estimate the right quartet probability for a block cipher E , classically we usually divide the cipher into 2 parts as $E = E_1 \circ E_0$. We then find a good differential for E_0 with the input difference α whose probability is p , and a good differential for E_1 with the output difference β whose probability is q (in practice, we find these differentials first and then determine α and β afterwards). The probability of obtaining a right quartet is then estimated as $2^{-n}p^2q^2$. Therefore, only when $p^2q^2 > 2^{-n}$, then we will have a rectangle distinguisher. According to Table 2, the best rectangle attack on uKNIT-BC can only reach 6 rounds.

Such an estimation heavily relies on the assumption that E_0 the E_1 are independent, which is not always true in practice [87]. Several refinements have been proposed to mitigate such independence problems [57,58,40]. The boomerang connectivity table (BCT) is one of the latest technique among them [40].

Definition 7 (Boomerang connectivity table (BCT) [40]). For an n -bit S-box, denoted by S , its BCT is a 2-dimensional $2^n \times 2^n$ table, with the entry $\text{BCT}(\delta, \delta')$ being

$$\text{BCT}(\delta, \delta') = \#\{x \in \mathbb{F}_2^n : S^{-1}(S(x) \oplus \delta') \oplus S^{-1}(S(x \oplus \delta) \oplus \delta') = \delta\}$$

The cipher is decomposed into three parts, $E = E_2 \circ E_1 \circ E_0$ and in the simple BCT attack, E_1 is only one substitution layer. We first find a good differential of E_0 , say $\alpha \rightarrow \alpha'$ and a good differential of E_2 , say $\beta' \rightarrow \beta$. For $E_1 = S_0 || S_1 || \dots || S_{15}$ where S_i are the 16 S-boxes, we have the following computation

$$\begin{aligned} P(E_1^{-1}(E_1(x) \oplus \beta) \oplus E_1^{-1}(E_1(x \oplus \alpha') \oplus \beta) = \alpha') \\ = \prod_{0 \leq i < 16} P(S_i^{-1}(S_i(x[i]_n) \oplus \beta[i]_n) \oplus S_i^{-1}(S_i(x[i]_n \oplus \alpha'[i]_n) \oplus \beta[i]_n) = \alpha'[i]_n) \end{aligned} \tag{5}$$

For each sbox, the right side of the equation is computed using its BCT. Denote the probability of E_1 calculated from Equation (5) by r , then the probability of obtaining a right quartet of E is estimated to be $2^{-n}p^2q^2r$. When $p^2q^2r > 2^{-n}$, then we have a rectangle distinguisher.

We constructed a SAT model for the rectangle distinguisher incorporating the BCT. Our targets are the windows with length 7. For $\mathcal{W}(i, 7)$, a differential search model covers $\mathcal{W}(i, 3)$ as E_0 and $\mathcal{W}(i+3, 3)$ without the first S-box layer as E_2 . The middle S-box layer is described using the BCT. The objective function is then to maximize p^2q^2r .

For windows with length 8, there are no rectangle distinguishers whose right quartet probability is close to 2^{-128} . However, for 7-length windows, the best rectangle distinguisher we detect is for $\mathcal{W}(2, 7)$, whose right quartet probability is 2^{-115} . We also modeled the key-recovery process, i.e., the backward propagation and forward propagation, based on the distinguisher part, which is similar to the differential attacks (Section 6.1). The key-recovery process in rectangle attacks can be complicated and many improvements have been proposed [22,100,113]. We have checked the complexity of the rectangle attack based on the formulas in [100,113] and we concluded that the key-recovery attack will not reach 10 rounds.

G.7 (Higher-order) differential-linear attacks

Similar to the boomerang attacks, resistance against the plain differential and linear cryptanalysis does not necessarily lead to resistance against their variants or combinations. Differential-linear cryptanalysis, proposed by Langford and Hellman in 1994 [76], combines the differential and linear attacks. For a cipher E , let $C = E(P)$ and $C' = E(P')$. Given a difference-mask pair (δ, λ) , the correlation of the DL distinguisher is

$$C(\delta, \lambda) = 2^{-n} \sum_{x \in \mathbb{F}_2^n} (-1)^{\lambda \cdot (E(x) \oplus E(x \oplus \delta))}$$

Similar to the case of linear cryptanalysis, if the correlation is not 0, we have an opportunity to distinguish the cipher from a random permutation.

The classical method to estimate the correlation is to divide the cipher into two parts as $E = E_1 \circ E_0$. If we have a DC for E_0 , say $\delta \rightarrow \delta'$, with a probability p , and a LC for E_1 say $\lambda' \rightarrow \lambda$, with a correlation q . The correlation can be estimated as pq^2 [76]. However, such an estimation relies on many assumptions such as E_0 and E_1 being independent. To overcome this assumption, Dunkelman *et al.* introduced the differential-linear connectivity table (DLCT) to connect the differential and linear parts [8]. The DLCT is defined as follows,

Definition 8 (Differential-linear connectivity table (DLCT) [8]). For an n -bit S-box denoted by S , its DLCT is a 2-dimensional $2^n \times 2^n$ table, with entry $\text{DLCT}(\delta, \lambda)$ being

$$\text{DLCT}(\delta, \lambda) = \sum_{x \in \mathbb{F}_2^n} (-1)^{\lambda \cdot (S(x) \oplus S(x \oplus \delta))}$$

With the DLCT, a cipher is decomposed into three parts, say $E = E_2 \circ E_1 \circ E_0$, where E_1 is an S-box layer. We then construct a SAT model to search for the DL distinguishers for uKNIT-BC. In the SAT model, we model E_0 and E_2 to search for DC and LC respectively and for E_1 , we model the DLCT for each S-box. With this method, the best DL distinguisher we could find has a correlation of 2^{-25} for $\mathcal{W}(2, 7)$. However, for 8 rounds, no DL distinguisher exists and their correlations are far from 2^{-32} . Therefore, we believe that a DL key-recovery attack would be weaker than the differential or linear attack, thus uKNIT-BC is resistant against DL cryptanalysis.

Higher-order differential-linear (HDL) cryptanalysis was first proposed by Biham *et al.* to study possible combined attacks [23]. In [66], the authors used the HATF technique inspired by the ATF methods [80] to give a practical estimation process of the HDL bias. This attack strategy can be effective against some low-degree permutation-based ciphers, but considering that uKNIT-BC is a block cipher with degree-3 S-boxes (all S-boxes are bit-permutation equivalence of the MANTIS S-box), we believe uKNIT-BC resists HDL attacks.

G.8 Invariant attack

Invariant attacks study the structural properties of a cipher where a partition of the plaintext space into a set S or its complement is preserved. In [77], Leander *et al.* introduced the invariant subspaces and in [104] Todo *et al.* presented the nonlinear invariants. Beierle *et al.* studied how to choose the constants to resist invariant-type attacks in [10]. Later, in [16] Beyne showed that the invariant attacks can be effectively explained by the eigenvectors of correlation matrices [43]. As discussed in [35, Chapter 9], the invariant attacks mainly provide threats to lightweight ciphers with a simple key schedule, *e.g.*, the identical round keys with different constants XORed (like PRINCE) or two round keys are used alternatively (like MIDORI). For a cipher with a relatively complicated key schedule, invariant attacks are much more difficult to be found. With our key schedule based on the gSTK, we believe that invariant attacks are not a threat to the security of uKNIT-BC.

G.9 Slide attacks and internal differential attacks

The slide attack and its variants [26,27,59,61] explore the symmetry of different round functions. While the internal differential attacks [93,53,116,115] uses the symmetry between a part of the round function and the other part. uKNIT-BC enjoys a highly non-aligned property where (almost) all S-boxes and linear layers are different so it has an inherent strong resistance against both attacks.

H Application to PRF: XoP with uKNIT-BC

One of the benefits of uKNIT is not only providing very competitive minimum latency, but also a wider range of area-latency trade-offs. As demonstrated in the last two rows of Table 3, uKNIT can provide $\sim 10\%$ lower latency than PRINCEv2, but, interestingly, for the same latency as PRINCEv2 it requires only $\sim 80\%$ of the area. This leads to a very interesting trade-off: consider a user that can dedicate the same latency and area as PRINCEv2 for low latency applications but requires a PRF rather than a block cipher, *e.g.*, using the CounTeR (CTR) mode. It is well known that using PRINCEv2 in the CTR mode is insecure as it only offers 32 bits of security. On the other hand, consider the single-permutation Sum of Permutation (XoP) construction given by:

$$E_K(0\|M) \oplus E_K(1\|M).$$

It was given in [65] and its security was proven in [47], where it was shown to have a PRF security bound $O(q/2^n)$, where $q < 2^{n-1}$ is the number of queries and n is the block size. Note that the construction consists of two calls to the block cipher with the same key. Thus, it requires implementing the main SPN twice, while the key schedule only once. This leads us to conclude that for roughly $1.33\times$ area as PRINCEv2 we can double the PRF security. This is an extra level of security optimization, on top of optimizing the primitive itself. Note that since PRINCEv2 does not have a key schedule per se (the round keys are just copies of the master key), the same does not apply to prince. At $1.65ns$, XoP-uKNIT-BC would need $\approx 37551.59\mu m^2$, while XoP-PRINCEv2 would need $\approx 55,128.24\mu m^2$.

Note that Orthros and Gleeok128 are both PRFs based on other variants of XoP, where the internal block ciphers are pruned by reducing their number of rounds compared to a secure PRP version. uKNIT has lower latency than both of them without this trade-off and is transformed into a PRF using the XoP construction above. However, the former PRFs operate on 128 bits.

