IACR Transactions on Symmetric Cryptology
ISSN 2519-173X, 59 pages.

uKNIT: Breaking Round-Alignment for Cipher Design

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

Kai Hu[1,4,5], Mustafa Khairallah[2], Thomas Peyrin[3] and Quan Quan Tan[3,6]

[1] School of Cyber Science and Technology, Shandong University, Qingdao, China
[2] Umeå University, Umeå, Sweden
[3] Nanyang Technological University, Singapore, Singapore
[4] State Key Laboratory of Cryptography and Digital Economy Security, Qingdao, China
[5] Suzhou Research Institute, Shandong University, Suzhou, China
[6] Inria, Paris, France

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 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 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 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` has an improvement in latency (10% better), while increasing resistance against classical differential/linear cryptanalysis. It also reduces the area by 17% and energy consumption by 44% when fixing the latency of both ciphers. As a contribution of independent interest, we describe a generalization of the Superposition-Tweakey (STK) construction for key schedules, unlocking its application to bit-oriented ciphers. We also discuss the benefits of `uKNIT` to many other possible use-cases.

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

Emails: kai.hu@sdu.edu.cn (Kai Hu), mustafa.khairallah@umu.se (Mustafa Khairallah), thomas.peyrin@ntu.edu.sg (Thomas Peyrin), quan-quan.tan@inria.fr (Quan Quan Tan)

This work is licensed under a "CC BY 4.0" license.

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) [MP13, SWW21, WWS23, LZH+24], Mixed-Integer Linear Programming (MILP) [MWGP11, BGG+23a, BGG+23b, LXC+23], and Constraint Programming (CP) [GLMS20, DDG+23, CHNE24] solvers have become essential in the cryptanalyst's arsenal, automating substantial parts of the analysis, even including the key-recovery phase [QDW+21]. Their ease of use has allowed researchers to explore more complex and interesting search spaces in the analysis of ciphers.

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). The core structure of most modern symmetric-key algorithms continues to be crafted by cryptographers, with automation typically limited to selecting internal components of the function. This approach is exemplified in widely adopted standards like `AES`, `SHA-2`, and `SHA-3`. This was historically favored for compact implementations and ease of human-guided analysis/provable security, but both advantages are increasingly irrelevant:

- In many scenarios, such as in fast software/low-latency hardware, unrolled implementations do not pose significant challenges (they are actually favored in low-latency hardware),
- Automated tools now dominate cryptanalysis, rendering the human-friendly iterative structure less critical as these tools fundamentally do not assume or care about iterative structures.

This opens the door to harnessing the power of automated tools for the design process itself as the potential remains relatively underexplored (one counter-example being the search for large `AES`-round-based permutations [JN16, Nik17, SLN+22]). We believe that harnessing these tools for the design process could lead to the discovery of more secure and efficient cryptographic primitives.

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 does not seem 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. The notion of "low latency" varies across applications and typically spans several orders of magnitude across deployment scenarios. In high-performance software and hardware acceleration settings, block-cipher operations are often expected to incur latencies on the order of a few nanoseconds to a few tens of nanoseconds per block (i.e. a few of CPU cycles). In embedded and cyber-physical systems such as automotive controllers, industrial control units, and real-time signal-processing pipelines, cryptographic operations must instead complete within microsecond-scale budgets to avoid interfering with control loops and time-critical sensing tasks. In pointer encryption, an encryption or decryption query may be invoked on every memory access or branch operation, requiring costs of only a few CPU

cycles per invocation. `Qarmav1` [Ava17], for instance, is used for pointer authentication on ARM architectures [Qua17].

This is notably different from the goal of lightweight cryptography which has just been standardized by the NIST [TMC+23]), 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. For instance, `PRINCE` is already used for RAM memory encryption/authentication [NXP23]; `BipBip` is designed to authenticate pointers for system security [BDD+23a] and `ChiLow` for embedded code encryption [BDG+25].

The first published cipher with low-latency as stated target is the block cipher `PRINCE` [BCG+12], later upgraded to `PRINCEv2` [BEK+20]. Other low-latency block ciphers have been proposed by the community [KDGD20, LMMR21, ZWZ+22, GMW24], as well as low-latency tweakable block ciphers [BJK+16, Ava17, ABD+23, CGL+23, BDD+23a, ADM24, BII+25][1] and various other low-latency primitives such as PRFs [BIL+21, BDD+23b, ABC+24, WHWL24, ABD+24]. Unfortunately, similarly to lightweight cryptography, the heavy constraints imposed on the performance led to several candidates presenting security issues or little security margin [DEKM16, MKPA22, BDBN23, BN24]. 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 [BDD+23a, CGL+23]. For more aggressive schemes such as `BipBip` and `ChiLow`-32, full round attacks have been proposed in [WBD+24] and [GWZ25, LNV25] respectively.

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 [LMMR21] (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 building the primitive 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 for locally adapting 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 a designer trying to maximize the resistance of his

[1] Independently, [BII+25] has also tackled the problem of breaking round-alignment, albeit not using an automated way to do so

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. `uKNIT-BC`, compared to the current best-performing low-latency block cipher, `PRINCEv2` [BEK+20] has a reduction of the latency by 10%. It also 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 a 64-bit cipher for 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 for many other sizes (128-bit and more), many other areas and use cases (e.g., lightweight cryptography, fast software encryption). We discuss some of them in Section 8.

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 has been 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 [JNP14] and later influenced various tweakable block cipher constructions [BJK+16]. Our generalization, so-called gSTK, is particularly interesting as it can apply to bit-oriented ciphers (in contrast to STK), while ensuring sufficient diffusion within the round keys.

Outline. In Section 2, we briefly describe the `uKNIT` framework and in Section 3 a method to automatically generate `uKNIT` cipher instances for SPNs, based on a genetic algorithm, concentrating on the case of low-latency, explaining the rationale for our low-latency block cipher proposal `uKNIT-BC`, later fully specified in Section 5. In Section 4, we introduced the generalization of the STK construction for key schedule. Then, 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 discuss and showcase possible other usages of `uKNIT` in Section 8 and conclude in Section 9.

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 $R_0, R_1, \ldots, R_{r-1}$ 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` [BBdS+20]). For example, an r-round iterative block cipher E, parameterized by a key K, taking as input a plaintext P, can be written as:

$$E_K(P) = R_{r-1,k_{r-1}} \circ R_{r-2,k_{r-2}} \circ \cdots \circ R_{0,k_0}(P).$$

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

$$R_{i,k_i} = L \circ S(X \oplus k_i \oplus c_i)$$

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 [DR01] for the `AES` block cipher is 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 [JR93] 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. The `SERPENT` [BAK98] block cipher, an `AES` competition candidate, is another example. It alternates between eight distinct Sboxes S_i using $S_{i \bmod 8}$ in round i.

A notable earlier line of work that seeks to gain security from non-identical rounds is the design methodology for Feistel ciphers that enforces optimal diffusion mappings over a small number of consecutive rounds [SP04, SS04]. The goal is to prevent difference cancellations in the Feistel structure that can arise when only a small number of S-boxes are active.

Another line of work deviating a little from the pure iterative constructions is the arithmetization-oriented family of primitives `LowMC` [ARS+15] and `RASTA` [DEG+18]. 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` [GLR+20] and `PRINCE` [BCG+12] 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 $R_0, R_1, \ldots, R_{r-1}$ applied to the input, where each round R_i does not necessarily apply the same transformation. Reusing our block cipher example, we would typically have

$$R_{i,k_i} = L_i \circ S_i(X \oplus k_i \oplus c_i)$$

where layers L_i and S_i might be completely distinct.

One obvious benefit from the `uKNIT` framework is that the space of 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. 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 one 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 an 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 a 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 fewer rounds, but that do 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.

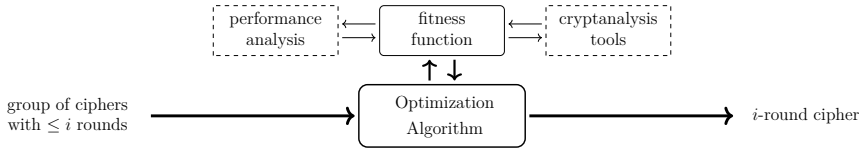


Figure 1: One step of our knitting methodology.

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.

In Figure 2, we describe the flow of the genetic algorithm we have used. We will explain each step, using the instance we have to obtain `uKNIT-BC`.

3.1 Structure selection

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

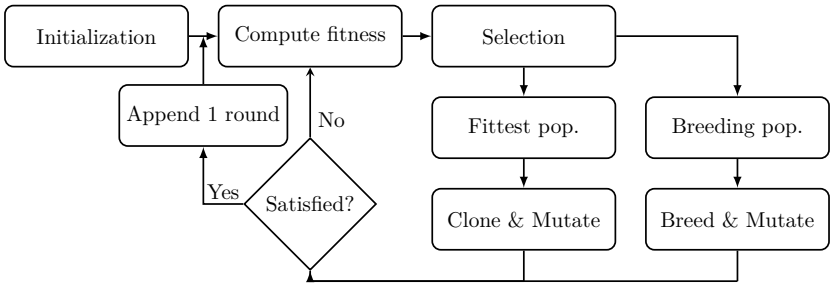


Figure 2: The flow of the genetic algorithm to search for SPN ciphers.

is influenced by existing low-latency and low-energy designs such as `PRINCE` [CFG+14, BEK+20], `MANTIS` [BJK+16] and `MIDORI` [BBI+15]. For the 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 (and modeling techniques) 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. *Remark.* We remark that while it is the choice that we made, it does not mean that the tool is incapable of performing for ciphers with a larger size. It just means that we have to use other techniques to evaluate it, such as the counting of active sboxes used in `AES` [DR01] or `SKINNY` [BJK+16].

Number of rounds. During the search, 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 to it and thus, slowed down the process significantly. Moreover, ciphers with fewer rounds are also more vulnerable to Meet-in-the-Middle (MitM) attacks. With the set of possible diffusion layers (see next paragraph), we evaluated that at least 10 rounds would be needed to resist MitM attacks. Hence, a first rough evaluation indicated that we need about 12 to 14 rounds.

The set of substitution layers. 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. For the generation of `uKNIT-BC`, 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. The mutation in this case allows the input (resp. output) bits to swap positions.

Remark. We initially 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. The mutation was set to allow bit, linear, affine permutation (which stay within the AE class) and also allowing a swap of indices ($S[i] \longleftrightarrow S[j]$ for some $i, j \in \{0, ..., 15\}$). While this approach enabled us to design ciphers with slightly lower latency but (much) stronger security properties, we decided to place more emphasis on even lower latency candidates. Furthermore, the complexity of these ciphers made them too challenging for our automated tools to analyze, especially when we have to evaluate a large number of candidates.

The set of linear layers. The depth of the linear circuit is a critical factor for 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 using exactly 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. In our design, we considered cases where $p = 0,\ 1$, and 2 [2]. With that in mind, we use four 16×16 binary matrices that are adapted from that of `PRINCE` and `MIDORI`. These matrices can be found in Appendix D. For initialization, we implemented the following steps:

1. Let $M \xleftarrow{\$} \{\tilde{M}^{(0)}_{\texttt{PRINCE}}, M_{\texttt{MIDORI}}\}$
2. Let $n \xleftarrow{\$} \{1, ..., 1000\}$ and for n times,
   (a) Let $a \xleftarrow{\$} \{0, 1\}$ to indicate row or column swap.
   (b) Let $b \xleftarrow{\$} \{0, 1, 2, 3\}$
   (c) Randomly choose two rows (resp. columns) with the indices in $\{4b, 4b+1, 4b+2, 4b+3\}$ in M to swap
3. Repeat Steps 1 and 2 for each of the four 16-bit sub-matrices.

To ensure the various cells are mixed thoroughly, we also implemented the ShiftRows operation identical to that of `PRINCE` after the matrix multiplication for all the candidates:

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

For mutation, we have to view the linear layer as a 64×64 binary matrix. The mutation function 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.

Remark. Please note that all the matrices are orthogonal (i.e. satisfying the property that $M^T M = I$) and all operations carried out in the mutation still ensure that this property is preserved. Note that while the operation in mutation carries a high probability of disrupting the branch number, our experiments show that as long as the number of swaps is kept low, the result candidates often demonstrate enhanced security, particularly in terms of their best differential characteristic.

3.2 The fitness computation

During the search, we need to evaluate the fitness of the current pool of candidates. In particular, we are interested in their performance (latency) and the security aspects.

Security. 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. Using each candidate's specification, we automatically convert them into SAT models that describe their differential and linear properties. These models are then analyzed by a SAT solver. The modeling method we used follows the techniques described in [SWW21]. For

[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.

this purpose, we use the SAT solver `CaDiCaL` [BFF+24], but our model is compatible with any SAT solver that accepts DIMACS format.

Remark. In some cases where the number of computations required during the security evaluations is too heavy, it might be useful to set an upper bound on the number of rounds we are analyzing. For example, an upper bound can be chosen as the number of rounds that an attacker uses as a distinguisher for a key recovery attack. In other words, we can choose to only study "windows" of rounds from the cipher candidates, instead of the entire cipher. 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 ith l-length window is the l consecutive rounds of C starting from the ith round, where $0 \leq i < r$ and $1 \leq l \leq r - i$. The ith l-length window is denoted by $\mathcal{W}(i, l)$.

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

Latency. To get a good measurement of the latency of the cipher, we used `OpenLane` [SE20], an open-source automated RTL to GDSII flow that leverages other available open-source CAD tools such as Yosys [Wol], a set of optimization scripts and synthesis strategies for latency and area and an open-source PDK known as Skywater 130nm [3]. Unlike other open-source PDKs, Skywater 130nm has been silicon-proven both academically and commercially.

Overall. 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:

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

Remark. While it may look more intuitive to have the above expression to be $\frac{security}{latency}$, 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 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 is the one that produces candidates that are consistent with our goals. One notable expression we have attempted was to give a penalty whenever the latency of a candidate comes close to that of `PRINCEv2`. However, it does not yield good candidates. Nevertheless, we do not claim that this would be the optimal fitness computation, and users should change this fitness computation to suit their respective goals.

3.3 Selection and Breeding

Selection. This part of the pipeline involves the selection of ciphers to survive or be eliminated. There are two components: the so-called `fittest ciphers`, $\mathcal{C}_{\text{fit}}$, and the `breeding ciphers`, $\mathcal{C}_{\text{breed}}$ which includes some diversity into the mix. $\mathcal{C}_{\text{fit}}$ is simply the top 10 ciphers (sorted by fitness). These ciphers are retained for the next generation. Additionally, each of them will also have a mutated copy of these ciphers introduced into the next generation. $\mathcal{C}_{\text{breed}}$ is a little hard to compute as we have to introduce the concept of *diversity*. This notion is explained in Appendix A. Then, $\mathcal{C}_{\text{breed}}$ is computed as follows:

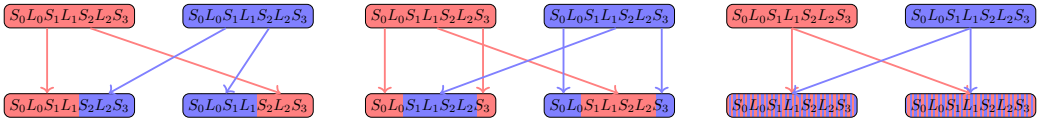
[3] https://github.com/google/skywater-pdk

1. Initialize the breeding set: $\mathcal{C}_{\text{breed}} \leftarrow \varnothing$
2. For each cipher c in the population, $\mathcal{C}$, compute Diversity$(c, \mathcal{C}_{\text{breed}})$
3. Normalize fitness and diversity scores[4]
4. Compute the score for each cipher:

$$\text{score}(c) = a\,\text{fitness}(c) + b\,\text{diversity}(c), \quad \text{with } a = 1,\ b = 10$$

5. Select the cipher with the highest score and add it to $\mathcal{C}_{\text{breed}}$
6. Repeat from Step 2 until $\mathcal{C}_{\text{breed}}$ contains 40% of the population

Breeding. There are three types of breeding we implement: single-point crossover, double-point crossover and uniform crossover as seen in Figure 3.



(a) Single point (b) Double point (c) Uniform

Figure 3: Various types of crossover.

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 follows:

1. Select two candidates $c, c' \in \mathcal{C}_{\text{breed}}$ uniformly at random.
2. Apply crossover between c and c' using one of the following:
   - single-point crossover with probability 0.4,
   - double-point with probability 0.4,
   - uniform with probability 0.2.
3. Then, with a probability of 0.05, apply mutation to the resulting offspring. If the offspring already exists in the set of generated candidates, increase the mutation probability to 1.
4. Repeat this process until a total of 200 offspring have been generated.

Remark. The probabilities are chosen after some trial and error and we tend to favor single-point and double-point as they preserve most of the cipher's structure.

3.4 Parameters and optimization concerns

In this subsection, we discuss some of the parameter choices and introduce methods to reduce the computational load that we have tested at some point, which proves to be quite useful at times.

Population size. A population size of 200 was selected in this particular case to ensure a diverse range of ciphers while keeping the total evaluation time manageable.

[4]We rescale both fitness and diversity to the range $[0, 1]$.

Number of generations. The number of generations is set to 100. Once again, this is set according to the amount of computation power we have.

Computational power. In our setup, we are using a server with a dual-socket AMD EPYC 7401 machine with 48 physical cores (96 logical threads). To optimize resource usage, we limited execution to 90 threads and the computation took approximately seven days to complete.

Smart brute-force. This method is only useful after sufficiently strong x-round candidates have been obtained. Instead of evaluating a total of $\sum_n (N_{gen_n} \times N_{pop_n})$ ciphers, where N_{gen_n} and N_{pop_n} is the number of generations and population size at round n, we just explore a lot of different possible extensions (by 1 round) by choosing from the pool of good substitution and linear layers. This is equivalent to setting $N_{gen_n} = 1$ and thus, we can have more N_{pop_n}. The term *smart* works like this: against differential and linear cryptanalysis, suppose we have an x-round cipher, we can look at the previous best differentia characteristic/linear trail, to choose the next linear layer such that it minimizes the differential probability/linear correlation among all possible linear layer candidates.

3.5 Backdoor concerns

A potential concern when using `uKNIT` for cipher design is the possibility of intentionally introducing hidden weaknesses or backdoors while attributing the resulting construction to the search procedure itself. For example, a malicious designer could selectively publish only candidates exhibiting hidden structural properties favorable to the designer yet difficult to detect through automated security evaluation. In some cases, a cipher containing a deliberate backdoor could even be manually designed and subsequently claimed to have been generated by `uKNIT` in an attempt to conceal its origin and evade suspicion.

To mitigate such risks, we emphasize two important principles. First, the entire search process should be reproducible. In particular, the random seeds, initialization parameters, mutation rules, fitness functions, and overall search configuration should be made publicly available whenever possible. This allows independent researchers to reproduce and verify the generation process. Second, candidates produced by the optimization framework must still undergo extensive independent cryptanalysis. We stress that the use of `uKNIT` does not replace the need for rigorous security evaluation. It merely serves as a tool for exploring the design space. As with any new symmetric-key primitive, confidence in the security of a generated cipher can only be established through broad public scrutiny and thorough analysis against a wide range of cryptanalytic techniques.

4 The key schedule

Jean *et al.* [JNP14] proposed a generalized construction that builds 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 [CJPS24], 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 in another proposal [NSS20]. In [KYB24], 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 $i \in \{0, 1, \ldots, r\}$, 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 $\mathbb{F}_2$: $G_0, G_1, \ldots, G_r$, such that $K_i = G_i K_m$. We call the set of matrices $\mathbf{G} = \{G_0, G_1, \ldots, G_r\}$ a linear key schedule.

Given Definition 2, we can now give a stronger goal for the key schedule, but that 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 $0 \leq i_0 < i_1 < \ldots < i_{p-1} \leq r$, $[\, G_{i_0}^T \, G_{i_1}^T \, \cdots \, G_{i_{p-1}}^T \,]$ 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 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 takes 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 pth 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 $G_2, \ldots, G_r$ 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 $A_2,\ldots,A_r$ and $B_2,\ldots,B_r$, 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

$$0 \neq \det\left(\begin{bmatrix} A^i & B^i \\ A^j & B^j \end{bmatrix}\right) = \det(A^i)\det(B^j - A^jA^{-i}B^i) =$$

$$\det(A^i)\det(B^j - A^{j-i}B^i) = \det(A^i)\det(B^{j-i} - A^{j-i})\det(B^i) \tag{1}$$

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 $\mathrm{GF}(2^b)$, where b is typically 4 or 8. We also define $B = AC$, where $C = cI$, for $c \in \mathrm{GF}(2^b)$ and $c \neq 0$ and $c^i \neq 1\ \forall\ 1 \leq r$. We can show that this is sufficient since C commutes with any matrix: $AC = AcI = cAI = cIA = CA$. Thus,

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

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 [Tri10, 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 the key schedule to 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 $A \in \mathrm{GF}(2^4)^{4\times 4}$ 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 5.4). 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 $G = [P \circ M, P \circ M \circ L_{LFSR}]$ (L_{LFSR} 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 5.4.

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 3 of Section 6.1.

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. In Appendix B, we give the specification of uKNIT-BC. In Appendix C, we also give an alternate representation of the cipher. We also provide a GitHub repository containing the implementation and test vectors here:

https://github.com/syllab-ntu/UKNIT.git

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: a substitution layer and a linear layer. The final round contains only the substitution layer. Figure 4 describes the overview of uKNIT-BC. In our description, each n-bit variable is a bit column vector indexed $0, 1, \ldots, n-1$ 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.

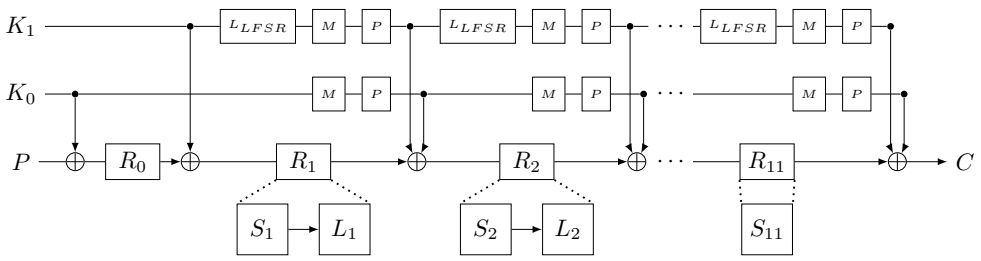


Figure 4: The structure of uKNIT-BC including the key schedule.

5.2 Substitution layers

The 4-bit S-boxes applied to each nibble of the internal state in uKNIT-BC can be represented by $(D \circ S_{\texttt{MANTIS}} \circ B)(x)$ where $S_{\texttt{MANTIS}}$ is the MANTIS S-box:

$$S_{\texttt{MANTIS}}[x] = [\texttt{c}, \texttt{a}, \texttt{d}, 3, \texttt{e}, \texttt{b}, \texttt{f}, 7, 8, 9, 1, 5, 0, 2, 4, 6]$$

and D and B are transposition matrices given in Table 6 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 9 and 10 of Appendix B.2 provide the list of these indices.

5.4 Key schedule

The key schedule of uKNIT-BC is depicted in Figure 4. 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

$$K_0 \leftarrow P \circ M(K_0), \quad K_1 \leftarrow P \circ M \circ L_{LFSR}(K_1), \quad K_i \leftarrow K_0 \oplus K_1,$$

where L_{LFSR} is a linear operation that applies the SKINNY-64 LFSR to each nibble of K_1. $L_{LFSR} : (x_3||x_2||x_1||x_0) \rightarrow (x_2||x_1||x_0||x_3 \oplus x_2)$. 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 ith nibble.

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

5.5 Decryption of uKNIT-BC

As the circuits of uKNIT-BC encryption and decryption are different, we recommend using uKNIT-BC for modes of operation that do not require much decryption. However, since the starting points of our genetic search algorithm are matrices of PRINCE and MANTIS, the decryption function for the data path of uKNIT-BC also enjoys the low-latency feature. Inherently, the inverse S-boxes are still the MANTIS S-box with some bit permutations and the inverse functions of the linear layers in uKNIT-BC ensure that each output bit is obtained by performing two XOR operations on three input bits. Therefore, the depths of the inverse functions of the linear layers in uKNIT-BC are also 2 layers.

Like other SPN ciphers, the decryption function of uKNIT-BC is also executed round by round in reverse order, applying the inverse functions of the substitution and linear layers. Since the round function of each round in uKNIT-BC is different, we also need to describe the inverse functions of the substitution layer and the linear layer for each round. To correspond with the encryption function, we describe them in order from the first round to the last round.

Inverses of substitution layers. The inverse functions of the 4-bit S-boxes are applied to the internal state of uKNIT-BC. Since each 4-bit uKNIT-BC S-box can be represented as $(D \circ S_{\texttt{MANTIS}} \circ B)(x)$ (D and B are provided in Table 6) and $S_{\texttt{MANTIS}}$ is involutory, the inverse of each S-box is represented as $(B^{-1} \circ S_{\texttt{MANTIS}} \circ D^{-1})(x)$. B^{-1} and D^{-1} are easily obtained from Table 6, as listed in Table 7.

Inverses of linear layers. The inverses of linear layers are also sparse 64×64 binary matrices applied to the internal state. Each row of these matrices contains three "1" with the rest being "0". Thus, we describe these inverse linear layers in a similar way with these indices. Tables 11 and 12 of Appendix B.2 provide the list of these indices.

Remark on the transparency of uKNIT-BC. The seed used to generate uKNIT-BC was not preserved, as we only realized the importance of retaining it after this issue was raised by the reviewers during the review process[5]. Unfortunately, by that time, the initial seed was no longer available. We emphasize that no backdoor was inserted and that no manual adjustment was made beyond the design process described in this paper. Nevertheless, we conducted an extensive security analysis of the cipher in Section 6. Furthermore, despite the lack of strong alignment among the components used in uKNIT-BC, each component is still derived from primitives with well-established cryptographic properties. We hope that these observations provide additional confidence in the soundness, transparency, and security of the overall design.

6 Security analysis of uKNIT-BC

In this section, we exhibit the extensive state-of-the-art cryptanalysis we performed on uKNIT-BC, see in Table 1 the best key-recovery attacks we obtained.

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

Attack	No. of rounds	Targeted Win.	Time	Data	Reference
Differential	10	$\mathcal{W}(0, 10)$	$2^{110.6}$ ops.	$2^{55.6}$ CP	Section 6.1
Impossible differential	10	$\mathcal{W}(1, 10)$	2^{98} ops.	2^{63} CP	Section 6.4
Demirci-Selçuk MitM	9	$\mathcal{W}(0, 9)$	2^{115} ops.	2^{61} CP	Section 6.7
Differential-linear	9	$\mathcal{W}(1, 9)$	$2^{92.7}$ ops.	$2^{53.7}$ CP	Section 6.9

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 [NIS18], 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: simple and multiple differential cryptanalysis (Sections 6.1 and 6.2), linear cryptanalysis (Section 6.3), impossible differential cryptanalysis (Section 6.4), zero-correlation cryptanalysis (Section 6.5), integral attacks (Section 6.6), Demirci-Selçuk meet-in-the-middle attack (Section 6.7), boomerang and rectangle attacks (Section 6.8),

[5]We thank the reviewers for raising this issue.

(higher-order) differential-linear attacks (Section 6.9), invariant attacks (Section 6.10), slide attacks and internal differential attacks (Section 6.11). 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 for cryptanalysis. We introduce a few more notations to facilitate the descriptions of our attacks. The S-boxes and linear layers used in the ith-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 ith-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 $z[t_0, t_1, t_2, \ldots, t_{n-1}]_b$ are the n bits of z as $z[t_j]_b, 0 \leq t \leq n-1$. Similarly, $z[t]_n$ is the t-th nibble of z, while $z[t_0, t_1, \ldots, t_{n-1}]_n$ are the n nibbles of $z[t_j]_n, 0 \leq j \leq n-1$. When $t_0, \ldots, t_{n-1}$ 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 Simple differential cryptanalysis

In this subsection, we analyze the security of uKNIT-BC against differential attacks [BS90]. 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 $\mathcal{W}(i, r)$, $1 \leq r \leq 12$ 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).

Even though we do not claim resistance against related-key attacks, we provide in Table 3 the related-key differentials probabilities we found on uKNIT-BC.

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 \ 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 \ 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, with a higher number of rounds, the freedom to combine different components enabled uKNIT-BC to achieve stronger differential security. Generally speaking, a weaker component is associated with better latency performance, which helps explain, to some extent, why uKNIT-BC is better than PRINCE in both the security and latency criteria.

For windows of length 8, all probabilities are lower than 2^{-64}. We also verified with the quasi-differential technique [BR22], the best differential characteristic for 8 rounds and concluded that its success probability 2^{-68} is independent of the key value. As a result, we expect that 8-round uKNIT-BC does not have any useful DCs for key-recovery attack.

Table 3: The probabilities, in $-\log_2$, of the related-key differentials of `uKNIT-BC` for $\mathcal{W}(i,r)$ for $1 \le r \le 12$ and $0 \le i \le 12-r$. The numbers in brackets mean that it is a lower bound.

r \ 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)	–	–	–	–	–	–	–	–	–	–	–

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 E for the model), we check $\mathcal{W}(0 \le i \le 1, 11)$ with 7-round DC and 2-round backward and forward propagations. However, no effective key-recovery attack was detected. We then check $\mathcal{W}(0 \le i \le 2, 10)$ 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 5 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 round-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. Under the linear key schedule, these 120 bits have rank 112 over the 128-bit master key. Thus, we initialize a rank-compressed counter $\mathbb{H}$ with 2^{112} entries.

2. One plaintext structure contains 2^{48} plaintexts. Encrypt plaintexts of s structures and combine them to $s \times 2^{95}$ pairs. Since the ciphertext difference has 7 inactive S-boxes, thus only $s \times 2^{95-28} = s \times 2^{67}$ pairs survive.

3. For each surviving pair, we guess all 2^{36} 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. The first 2^{112-a} key candidates with largest occurrence times are selected. Each rank-112 candidate leaves $128-112=16$ master-key bits undetermined. Thus, the final exhaustive verification contains $2^{112-a} \cdot 2^{16} = 2^{128-a}$ master-key candidates.

Complexity. From s structures, $s \times 2^{95}$ pairs can be obtained. The probability they can reach $\Delta(y_1)$ is 2^{-48}, *i.e.*, $s \times 2^{47}$ pairs will reach $\Delta(y_1)$. According to [Sel08], the number of pairs entering the distinguisher can be computed as

$$N_{pairs} = \frac{(\Phi^{-1}(P_S)\sqrt{S_N+1} + \Phi^{-1}(1-2^{-a}))^2}{pS_N},$$

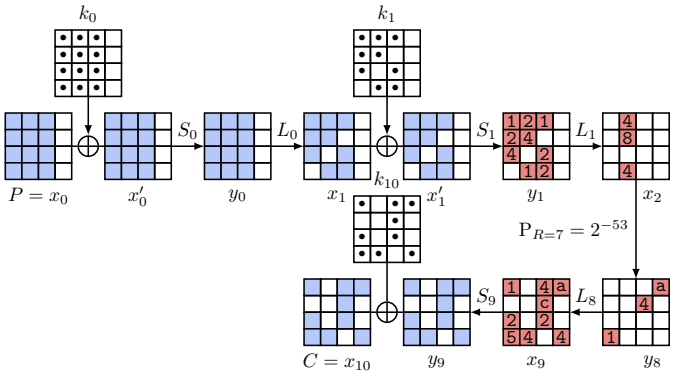


Figure 5: The differential key-recovery attack on $\mathcal{W}(0, 10)$ based on a DC of $\mathcal{W}(2, 7)$ (red color) whose probability is 2^{-53}.

where P_S is the success probability, Φ is CDF of the standard normal distribution, $p = 2^{-53}$ is the probability of the DC, $S_N = p/2^{-n}$ is the signal-to-noise ratio, and a is the advantage. We set $P_S = 0.95$ and $a = 20$, $N_{pairs} \approx 2^{54.6}$. Thus $2^{7.6}$ structures are needed. The final data complexity is about $2^{55.6}$ chosen plaintexts. After the ciphertext filtering, about $2^{7.6+67} = 2^{74.6}$ pairs remain, and each pair requires enumerating 2^{36} values of $\Delta(x_1[0 \sim 2, 4, 5, 7, 8, 10, 11]_n)$. The online processing therefore costs about $2^{74.6+36} = 2^{110.6}$ operations. The final exhaustive verification costs $2^{128-20} = 2^{108}$ encryptions, which is dominated by the online processing. The overall time complexity is about $2^{110.6}$ operations, and the memory complexity is dominated by the 2^{112} counters of $\mathbb{H}$.

Verifying the differential cryptanalysis with quasi-differential techniques. The classical differential cryptanalysis relies on assumptions such as the hypothesis of stochastic equivalence and round-independence assumptions. In reality, the differential cryptanalysis works under the fixed-key scenario, thus both assumptions might fail. To analyze the differential propagation property under the fixed-key setting, Beyne and Rijmen introduced the quasi-differential framework [BR22]. In this framework, the quasi-differential basis is chosen to generate the quasi-differential transition matrix.

Definition 4 (Quasi-differential transition matrix [BR22])**.** Let n and m be positive integers and $\mathsf{F} : \mathbb{F}_2^n \to \mathbb{F}_2^m$ a function. The coordinate of the quasi-differential transition matrix D^{F} of F is computed by

$$D^{\mathsf{F}}_{(v_0,v_1),(u_0,u_1)} = 2^{-n} \sum_{\substack{x \in \mathbb{F}_2^n \\ \mathsf{F}(x)+\mathsf{F}(x+u_1)=v_1}} (-1)^{u_0^\top x + v_0^\top \mathsf{F}(x)}. \tag{2}$$

(u_1, v_1) is the difference pair and (u_0, v_0) is the mask pair. The value $D^{\mathsf{F}}_{(v_0,v_1),(u_0,u_1)}$ is called the correlation. When $u_0 = v_0 = 0$, one can find $D^{\mathsf{F}}_{(v_0,v_1),(u_0,u_1)}$ represents the probability of F with the input difference u_1 and output difference v_1.

Let F be an iterative function as $\mathsf{F} = \mathsf{F}_r \circ \cdots \circ \mathsf{F}_1$. When fixing an r-round DC $(u_0, u_1, \ldots, u_r)$ for F, there might be many quasi-differential trails related to this DC, denoted by $((v_0, u_0), (v_1, u_1), \ldots, (v_r, u_r))$. The probability of a DC under the fixed-key setting can be computed as

$$p = \sum_{\omega_r, \omega_{r-1}, \ldots, \omega_0} \prod_{0 \le i < r} D^{\mathsf{F}}_{(\omega_{i+1}, u_{i+1}),(\omega_i, u_i)}.$$

Thus, we can use this formula to check if or how a DC is influenced by the secret keys.

We use the quasi-differential framework to verify the 8-round DC in Figure 6, as this DC for $\mathcal{W}(2,8)$ is the strongest among all 8-length windows. Since the S-boxes for each position of each round are different in uKNIT-BC, we need to generate the corresponding propagation tables according to each S-box S and its input difference u_1 and output difference v_1.

$$Q^{S}_{v_0,u_0} = 2^{-n} \sum_{\substack{x\in\mathbb{F}_2^n \\ \mathsf{S}(x)+\mathsf{S}(x+u_1)=v_1}} (-1)^{u_0^\top x + v_0^\top \mathsf{S}(x)}.$$

For a linear layer L, the propagation rule for the input/output mask pair (u_0, v_1) is $u_0 = \mathsf{L}^\top v_0$, which is the same as the linear cryptanalysis.

The corresponding automatic search model for masks can be established commonly. In our case, we focus our attention on the weakest 8-round windows, which is the DC from $\mathcal{W}(2,8)$ that can be found in Figure 6. We found that there is only one quasi-differential trail with all masks being 0. This shows that the probability of this DC is independent of keys.

6.2 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

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

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

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

In [CFG+14], 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 [BEK+20]. 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 6 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 $2 \le i < 9$, 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 uth column and v^{th} row, denoted by $M^{L_i}_{v,u}$, is defined as

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

where $\sup(b)$ 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}

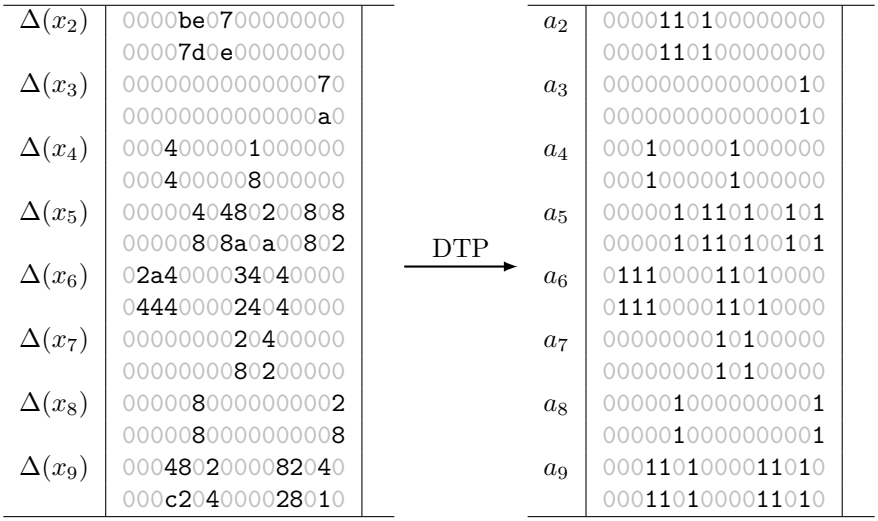


Figure 6: Generation of a DTP from a DC of $\mathcal{W}(2,8)$. For convenience, we also give the differences and patterns of $\Delta(y_i)$.

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

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

Therefore, using the same principles mentioned in [CFG+14], 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 6, 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`.

6.3 Linear (hull) cryptanalysis

Linear cryptanalysis [Mat93] 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 $\mathcal{W}(i,r)$ are 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 this 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 Section 6.2. 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 6 (Expected linear potential (ELP) [Nyb94])**.** 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

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

where $\mathrm{C}(\Gamma(x_i),\Gamma(x_{i+1}),\ldots,\Gamma(x_{i+r}))$ 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

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

The data complexity of linear cryptanalysis based on the ELP should be proportional to $\mathcal{O}(\mathrm{ELP}^{-1})$. Thus, the 8-round linear hulls should be hard to use in the key-recovery attack, considering our data limitation of 2^{-47} chosen-plaintexts.

6.4 Impossible differential cryptanalysis

Impossible differential (ID) cryptanalysis was proposed independently by Knudsen [Knu98] and Biham *et al.* [BBS99], remaining one of the most powerful attacks on block ciphers. The automatic search methods for ID distinguishers were developed from the DC searches [ST17, CJF+16]. 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 an inactive S-box by 0, the 2 impossible differentials are

$$\begin{aligned}(0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1) &\overset{7R}{\not\to} (1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0) \\ (0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1) &\overset{7R}{\not\to} (0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0)\end{aligned} \tag{3}$$

We use the first impossible differentials (Equation (3)) in a key-recovery attack on $\mathcal{W}(1,10)$. Two rounds are added before the distinguisher and one round after. The key-recovery attack process is shown in Figure 7. 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 $\Delta(x_{11})$ with probability 1. The active cells are shown in Figure 7. 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 7, 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 12 inactive cells on the ciphertext side, $2^{119-48} = 2^{71}$ pairs survive. For 2^t structures, there are 2^{t+71} pairs.

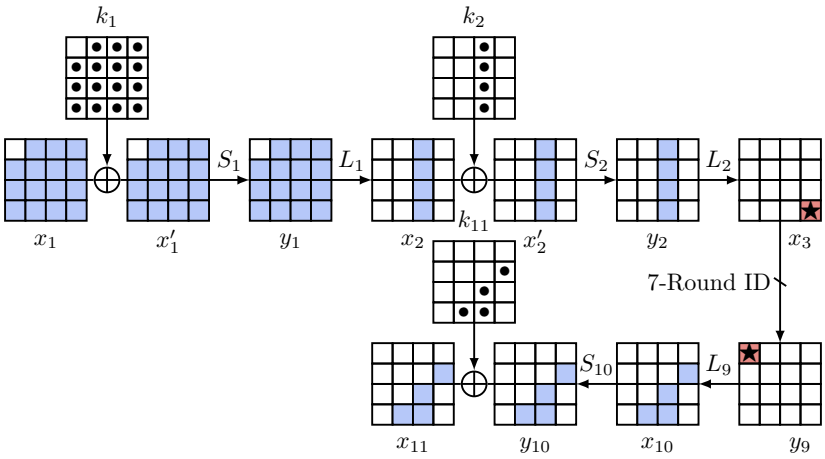


Figure 7: The key-recovery process for $\mathcal{W}(1, 10)$ based on the first ID of $\mathcal{W}(3, 7)$ (Equation (3)).

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.

4. 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+71} pairs, the probability that one wrong key cannot be excluded by all pairs is $(1 - 2^{-68})^{2^{t+71}} = (1 - 2^{-68})^{2^{68} \times 2^{-68} \times 2^{t+71}} \approx e^{-2^{t+3}}$. As we want the number of remaining wrong keys to be less than 1, we have

$$2^{92} \times e^{-2^{t+3}} < 1,$$

which means $t \geq 3$. In other words, we need 2^{63} plaintexts and process $2^3 \times 2^{60 \times 2 - 1 - 48} = 2^{74}$ pairs.

Complexity. The data complexity is 2^{63} 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 $2^{74+24} = 2^{98}$ encryptions. The memory complexity is dominated by $\mathbb{H}$, which is 2^{92} bits that is equivalent to 2^{86} 64-bit blocks.

6.5 Zero-correlation cryptanalysis

Zero-correlation (ZC) attacks are an extension of linear cryptanalysis based on linear hull with exact zero correlation (ZC linear hull) [BR14, BLNW12, BW12]. The automatic search methods are similar to those for ID, which was first proposed in [CJF+16].

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 an inactive S-box by 0, the 2 ZC linear hulls are as follows,

$$(0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1) \overset{7R}{\not\to} (1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0)$$
$$(0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1) \overset{7R}{\not\to} (0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0)$$

They are the same as the above two IDs, which is not surprising as all matrices of `uKNIT-BC` satisfy $M = (M^T)^{-1}$. For SPN ciphers, the ZC attacks are usually weaker than ID attacks. Thus, we believe the ZC attacks will not be a threat to `uKNIT-BC`.

6.6 Integral attacks

The integral attacks [KW02] were first known as *Square attack* as it was proposed for attacking the block cipher `SQUARE` [DKR97], and it was also studied under different names such as *saturation attack* [Luc01] or *multiset attacks* [BS01]. Nowadays, the most effective way to search for integral properties is the division property method and its automatic search tools [Tod15, TM16, XZBL16, BC16, HSWW20, BV23].

By modeling the 2-subset bit-based division property [TM16] with SAT solvers and setting the input division vector as all one but zero at one bit, the longest zero-sum property we have detected is for 7 rounds of `uKNIT-BC`. We did not find any zero-sum properties for all $\mathcal{W}(i, 8)$, $0 \leq i \leq 4$. We also considered the round keys using the monomial prediction [HSWW20, HE22] to search for possible one-sum properties, but found nothing for windows of length 8.

From the relationship between the zero-correlation linear hulls and the integral properties [SLR+15, BBW14], we derive a better integral distinguisher with less data, i.e.,

$$\overline{\{60, 61, 62, 63\}} \xrightarrow{\mathcal{W}(3,7)} \{0, 1, 2, 3\}, \ \overline{\{60, 61, 62, 63\}} \xrightarrow{\mathcal{W}(3,7)} \{12, 13, 14, 15\}$$

With a 7-round integral distinguisher, we believe integral attacks will not threaten the security of the 12-round `uKNIT-BC`, as adding forward propagation rounds is more difficult than the differential and linear cases.

6.7 Demirci-Selçuk meet-in-the-middle attack

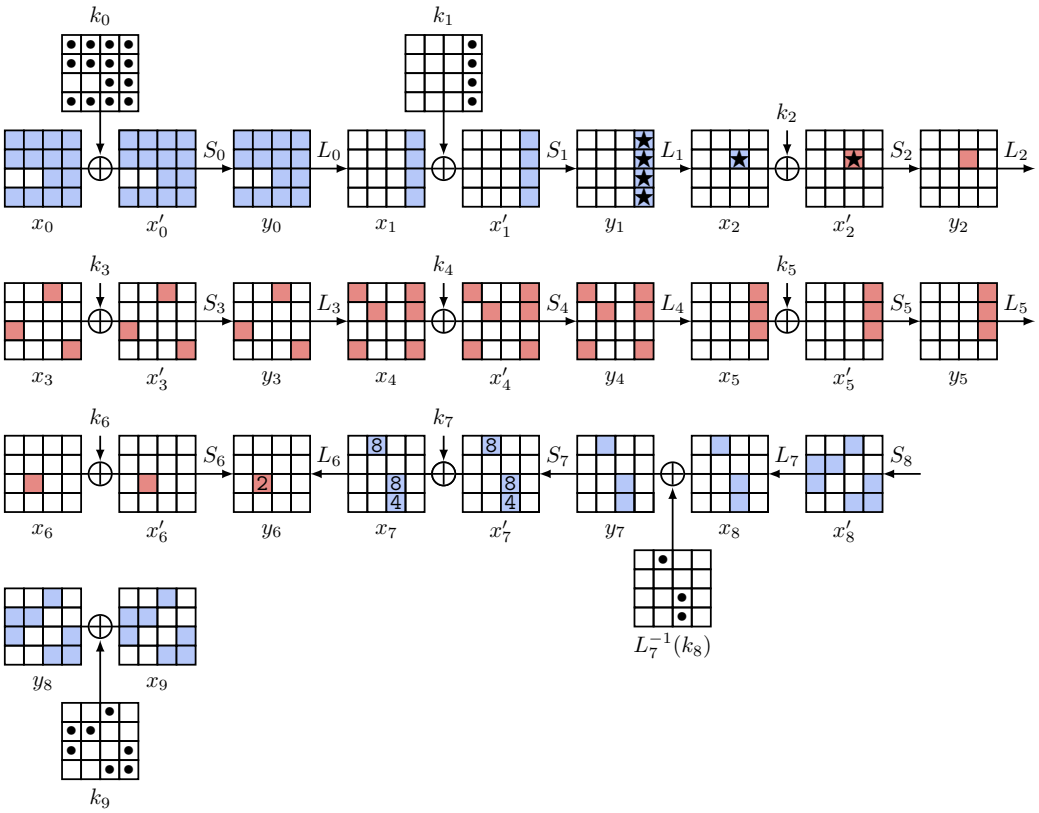


Figure 8: The DS-MITM attack process for $\mathcal{W}(0, 9)$. The distinguisher is applied to $\mathcal{W}(2, 5)$ and labeled in red, while the online phase is labeled in blue.

The Demirci-Selçuk Meet-in-the-Middle (DS-MitM) attack was proposed by Demirci and Selçuk as a new attack on `AES` [DR02] in 2008 [DS08]. Later, this attack was significantly enhanced by Dunkelman *et al.* [DKS10b] and eventually became the current best attack on `AES-128` thanks to Derbez *et al.* [DFJ13]. Several tools have been proposed to search for DS-MitM attacks for different ciphers. Some are designed with general-purpose programming languages such as [DF13, DF16], others are built upon the automatic search tools such as [SSD+18, SSS+23]. We mainly implemented the methods in [SSD+18] to search for DS-MitM attacks on `uKNIT-BC`.

As shown in Figure 8, the best attack we detect covers $\mathcal{W}(0,9)$ with a distinguisher on $\mathcal{W}(2,5)$. Both the backward and forward propagations in the key-recovery process cover 2 rounds. Before we introduce this attack, we give the definition of δ-set [DS08] in our context, which is a core notion in DS-MitM attacks.

Definition 7 (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 $X_0, X_1, \ldots, X_{15}$. The total number of possible 60-bit ordered sequences*

$$f(X_0) \oplus f(X_1), f(X_0) \oplus f(X_2), \ldots, f(X_0) \oplus f(X_{15}) \tag{4}$$

is fully determined by the 56-bit parameters,

$$x'_2[9]_n,\ x'_3[2,8,15]_n,\ x'_4[0,3,5,12,13,15]_n,\ x'_5[12 \sim 14]_n,\ x'_6[6]_n \tag{5}$$

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

With the *differential enumeration technique* [DKS10b, DFJ13], the number of parameter bits in Equation (5) can be reduced to 36 bits.

Observation 2 (Differential enumeration technique [DKS10b, DFJ13])**.** *Suppose that we have a truncated differential $\Delta(x'_2) \to \Delta(y_6)$ where $\Delta(x'_2)[9]_n = \bigstar$ (any value except `0`) and $\Delta(y_6)[6]_n = \mathtt{2}$. The 56-bit parameters can be determined by the following 36-bit differences,*

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

As we have 2^{60} possibilities of Equation 4 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^{36} 60-bit sequences of Equation 4 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 4.

1. As shown in Figure 8, 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 $\Delta(y_6)[6]_n = 2$ with a probability of 2^{-28}.

In total, the probability of obtaining 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} candidate pair, at the plaintext side, we guess all possible 2^{16} $\Delta(x_1)$ and 15 $\Delta(x_2)$ (as $\Delta(x_2) \neq 0$), then extract on average one candidate of $k_0[0, 1, 3 \sim 5, 7 \sim 15]_n$ and $k_1[12 \sim 15]_n$ per guess. 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 $k_9[1, 2, 5, 8, 11, 14, 15]_n$ per guess.

4. Suppose that we select $x_2'[9]_n$, 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. Use the corresponding 16 ciphertexts and the guessed k_8 and k_9 bits to decrypt them to y_6, and compute the sequence of Equation (4).

5. Check if the calculated sequence is in $\mathbb{H}$ built in the offline phase. The sequence has 2^{56} possibilities (4 bits being fixed by the truncated differential $\Delta(y_6)[6]_n = 2$), and $|\mathbb{H}| = 2^{36}$, so each (pair, key candidate) tuple passes the test with probability $2^{36-56} = 2^{-20}$. Per pair, the guesses in the previous step yield $2^{16} \cdot 15 \cdot 2^{12} \approx 2^{32}$ key candidates. In total,
$$2^{80} \cdot 2^{32} \cdot 2^{-20} = 2^{92}$$
surviving (pair, key candidate) tuples remain. Each one specifies the following key bits:
$$k_0[0, 1, 3 \sim 5, 7 \sim 15]_n \text{ (56 bits)}, \quad k_1[12 \sim 15]_n \text{ (16 bits)},$$
$$L_7^{-1}(k_8)[4, 10, 11]_n \text{ (12 bits)}, \quad k_9[1, 2, 5, 8, 11, 14, 15]_n \text{ (28 bits)}.$$

6. Since the key schedule of `uKNIT-BC` is linear in the 128-bit master key $K_m = K_0 \| K_1$, the $56 + 16 + 12 + 28 = 112$ recovered bits translate to 112 linear equations on K_m. A generic rank-112 system leaves 2^{16} master-key candidates per surviving tuple, hence $2^{92} \cdot 2^{16} = 2^{108}$ candidate master keys in total. Each candidate is verified against one known plaintext-ciphertext pair (2^{-64} filter), and the survivors against a second pair, which leaves only the correct key.

Complexity. The data complexity is the $2^5 \cdot 2^{56} = 2^{61}$ plaintexts encrypted. The memory complexity is dominated by the storage of $\mathbb{H}$, namely 2^{36} 60-bit sequences, which is $2^{41.9}$ bits or $2^{35.9}$ 64-bit blocks. The offline phase costs $2^{36+4} = 2^{40}$ 5-round encryptions to populate $\mathbb{H}$. For the online phase, the dominant cost is partially encrypting the δ-set for every (pair, key candidate) tuple before the $\mathbb{H}$ test: this gives $15 \times 2^{80+16+12+4} \approx 2^{116}$ 4-round encryptions, equivalent to approximately 2^{115} 9-round encryptions. The subsequent exhaustive verification of the 2^{108} candidate master keys requires 2^{108} encryptions. The overall time complexity is therefore 2^{115} 9-round encryptions.

6.8 Boomerang and rectangle attacks

The boomerang attack was proposed in 1999 as a variant of differential cryptanalysis [Wag99]. 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 $p_2 + p_3 = \alpha$. 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 [KKS00, BDK01]. 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 [Mur11]. Several refinements have been proposed to mitigate such independence problems [DKS10a, DKS14, CHP+18]. The boomerang connectivity table (BCT) is one of the latest technique among them [CHP+18].

Definition 8 (Boomerang connectivity table (BCT) [CHP+18])**.** For an n-bit S-box, denoted by S, its BCT is a 2-dimensional $2^n \times 2^n$ table, with the entry $\mathrm{BCT}(\delta, \delta')$ being

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

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 $\alpha \to \alpha'$ and a good differential of E_2, say $\beta' \to \beta$. For $E_1 = S_0||S_1||\cdots||S_{15}$ where S_i are the 16 S-boxes, we have the following computation

$$\begin{aligned} &\mathrm{P}(E_1^{-1}(E_1(x) \oplus \beta) \oplus E_1^{-1}(E_1(x \oplus \alpha') \oplus \beta) = \alpha') \\ &\quad = \prod_{0 \le i < 16} \mathrm{P}\left(S_i^{-1}(S_i(x[i]_n) \oplus \beta[i]_n) \oplus S_i^{-1}(S_i(x[i]_n \oplus \alpha'[i]_n) \oplus \beta[i]_n) = \alpha'[i]_n\right) \end{aligned} \quad (6)$$

For each S-box, the right side of the equation is computed using its BCT. Denote the probability of E_1 calculated from Equation (6) 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 a 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, the best rectangle distinguisher detected by us has an estimated probability $2^{-64} \times 2^{-77} = 2^{-141}$ for $\mathcal{W}(1, 8)$. The best rectangle distinguisher we found for a 7-round window is one for $\mathcal{W}(2, 7)$ with probability of $2^{-64} \times 2^{-51} = 2^{-115}$. Both rectangle distinguishers are illustrated in Figure 9. As stated in [SYC+24, YSZ+24], the data complexity for a rectangle attack is

$$D = \frac{\sqrt{s}2^{\frac{n}{2}+1}}{\sqrt{p}},$$

where s is the number of right quartets, n is the block length and p is the boomerang probability (rather than the rectangle probability). Thus, both rectangle distinguishers cannot work within the data limitation of 2^{-47} chosen-plaintexts, even if we consider $s = 1$.

$\Delta(x_1)$	0000092820240000	2^{-12}	
	0000042180880000		
$\Delta(x_2)$	0000000000000041	2^{-4}	
	0000000000000088		Upper
$\Delta(x_3)$	0000000009000000	2^{-3}	Diff.
	0000000002000000		
$\Delta(x_4)$	0000010040000002	2^{-6}	
	0000010040000008		
$\Delta(x_5)$	0024214041010010	2^{-3}	BCT
	000000000228c119		
$\Delta(x_6)$	0023000000000000	2^{-4}	
	008a000000000000		
$\Delta(x_7)$	0000000000100000	2^{-2}	Lower
	0000000000100000		Diff.
$\Delta(x_8)$	0080000080000080	2^{-6}	
	0010000020000002		
Boom. Pro. $= 2^{77}$ (expl. 2^{-14})			

$\Delta(x_2)$	0000000018400000	2^{-6}	
	0000000048100000		
$\Delta(x_3)$	0000000080000000	2^{-2}	Upper
	0000000010000000		Diff.
$\Delta(x_4)$	00c0000000000004	2^{-4}	
	0050000000000002		
$\Delta(x_5)$	3000002000260200	2^{-1}	BCT
	a3c0804100000000		
$\Delta(x_6)$	6100000000000000	2^{-5}	
	1300000000000000		
$\Delta(x_7)$	8000000000000000	2^{-2}	Lower
	2000000000000000		Diff.
$\Delta(x_8)$	0000000100400200	2^{-6}	
	0000000800800400		
Boom. Pro. $= 2^{-51}$ (expl. 2^{-10})			

Figure 9: The upper and lower DCs and connectivity of the rectangle distinguishers for $\mathcal{W}(1,8)$ (left) and $\mathcal{W}(2,7)$ (right). The expl. values are the sum of the experimental probability of the 3 rounds with the BCT round in the middle.

6.9 (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 (DL) cryptanalysis, proposed by Langford and Hellman in 1994 [LH94], 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

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

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 $\delta \to \delta'$, with a probability p, and a LC for E_1 say $\lambda' \to \lambda$, with a correlation q. The correlation can be estimated as pq^2 [LH94]. 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 [BDKW19]. The DLCT is defined as follows,

Definition 9 (Differential-linear connectivity table (DLCT) [BDKW19])**.** For an n-bit S-box denoted by S, its DLCT is a 2-dimensional $2^n \times 2^n$ table, with entry $\mathrm{DLCT}(\delta,\lambda)$ being

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

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 $\mathcal{W}(2,7)$. With 2^{28} messages under 100 randomly-chosen keys, we verified the correlation of the middle

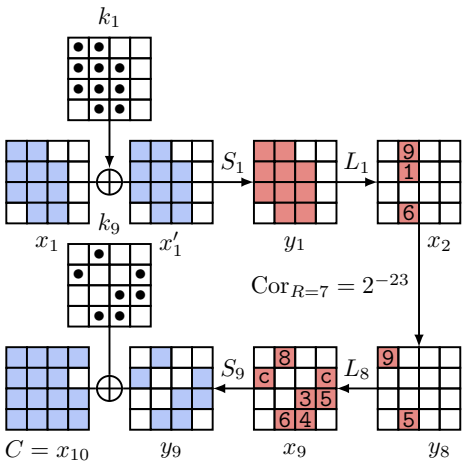


Figure 10: The key-recovery process for $\mathcal{W}(1, 9)$ based on the 7-round DL distinguisher of $\mathcal{W}(2, 7)$ in Figure 11 (right).

4 rounds (from `0004000001000000` to `2100000000000000`). The average experimental correlation is about 2^{-11}. Thus, the correlation for the full 7-round DL distinguisher is about $c = 2^{-8-11-2\times2} = 2^{-23}$, which is potential to use in a key-recovery attack. After propagating the input difference (resp. output mask) backwards (resp. forwards) 1 round, we obtain a key-recovery process illustrated in Figure 10. The active patterns in this figure are obtained from the right-hand DL distinguisher in Figure 11, with $\Delta(x_2) =$ `0000910600000000` and $\Gamma(y_8) =$ `9000000500000000`. On the ciphertext side, applying L_8 to $\Gamma(y_8)$ gives $\Gamma(x_9) =$ `0c00800600340c50`, so the guessed key cells are $k_9[1, 4, 7, 10, 11, 13, 14]_n$. To compute these seven cells from ciphertexts, the inverse of the last linear layer L_9 requires the projection $C[0 \sim 12, 14, 15]_n$ (15 nibbles, only nibble 13 being inactive).

According to [Sel08], the number of pairs we need for the key-recovery attack is computed as

$$D_{pairs} = \frac{(\Phi^{-1}(1 - 2^{-(a+1)}) + \Phi^{-1}(P_S))^2}{|c|^2},$$

where Φ is the CDF of the standard normal distribution, a is the advantage, P_S is the success probability, and c is the DL correlation. We set the success probability as 0.95, then

$$D_{pairs} = (\Phi^{-1}(1 - 2^{-(a+1)}) + 1.65)^2 \times 2^{46}.$$

We will use structures for the key-recovery attack. Propagating $\Delta(x_2) =$ `0000910600000000` one round backwards activates 10 nibbles of x_1 (specifically $x_1[0 \sim 2, 4 \sim 7, 9 \sim 11]_n$), so each plaintext structure has size 2^{40} and yields $\binom{2^{40}}{2} \approx 2^{79}$ pairs. Since L_1 is bijective, the constraint $\Delta(x_2) =$ `0000910600000000` pins $\Delta(y_1) = L_1^{-1}(\Delta(x_2))$ to a single 40-bit value within the 10-active-nibble subspace, so a random pair satisfies it with probability 2^{-40}, leaving $2^{79-40} = 2^{39}$ surviving pairs per structure. Thus, the number of structures we need is

$$s = D_{pairs} 2^{-39},$$

thus the data complexity is $D = s \times 2^{40}$.

The key-recovery process is as follows:

Online phase.

1. Initialize a counter array indexed by the following 68-bit tuple:
$$\big(k_1[0 \sim 2, 4 \sim 7, 9 \sim 11]_n,$$
$$k_9[1, 4, 7, 10, 11, 13, 14]_n\big);$$
2. Prepare $s \times 2^{40}$ plaintexts, where the 2^{40} plaintexts of each structure are obtained by varying $x_1[0 \sim 2, 4 \sim 7, 9 \sim 11]_n$ in Figure 10;
3. Query the encryption oracle on the $s \times 2^{40}$ plaintexts to obtain the corresponding ciphertexts;
4. For each of the 2^{40} values of $k_1[0 \sim 2, 4 \sim 7, 9 \sim 11]_n$:
   (a) Partially encrypt the $s \times 2^{40}$ plaintexts through S_1, L_1 under this k_1 guess, form pairs, and keep only the $s \times 2^{39}$ pairs whose intermediate difference equals $\Delta(x_2)$;
   (b) For each surviving pair (C, C') and each candidate value of $k_9[1, 4, 7, 10, 11, 13, 14]_n$, compute the parity $\Gamma(x_9) \cdot (x_9 \oplus x_9')$ by partially inverting L_9 and S_9 at the 7 active nibbles, and increment (resp. decrement) the counter at (k_1, k_9) if the parity agrees (resp. disagrees). The contribution of each pair decomposes as a nibble-wise XOR over the 7 active S-boxes of S_9, so the 2^{28} key-dependent parity computations are organized by the partial-sum technique [FKL+00]; in the complexity estimate below, the amortized ciphertext-side work is counted as linear in the number of surviving pairs.
5. Select the top 2^{68-a} key tuples by absolute counter value, combine them with the remaining $128 - 68$ master-key bits, and exhaustively search.

Time complexity. With partial-sum amortization in step 4(b), the dominant cost is
$$T \approx \max\big\{D,\ 2^{40} \cdot 2^{39} \cdot s,\ 2^{128-a}\big\}.$$
When $a = 54$, $D_{pairs} \approx 2^{52.65}$, giving time complexity $2^{92.65}$ operations and data complexity $2^{53.65}$ chosen plaintexts. The memory complexity is dominated by the 2^{68} counters.

However, for 8 rounds, the best DL distinguisher we detect has a correlation of 2^{-35}, as shown in Figure 11 (left). We experimentally verified the middle three rounds (from `e000000000d00700` to `0000000000008041`) with 2^{26} messages, and the correlation without the last diffusion layer is about $2^{-9.83}$. Combining with the DC and LC parts outside the middle gives a full 8-round correlation $c \approx 2^{-6-9.83-16} \approx 2^{-31.83}$. The data complexity of a DL attack scales as $\Omega(1/c^2) \approx 2^{63.66}$, and even targeting an advantage as small as $a = 1$ (with $P_S = 0.95$) requires $D_{pairs} \approx (\Phi^{-1}(0.75) + 1.65)^2 \cdot 2^{63.66} \approx 2^{66.1}$, already exceeding the full 2^{64} codebook of `uKNIT-BC`. Hence the 8-round DL approximation poses no threat to `uKNIT-BC`.

Higher-order differential cryptanalysis. Higher-order differential-linear (HDL) cryptanalysis was first proposed by Biham *et al.* to study possible combined attacks [BDK05]. In [HPTY23], the authors used the HATF technique inspired by the ATF methods [LLL21] 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 equivalent to the `MANTIS` S-box), we believe `uKNIT-BC` resists HDL attacks.

6.10 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 [LAAZ11], Leander *et al.*

$\Delta(x_3)$	0000000000260000 0000000000120000	2^4	Diff.
$\Delta(x_4)$	00c0000000000000 00b0000000000000	2^2	
$\Delta(x_5)$	e000000000d007000 5000000000e001000	2^7	
$\Delta(x_6)$	90b10d8001500060e 0000210488022024	2^0	DLCT
$\Delta(x_7)$	0000000000004048 0000000000008041	2^3	Linear
$\Delta(x_8)$	0000000000004000 0000000000008000	2^1	
$\Delta(x_9)$	0001000008008000 0008000002004000	2^3	
$\Delta(x_{10})$	0000064041000000 00000a3021000000	2^4	
DL Cor. $= 2^{-35}$ (expl. $2^{-31.83}$)			

$\Delta(x_2)$	0000910600000000 0000930300000000	2^6	Diff.
$\Delta(x_3)$	0000000000000090 00000000000000a0	2^2	
$\Delta(x_4)$	0004000001000000 0004000008000000	2^4	
$\Delta(x_5)$	0000040480200808 6a6000000442000	2^1	DLCT
$\Delta(x_6)$	3010000000000000 1020000000000000	2^2	Linear
$\Delta(x_7)$	8800000000000000 2100000000000000	2^2	
$\Delta(x_8)$	8000000100000000 9000000500000000	2^2	
DL Cor. $= 2^{-25}$ (expl. 2^{-23})			

Figure 11: The DC and LC and connectivity part of the DL distinguishers for $\mathcal{W}(3,8)$ (left) and $\mathcal{W}(2,7)$ (right).

introduced the invariant subspaces and in [TLS16] Todo *et al.* presented the nonlinear invariants. Beierle *et al.* studied how to choose the constants to resist invariant-type attacks in [BCLR17]. Later, in [Bey18] Beyne showed that the invariant attacks can be effectively explained by the eigenvectors of correlation matrices [DGV94]. As discussed in [BNP24, 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, variant attacks are much more difficult to find. With our key schedule based on the gSTK, we believe that invariant attacks are not a threat to the security of `uKNIT-BC`.

6.11 Slide attacks and internal differential attacks

The slide attack and its variants [BW99, BW00, DKS15, GLP08] explore the symmetry of different round functions. While the internal differential attacks [Pey10, DDS13, ZHL24, ZHL23] use 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.

7 Implementations and benchmarks

General comparison. We synthesized the different ciphers for the TSMC 65nm technology library using a design compiler. For each cipher, we derived two measurements: the first one is a combinational unconstrained circuit as a baseline. For the second result, we place each cipher between input and output registers and set the target clock period to $1ns$. For all the ciphers tested, this goal is unattainable. However, the synthesis algorithm outputs the closest netlist it could reach to this goal, at the expense of a larger area. In both measurements, 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` [BEK+20]. 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 fairer comparison to also include the area gain when only the

maximum achievable frequency of PRINCEv2 is targeted. We also include the results with the key schedule and with precomputed round keys (key side loading). The results are given in Table 4. In the comparison, we include 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 [ABC+24] and Orthros [BIL+21] 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 F we discuss how to build a PRF from uKNIT-BC, not even considering using less rounds. BipBip [BDD+23a] has a 24-bit block, yet its latency is not that much better than uKNIT-BC (less than 3%). Qarmav1 [Ava17] is a TBC with a large block size and is slower than uKNIT-BC. Among BCs, we have implemented MIDORI and MANTIS with 7 rounds on either side of the middle round (with no tweak and similar security claim to PRINCEv2 and uKNIT-BC). Both of these ciphers are more costly than PRINCEv2, but we included them in the table for completeness. SPEEDY [LMMR21] is a 192-bit BC, which comes in three variants differing mainly in the number of rounds; 5, 6 and 7 rounds. The large block size makes the area very large for all variants, and impacts its usability in applications that require a small block size. uKNIT-BC is significantly faster than the 7-round variant and moderately faster than the 6-round variant, while using only a fraction of the area. The 5-round variant is faster than most other ciphers on the list. However, a few points must be noted:

1. The design goals of the different ciphers are different and the head-to-head comparison is not straightforward.

2. The uKNIT framework can be applied to wide block sizes, such as 192 bits. Although we leave that for future work, in the next section we show examples of the applicability of the uKNIT framework to other use cases than uKNIT-BC. It is likely that the larger block size gives more degrees of freedom, and we can capture this in the uKNIT framework.

3. The security of 5-round SPEEDY has been under scrutiny, with a very slim security margin [BDBN23, BN24]. In addition, while the 7-round variants achieve much higher security, their security claims have been broken in [WNL+23].

Last but not least, the candidate that is closest to uKNIT-BC in terms of security model and block size is PRINCEv2. *The ideal way to assess the power of the* uKNIT *framework is to compare its outcome* uKNIT-BC *against the best cipher with the same goal, application and parameters (key and block sizes), and such BC happens to be* PRINCEv2. uKNIT-BC has $\approx 10\%$ lower latency, and for the lowest latency of PRINCEv2, we need only half the area with precomputed round keys ($\approx 18\%$ 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 a frequency, 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

Table 4: 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 a target latency of $1ns$. For `KoalaP` [ABD+24], the first row is just the permutation, while the second row is the latency, which 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` [ABC+24]	128	3.45	73,078.92
		128	1.61	133,343.99
	`Orthros` [BIL+21]	128	2.66	40,932.36
		128	1.59	77,437.08
TBC	`BipBip` [BDD+23a]	24	4.03	39,278.52
		24	1.45	60,630.12
	`MANTIS` [BJK+16]	64	3.39	13,359.60
		64	2.01	31,020.48
	`Qarmav1` 9 rnds [Ava17]	128	4.84	42,787.08
		128	2.74	94,944.23
Public Perm.	`KoalaP` [ABD+24]	64	1.46	24,104.88
		64	1.16	52,965.36
BC	`MIDORI` [BBI+15]	64	3.46	13,367.16
		64	2.05	30,475.44
	`SPEEDY` 5 rnds [LMMR21]	192	2.63	32,565.96
		192	1.26	66,251.16
	`SPEEDY` 6 rnds [LMMR21]	192	3.13	39,681.36
		192	1.52	77,387.40
	`SPEEDY` 7 rnds [LMMR21]	192	3.75	46,826.64
		192	1.79	88,331.04
	`PRINCEv2` [BEK+20]	64	2.90	12,006.72
		64	1.65	27,564.12
	`uKNIT-BC` (precomputed keys)	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
	`uKNIT-BC` Decryption (precomputed keys)	64	2.50	9,842.75
		64	1.64	14,664.60
		64	1.49	23,095.08

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. We emphasize that we do not set the key to 0, we only instruct the synthesizer to consider the scenario that the key does not change frequently. 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 15 (top). The results indeed show the subtleties of comparing power consumption.

In the medium range of frequency ($250 \sim 500$ MHz), `uKNIT-BC` experiences approximately $25\% \sim 35\%$ improvement without the key schedule and approximately $8\% \sim 19\%$ with the 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 15 (mid) includes the power results, while Table 15 (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.

Combined encryption and decryption. We acknowledge that it is cheaper for PRINCEv2 to have a combined encryption-decryption circuit, due to its reflection property. However, the area of a combined encryption-decryption circuit for uKNIT-BC is close to $40,000\mu m^2$ compared to $27,000\mu m^2$ for PRINCEv2, at the maximum frequency possible by PRINCEv2. Although some may view this as a limitation, we believe that whether such an implementation is needed depends heavily on the application and mode of operation. We do not recommend using a simple block-cipher mode of operation with a 64-bit block cipher, but we recommend using the XoP construction (see Appendix F), which requires two encryption circuits rather than encryption-decryption. In this use case, PRINCEv2 is actually more expensive. In addition, we believe that reaching lower latency is worth the area sacrifice for the right applications, and we view the fact that, for the same frequency, the combined encryption-decryption circuit is still less than twice PRINCEv2 as positive.

8 Discussion

8.1 Alternative use-cases

Although we have applied the uKNIT framework specifically to the design of low-latency ciphers, its capabilities extend beyond this particular application. By simply replacing the security-latency criterion with another metric, the framework can be adapted for different cipher design objectives (potentially again mixing performance and security objectives). Additionally, although this paper uses the uKNIT framework for a 64-bit low-latency cipher design, its potential is much greater than this, which also applies to designing other cipher sizes.

As a proof of concept, we applied our genetic algorithm in the uKNIT framework to SPECK-32 and SPECK-128 [BSS+13], allowing rotation values to vary between rounds. Different rotation values lead to a cipher with about the same performance per round as the original SPECK, but the security is greatly improved.

We achieve this by setting the initial rotation values to be the same as SPECK. However, with each generation, the rotation at each round has a probability of increasing or decreasing by 1. Table 5 (left) compares the probabilities of the best differential characteristics obtained when using different round rotations versus the original cipher. The results show a significant security improvement, as measured by the probability of the best differential characteristic, when different rotations are used for each round. Note that SPECK designers mention in [BSS+17] an extra criterion on the rotation values, by minimizing the distance

Table 5: **Probability of the best differential characteristic** of SPECK-32 and SPECK-128 to variants that use different **rotation values** each round (left table); and the **minimum number of active S-boxes** in AES to a variant uses different **byte-permutation** each round (right table). The rotation and byte-permutations are provided in Appendix G.

Cipher		Rounds				
		5	6	7	8	9
SPECK-32	orig.	9	13	18	24	30
	diff.	12	19	25	30	-
SPECK-128	orig.	10	15	21	-	-
	diff.	12	21	-	-	-

Cipher		Rounds			
		3	4	5	6
AES	orig.	9	25	26	30
	diff.	9	25	28	34

to a multiple of 8 (for faster performance on certain micro-controllers. If we also take this criterion into account, we could find rotation values on 8 rounds that have a distance lower than that of SPECK (thus potentially leading to increased performances), while also improving security bounds.

We applied the same concept to AES [DR02]. Although evaluating the differential characteristics of AES is highly time-consuming, the security of AES (in terms of a lower bound on the differential probability) can be assessed using the wide-trail strategy. Similarly, we can modify our evaluation function to incorporate the wide-trail strategy. In this case, we varied the ShiftRows (to a more generic byte-permutation) operation for each round. The results are presented in Table 5 (right).

8.2 Time complexity and optimality

We emphasize that we do not claim that our design strategy leads to optimal constructions. However, the uKNIT framework indeed provides a way to find good candidates from a very large search space (much larger than the original same-round cipher design paradigms). One may argue that it is unsurprising that using different round functions can lead to better performance, however, no work has paid attention to exploring such design possibilities for the full cipher (one can mention the ARX-based S-box of Alzette [BBdS+20]). The uKNIT framework takes the first main step in this direction: we are the first to take full advantage of this design idea to design a full-fledged cipher.

The uKNIT framework provides a structured and guided approach to exploring this search space and identifying improved designs. Since the framework evaluates ciphers' strength based on various metrics, its time complexity is directly proportional to the time required to evaluate a single cipher (for performance and security), with the multiplier depending on the chosen hyperparameters. Consequently, if a design is infeasible to evaluate using existing methods due to time constraints, uKNIT will also be unable to evaluate it.

During the early stages of this research, we encountered candidates that exhibited exceptionally strong security-to-latency ratios. However, due to the extensive time required to evaluate even a single cipher, we had no choice but to abandon them in favor of alternatives whose security could be feasibly assessed.

It is important to note also that the uKNIT framework is not limited to evaluating security based on the best differential characteristic. Any assessable security metric could be used. As demonstrated with AES, the security metric can be adapted, like calculating the minimum number of active S-boxes, offering an alternative evaluation approach for faster search if time complexity becomes an issue.

9 Conclusion

In this paper, we introduced a new design framework called `uKNIT`, aimed at creating ciphers round-by-round with various performance/security objectives and breaking the round alignment paradigm that prevailed in the field. To showcase that excellent cipher candidates do exist within this gigantic search space, we proposed a genetic search algorithm for the case of SPN schemes and we instantiated a specific example focused on creating an ultra-low latency block cipher, `uKNIT-BC`. 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 a cipher's design process. Automation enables us to explore uncharted territories more effectively, paving the way for innovative cryptographic schemes.

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. This work was partially supported by the Wallenberg AI, Autonomous Systems and Software Program (WASP) funded by the Knut and Alice Wallenberg Foundation. This work was also partially funded by the European Union ERC-2023-COG, SoBaSyC, 101125450. Views and opinions expressed are however those of the author(s) only and do not necessarily reflect those of the European Union or the European Research Council Executive Agency. Neither the European Union nor the granting authority can be held responsible for them. Additionally, the first author is supported by the National Cryptologic Science Fund of China (2025NCSF02007), the National Natural Science Foundation of China (62402283), Shandong Provincial Natural Science Foundation (No.2025HWYQ-025), the Natural Science Foundation of Jiangsu Province (BK20240420), Program of TaiShan Scholars Special Fund for young scholars (Grant NO.tsqn202507063) and Program of Qilu Young Scholars of Shandong University.

References

[ABC+24] Ravi Anand, Subhadeep Banik, Andrea Caforio, Tatsuya Ishikawa, Takanori Isobe, Fukang Liu, Kazuhiko Minematsu, Mostafizar Rahman, and Kosei Sakamoto. Gleeok: A Family of Low-Latency PRFs and its Applications to Authenticated Encryption. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2024(2):545–587, 2024. doi:10.46586/TCHES.V2024.I2.545-587.

[ABD+23] Roberto Avanzi, Subhadeep Banik, Orr Dunkelman, Maria Eichlseder, Shibam Ghosh, Marcel Nageler, and Francesco Regazzoni. The QARMAv2 Family of Tweakable Block Ciphers. *IACR Trans. Symmetric Cryptol.*, 2023(3):25–73, 2023. doi:10.46586/TOSC.V2023.I3.25-73.

[ABD+24] Parisa Amiri-Eliasi, Yanis Belkheyar, Joan Daemen, Santosh Ghosh, Daniël Kuijsters, Alireza Mehrdad, Silvia Mella, Shahram Rasoolzadeh, and Gilles Van Assche. Koala: A Low-Latency Pseudorandom Function. *IACR Cryptol. ePrint Arch.*, page 1249, 2024. URL: https://eprint.iacr.org/2024/1249.

[ADM24] Roberto Avanzi, Orr Dunkelman, and Kazuhiko Minematsu. MATTER: A Wide-Block Tweakable Block Cipher. *IACR Cryptol. ePrint Arch.*, page 1186, 2024. URL: https://eprint.iacr.org/2024/1186.

[ARS+15] Martin R. Albrecht, Christian Rechberger, Thomas Schneider, Tyge Tiessen, and Michael Zohner. Ciphers for MPC and FHE. In Elisabeth Oswald and Marc Fischlin, editors, *Advances in Cryptology - EUROCRYPT 2015*, volume 9056 of *LNCS*, pages 430–454. Springer, 2015. doi:10.1007/978-3-662-46800-5_17.

[Ava17] Roberto Avanzi. The QARMA Block Cipher Family. Almost MDS Matrices Over Rings With Zero Divisors, Nearly Symmetric Even-Mansour Constructions With Non-Involutory Central Rounds, and Search Heuristics for Low-Latency S-Boxes. *IACR Trans. Symmetric Cryptol.*, 2017(1):4–44, 2017. doi:10.13154/TOSC.V2017.I1.4-44.

[BAK98] Eli Biham, Ross J. Anderson, and Lars R. Knudsen. Serpent: A New Block Cipher Proposal. In Serge Vaudenay, editor, *Fast Software Encryption, 5th International Workshop, FSE '98, Paris, France, March 23-25, 1998, Proceedings*, volume 1372 of *Lecture Notes in Computer Science*, pages 222–238. Springer, 1998. doi:10.1007/3-540-69710-1_15.

[BBdS+20] Christof Beierle, Alex Biryukov, Luan Cardoso dos Santos, Johann Großschädl, Léo Perrin, Aleksei Udovenko, Vesselin Velichkov, and Qingju Wang. Alzette: A 64-Bit ARX-box - (Feat. CRAX and TRAX). In Daniele Micciancio and Thomas Ristenpart, editors, *Advances in Cryptology - CRYPTO 2020*, volume 12172 of *LNCS*, pages 419–448. Springer, 2020. doi:10.1007/978-3-030-56877-1_15.

[BBI+15] Subhadeep Banik, Andrey Bogdanov, Takanori Isobe, Kyoji Shibutani, Harunaga Hiwatari, Toru Akishita, and Francesco Regazzoni. Midori: A Block Cipher for Low Energy. In Tetsu Iwata and Jung Hee Cheon, editors, *Advances in Cryptology - ASIACRYPT 2015*, volume 9453 of *LNCS*, pages 411–436. Springer, 2015. doi:10.1007/978-3-662-48800-3_17.

[BBS99] Eli Biham, Alex Biryukov, and Adi Shamir. Cryptanalysis of Skipjack Reduced to 31 Rounds Using Impossible Differentials. In Jacques Stern, editor, *Advances in Cryptology - EUROCRYPT '99*, volume 1592 of *LNCS*, pages 12–23. Springer, 1999. doi:10.1007/3-540-48910-X_2.

[BBW14] Céline Blondeau, Andrey Bogdanov, and Meiqin Wang. On the (In)Equivalence of Impossible Differential and Zero-Correlation Distinguishers for Feistel- and Skipjack-Type Ciphers. In Ioana Boureanu, Philippe Owesarski, and Serge Vaudenay, editors, *Applied Cryptography and Network Security - ACNS 2014*, volume 8479 of *LNCS*, pages 271–288. Springer, 2014. doi:10.1007/978-3-319-07536-5_17.

[BC16] Christina Boura and Anne Canteaut. Another View of the Division Property. In Matthew Robshaw and Jonathan Katz, editors, *Advances in Cryptology - CRYPTO 2016*, volume 9814 of *LNCS*, pages 654–682. Springer, 2016. doi:10.1007/978-3-662-53018-4_24.

[BCG+12] Julia Borghoff, Anne Canteaut, Tim Güneysu, Elif Bilge Kavun, Miroslav Knezevic, Lars R. Knudsen, Gregor Leander, Ventzislav Nikov, Christof Paar, Christian Rechberger, Peter Rombouts, Søren S. Thomsen, and Tolga Yalçin. PRINCE - A Low-Latency Block Cipher for Pervasive Computing Applications - Extended Abstract. In Xiaoyun Wang and Kazue Sako, editors, *Advances in Cryptology - ASIACRYPT 2012*, volume 7658 of *LNCS*, pages 208–225. Springer, 2012. doi:10.1007/978-3-642-34961-4_14.

[BCLR17] Christof Beierle, Anne Canteaut, Gregor Leander, and Yann Rotella. Proving Resistance Against Invariant Attacks: How to Choose the Round Constants. In Jonathan Katz and Hovav Shacham, editors, *Advances in Cryptology - CRYPTO 2017*, volume 10402 of *LNCS*, pages 647–678. Springer, 2017. doi:10.1007/978-3-319-63715-0_22.

[BDBN23] Christina Boura, Nicolas David, Rachelle Heim Boissier, and María Naya-Plasencia. Better Steady than Speedy: Full Break of SPEEDY-7-192. In Carmit Hazay and Martijn Stam, editors, *Advances in Cryptology - EUROCRYPT 2023*, volume 14007 of *LNCS*, pages 36–66. Springer, 2023. doi:10.1007/978-3-031-30634-1_2.

[BDD+23a] Yanis Belkheyar, Joan Daemen, Christoph Dobraunig, Santosh Ghosh, and Shahram Rasoolzadeh. BipBip: A Low-Latency Tweakable Block Cipher with Small Dimensions. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2023(1):326–368, 2023. doi:10.46586/TCHES.V2023.I1.326-368.

[BDD+23b] Yanis Belkheyar, Joan Daemen, Christoph Dobraunig, Santosh Ghosh, and Shahram Rasoolzadeh. Introducing two Low-Latency Cipher Families: Sonic and SuperSonic. *IACR Cryptol. ePrint Arch.*, page 878, 2023. URL: https://eprint.iacr.org/2023/878.

[BDG+25] Yanis Belkheyar, Patrick Derbez, Shibam Ghosh, Gregor Leander, Silvia Mella, Léo Perrin, Shahram Rasoolzadeh, Lukas Stennes, Siwei Sun, Gilles Van Assche, and Damian Vizár. ChiLow and ChiChi: New Constructions for Code Encryption. In Serge Fehr and Pierre-Alain Fouque, editors, *Advances in Cryptology - EUROCRYPT 2025 - 44th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Madrid, Spain, May 4-8, 2025, Proceedings, Part I*, volume 15601 of *Lecture Notes in Computer Science*, pages 212–243. Springer, 2025. doi:10.1007/978-3-031-91107-1_8.

[BDK01] Eli Biham, Orr Dunkelman, and Nathan Keller. The Rectangle Attack - Rectangling the Serpent. In Birgit Pfitzmann, editor, *Advances in Cryptology - EUROCRYPT 2001*, volume 2045 of *LNCS*, pages 340–357. Springer, 2001. doi:10.1007/3-540-44987-6_21.

[BDK05] Eli Biham, Orr Dunkelman, and Nathan Keller. New Combined Attacks on Block Ciphers. In Henri Gilbert and Helena Handschuh, editors, *Fast Software Encryption - FSE 2005*, volume 3557 of *LNCS*, pages 126–144. Springer, 2005. doi:10.1007/11502760_9.

[BDKW19] Achiya Bar-On, Orr Dunkelman, Nathan Keller, and Ariel Weizman. DLCT: A New Tool for Differential-Linear Cryptanalysis. In Yuval Ishai and Vincent Rijmen, editors, *Advances in Cryptology - EUROCRYPT 2019*, volume 11476 of *LNCS*, pages 313–342. Springer, 2019. doi:10.1007/978-3-030-17653-2_11.

[BEK+20] Dusan Bozilov, Maria Eichlseder, Miroslav Knezevic, Baptiste Lambin, Gregor Leander, Thorben Moos, Ventzislav Nikov, Shahram Rasoolzadeh, Yosuke Todo, and Friedrich Wiemer. PRINCEv2 - More Security for (Almost) No Overhead. In Orr Dunkelman, Michael J. Jacobson Jr., and Colin O'Flynn, editors, *Selected Areas in Cryptography - SAC 2020*, volume 12804 of *LNCS*, pages 483–511. Springer, 2020. doi:10.1007/978-3-030-81652-0_19.

[Bey18] Tim Beyne. Block Cipher Invariants as Eigenvectors of Correlation Matrices. In Thomas Peyrin and Steven D. Galbraith, editors, *Advances in Cryptology - ASIACRYPT 2018*, volume 11272 of *LNCS*, pages 3–31. Springer, 2018. doi:10.1007/978-3-030-03326-2_1.

[BFF+24] Armin Biere, Tobias Faller, Katalin Fazekas, Mathias Fleury, Nils Froleyks, and Florian Pollitt. CaDiCaL 2.0. In Arie Gurfinkel and Vijay Ganesh, editors, *Computer Aided Verification - CAV 2024*, volume 14681 of *LNCS*, pages 133–152. Springer, 2024. doi:10.1007/978-3-031-65627-9_7.

[BGG+23a] Emanuele Bellini, David Gérault, Juan Grados, Rusydi H. Makarim, and Thomas Peyrin. Boosting Differential-Linear Cryptanalysis of ChaCha7 with MILP. *IACR Trans. Symmetric Cryptol.*, 2023(2):189–223, 2023. doi:10.46586/TOSC.V2023.I2.189-223.

[BGG+23b] Emanuele Bellini, David Gérault, Juan Grados, Rusydi H. Makarim, and Thomas Peyrin. Fully Automated Differential-Linear Attacks Against ARX Ciphers. In Mike Rosulek, editor, *Topics in Cryptology - CT-RSA 2023*, volume 13871 of *LNCS*, pages 252–276. Springer, 2023. doi:10.1007/978-3-031-30872-7_10.

[BII+25] Subhadeep Banik, Tatsuya Ishikawa, Takanori Isobe, Ryoma Ito, Kazuhiko Minematsu, Kazuma Nakata, Mostafizar Rahman, and Kosei Sakamoto. Dialga: A Family of Low-Latency Tweakable Block Ciphers Using Multiple Linear Layers. *IACR Trans. Symmetric Cryptol.*, 2025(4):70–124, 2025. doi:10.46586/TOSC.V2025.I4.70-124.

[BIL+21] Subhadeep Banik, Takanori Isobe, Fukang Liu, Kazuhiko Minematsu, and Kosei Sakamoto. Orthros: A Low-Latency PRF. *IACR Cryptol. ePrint Arch.*, page 390, 2021. URL: https://eprint.iacr.org/2021/390.

[BJK+16] Christof Beierle, Jérémy Jean, Stefan Kölbl, Gregor Leander, Amir Moradi, Thomas Peyrin, Yu Sasaki, Pascal Sasdrich, and Siang Meng Sim. The SKINNY Family of Block Ciphers and Its Low-Latency Variant MANTIS. In Matthew Robshaw and Jonathan Katz, editors, *Advances in Cryptology - CRYPTO 2016*, volume 9815 of *LNCS*, pages 123–153. Springer, 2016. doi:10.1007/978-3-662-53008-5_5.

[BLNW12] Andrey Bogdanov, Gregor Leander, Kaisa Nyberg, and Meiqin Wang. Integral and Multidimensional Linear Distinguishers with Correlation Zero. In Xiaoyun Wang and Kazue Sako, editors, *Advances in Cryptology - ASIACRYPT 2012*, volume 7658 of *LNCS*, pages 244–261. Springer, 2012. doi:10.1007/978-3-642-34961-4_16.

[BN24] Tim Beyne and Addie Neyt. Note on the cryptanalysis of Speedy. Cryptology ePrint Archive, Paper 2024/262, 2024. URL: https://eprint.iacr.org/2024/262.

[BNP24] Christina Boura and Maria Naya-Plasencia. *Symmetric Cryptography, Volume 2: Cryptanalysis and Future Directions*. John Wiley & Sons, 2024.

[BR14] Andrey Bogdanov and Vincent Rijmen. Linear hulls with correlation zero and linear cryptanalysis of block ciphers. *Des. Codes Cryptogr.*, 70(3):369–383, 2014. doi:10.1007/S10623-012-9697-Z.

[BR22] Tim Beyne and Vincent Rijmen. Differential Cryptanalysis in the Fixed-Key Model. In Yevgeniy Dodis and Thomas Shrimpton, editors, *Advances in Cryptology - CRYPTO 2022 - 42nd Annual International Cryptology Conference, CRYPTO 2022, Santa Barbara, CA, USA, August 15-18, 2022, Proceedings, Part III*, volume 13509 of *Lecture Notes in Computer Science*, pages 687–716. Springer, 2022. doi:10.1007/978-3-031-15982-4_23.

[BS90] Eli Biham and Adi Shamir. Differential Cryptanalysis of DES-like Cryptosystems. In Alfred Menezes and Scott A. Vanstone, editors, *Advances in Cryptology - CRYPTO '90*, volume 537 of *LNCS*, pages 2–21. Springer, 1990. doi:10.1007/3-540-38424-3_1.

[BS01] Alex Biryukov and Adi Shamir. Structural Cryptanalysis of SASAS. In Birgit Pfitzmann, editor, *Advances in Cryptology - EUROCRYPT 2001*, volume 2045 of *LNCS*, pages 394–405. Springer, 2001. doi:10.1007/3-540-44987-6_24.

[BSS+13] Ray Beaulieu, Douglas Shors, Jason Smith, Stefan Treatman-Clark, Bryan Weeks, and Louis Wingers. The SIMON and SPECK Families of Lightweight Block Ciphers. *IACR Cryptol. ePrint Arch.*, page 404, 2013. URL: http://eprint.iacr.org/2013/404.

[BSS+17] Ray Beaulieu, Douglas Shors, Jason Smith, Stefan Treatman-Clark, Bryan Weeks, and Louis Wingers. Notes on the design and analysis of SIMON and SPECK. Cryptology ePrint Archive, Paper 2017/560, 2017. URL: https://eprint.iacr.org/2017/560.

[BV23] Tim Beyne and Michiel Verbauwhede. Integral Cryptanalysis Using Algebraic Transition Matrices. *IACR Trans. Symmetric Cryptol.*, 2023(4):244–269, 2023. doi:10.46586/TOSC.V2023.I4.244-269.

[BW99] Alex Biryukov and David A. Wagner. Slide Attacks. In Lars R. Knudsen, editor, *Fast Software Encryption - FSE '99*, volume 1636 of *LNCS*, pages 245–259. Springer, 1999. doi:10.1007/3-540-48519-8_18.

[BW00] Alex Biryukov and David A. Wagner. Advanced Slide Attacks. In Bart Preneel, editor, *Advances in Cryptology - EUROCRYPT 2000*, volume 1807 of *LNCS*, pages 589–606. Springer, 2000. doi:10.1007/3-540-45539-6_41.

[BW12] Andrey Bogdanov and Meiqin Wang. Zero Correlation Linear Cryptanalysis with Reduced Data Complexity. In Anne Canteaut, editor, *Fast Software Encryption - FSE 2012*, volume 7549 of *LNCS*, pages 29–48. Springer, 2012. doi:10.1007/978-3-642-34047-5_3.

[CFG+14] Anne Canteaut, Thomas Fuhr, Henri Gilbert, María Naya-Plasencia, and Jean-René Reinhard. Multiple Differential Cryptanalysis of Round-Reduced PRINCE. In Carlos Cid and Christian Rechberger, editors, *Fast Software Encryption - FSE 2014*, volume 8540 of *LNCS*, pages 591–610. Springer, 2014. doi:10.1007/978-3-662-46706-0_30.

[CGL+23] Federico Canale, Tim Güneysu, Gregor Leander, Jan Philipp Thoma, Yosuke Todo, and Rei Ueno. SCARF - A Low-Latency Block Cipher for Secure Cache-Randomization. In Joseph A. Calandrino and Carmela Troncoso, editors, *USENIX Security Symposium, 2023*, pages 1937–1954. USENIX Association, 2023. URL: https://www.usenix.org/conference/usenixsecurity23/presentation/canale.

[CHNE24] Debasmita Chakraborty, Hosein Hadipour, Phuong Hoa Nguyen, and Maria Eichlseder. Finding Complete Impossible Differential Attacks on AndRX Ciphers and Efficient Distinguishers for ARX Designs. *IACR Trans. Symmetric Cryptol.*, 2024(3):84–176, 2024. doi:10.46586/TOSC.V2024.I3.84-176.

[CHP+18] Carlos Cid, Tao Huang, Thomas Peyrin, Yu Sasaki, and Ling Song. Boomerang Connectivity Table: A New Cryptanalysis Tool. In Jesper Buus Nielsen and Vincent Rijmen, editors, *Advances in Cryptology - EUROCRYPT 2018*, volume 10821 of *LNCS*, pages 683–714. Springer, 2018. doi:10.1007/978-3-319-78375-8_22.

[CJF+16] Tingting Cui, Keting Jia, Kai Fu, Shiyao Chen, and Meiqin Wang. New Automatic Search Tool for Impossible Differentials and Zero-Correlation Linear Approximations. *IACR Cryptol. ePrint Arch.*, page 689, 2016. URL: http://eprint.iacr.org/2016/689.

[CJPS24] Benoît Cogliati, Jérémy Jean, Thomas Peyrin, and Yannick Seurin. A long tweak goes a long way: High multi-user security authenticated encryption from tweakable block ciphers. *IACR Communications in Cryptology*, 1(2), 2024. doi:10.62056/a3qjp2fgx.

[DDG+23] François Delobel, Patrick Derbez, Arthur Gontier, Loïc Rouquette, and Christine Solnon. A CP-Based Automatic Tool for Instantiating Truncated Differential Characteristics. In Anupam Chattopadhyay, Shivam Bhasin, Stjepan Picek, and Chester Rebeiro, editors, *Progress in Cryptology - INDOCRYPT 2023*, volume 14459 of *LNCS*, pages 247–268. Springer, 2023. doi:10.1007/978-3-031-56232-7_12.

[DDS13] Itai Dinur, Orr Dunkelman, and Adi Shamir. Collision Attacks on Up to 5 Rounds of SHA-3 Using Generalized Internal Differentials. In Shiho Moriai, editor, *Fast Software Encryption - FSE 2013*, volume 8424 of *LNCS*, pages 219–240. Springer, 2013. doi:10.1007/978-3-662-43933-3_12.

[DEG+18] Christoph Dobraunig, Maria Eichlseder, Lorenzo Grassi, Virginie Lallemand, Gregor Leander, Eik List, Florian Mendel, and Christian Rechberger. Rasta: A cipher with low ANDdepth and few ANDs per bit. *IACR Cryptol. ePrint Arch.*, page 181, 2018. URL: http://eprint.iacr.org/2018/181.

[DEKM16] Christoph Dobraunig, Maria Eichlseder, Daniel Kales, and Florian Mendel. Practical Key-Recovery Attack on MANTIS5. *IACR Trans. Symmetric Cryptol.*, 2016(2):248–260, 2016. doi:10.13154/TOSC.V2016.I2.248-260.

[DF13] Patrick Derbez and Pierre-Alain Fouque. Exhausting Demirci-Selçuk Meet-in-the-Middle Attacks Against Reduced-Round AES. In Shiho Moriai, editor, *Fast Software Encryption - FSE 2013*, volume 8424 of *LNCS*, pages 541–560. Springer, 2013. doi:10.1007/978-3-662-43933-3_28.

[DF16] Patrick Derbez and Pierre-Alain Fouque. Automatic Search of Meet-in-the-Middle and Impossible Differential Attacks. In Matthew Robshaw and Jonathan Katz, editors, *Advances in Cryptology - CRYPTO 2016*, volume 9815 of *LNCS*, pages 157–184. Springer, 2016. doi:10.1007/978-3-662-53008-5_6.

[DFJ13] Patrick Derbez, Pierre-Alain Fouque, and Jérémy Jean. Improved Key Recovery Attacks on Reduced-Round AES in the Single-Key Setting. In Thomas

Johansson and Phong Q. Nguyen, editors, *Advances in Cryptology - EUROCRYPT 2013*, volume 7881 of *LNCS*, pages 371–387. Springer, 2013. doi:10.1007/978-3-642-38348-9_23.

[DGV94] Joan Daemen, René Govaerts, and Joos Vandewalle. Correlation Matrices. In Bart Preneel, editor, *Fast Software Encryption - FSE 1994*, volume 1008 of *LNCS*, pages 275–285. Springer, 1994. doi:10.1007/3-540-60590-8_21.

[DHT17] Wei Dai, Viet Tung Hoang, and Stefano Tessaro. Information-Theoretic Indistinguishability via the Chi-Squared Method. In Jonathan Katz and Hovav Shacham, editors, *Advances in Cryptology - CRYPTO 2017*, volume 10403 of *LNCS*, pages 497–523. Springer, 2017. doi:10.1007/978-3-319-63697-9_17.

[DKR97] Joan Daemen, Lars R. Knudsen, and Vincent Rijmen. The Block Cipher Square. In Eli Biham, editor, *Fast Software Encryption - FSE '97*, volume 1267 of *LNCS*, pages 149–165. Springer, 1997. doi:10.1007/BFB0052343.

[DKS10a] Orr Dunkelman, Nathan Keller, and Adi Shamir. A Practical-Time Related-Key Attack on the KASUMI Cryptosystem Used in GSM and 3G Telephony. In Tal Rabin, editor, *Advances in Cryptology - CRYPTO 2010*, volume 6223 of *LNCS*, pages 393–410. Springer, 2010. doi:10.1007/978-3-642-14623-7_21.

[DKS10b] Orr Dunkelman, Nathan Keller, and Adi Shamir. Improved Single-Key Attacks on 8-Round AES-192 and AES-256. In Masayuki Abe, editor, *Advances in Cryptology - ASIACRYPT 2010*, volume 6477 of *LNCS*, pages 158–176. Springer, 2010. doi:10.1007/978-3-642-17373-8_10.

[DKS14] Orr Dunkelman, Nathan Keller, and Adi Shamir. A Practical-Time Related-Key Attack on the KASUMI Cryptosystem Used in GSM and 3G Telephony. *J. Cryptol.*, 27(4):824–849, 2014. doi:10.1007/S00145-013-9154-9.

[DKS15] Orr Dunkelman, Nathan Keller, and Adi Shamir. Slidex Attacks on the Even-Mansour Encryption Scheme. *J. Cryptol.*, 28(1):1–28, 2015. doi:10.1007/S00145-013-9164-7.

[DR01] Joan Daemen and Vincent Rijmen. The Wide Trail Design Strategy. In Bahram Honary, editor, *Cryptography and Coding, 8th IMA International Conference*, volume 2260 of *LNCS*, pages 222–238. Springer, 2001. doi:10.1007/3-540-45325-3_20.

[DR02] Joan Daemen and Vincent Rijmen. *The Design of Rijndael: AES - The Advanced Encryption Standard*. Information Security and Cryptography. Springer, 2002. doi:10.1007/978-3-662-04722-4.

[DS08] Hüseyin Demirci and Ali Aydin Selçuk. A Meet-in-the-Middle Attack on 8-Round AES. In Kaisa Nyberg, editor, *Fast Software Encryption - FSE 2008*, volume 5086 of *LNCS*, pages 116–126. Springer, 2008. doi:10.1007/978-3-540-71039-4_7.

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

[GLMS20] David Gérault, Pascal Lafourcade, Marine Minier, and Christine Solnon. Computing AES related-key differential characteristics with constraint programming. *Artif. Intell.*, 278, 2020. doi:10.1016/J.ARTINT.2019.103183.

[GLP08] Michael Gorski, Stefan Lucks, and Thomas Peyrin. Slide Attacks on a Class of Hash Functions. In Josef Pieprzyk, editor, *Advances in Cryptology - ASIACRYPT 2008*, volume 5350 of *LNCS*, pages 143–160. Springer, 2008. doi:10.1007/978-3-540-89255-7_10.

[GLR+20] Lorenzo Grassi, Reinhard Lüftenegger, Christian Rechberger, Dragos Rotaru, and Markus Schofnegger. On a Generalization of Substitution-Permutation Networks: The HADES Design Strategy. In Anne Canteaut and Yuval Ishai, editors, *Advances in Cryptology - EUROCRYPT 2020*, volume 12106 of *LNCS*, pages 674–704. Springer, 2020. doi:10.1007/978-3-030-45724-2_23.

[GMW24] Patricia Greene, Mark Motley, and Bryan Weeks. ARADI and LLAMA: Low-Latency Cryptography for Memory Encryption. *IACR Cryptol. ePrint Arch.*, page 1240, 2024. URL: https://eprint.iacr.org/2024/1240.

[GWZ25] Jian Guo, Shichang Wang, and Tianyu Zhang. Breaking Full ChiLow-32. Cryptology ePrint Archive, Paper 2025/1648, 2025. URL: https://eprint.iacr.org/2025/1648.

[HE22] Hosein Hadipour and Maria Eichlseder. Integral Cryptanalysis of WARP based on Monomial Prediction. *IACR Trans. Symmetric Cryptol.*, 2022(2):92–112, 2022. doi:10.46586/TOSC.V2022.I2.92-112.

[HPTY23] Kai Hu, Thomas Peyrin, Quan Quan Tan, and Trevor Yap. Revisiting Higher-Order Differential-Linear Attacks from an Algebraic Perspective. In Jian Guo and Ron Steinfeld, editors, *Advances in Cryptology - ASIACRYPT 2023*, volume 14440 of *LNCS*, pages 405–435. Springer, 2023. doi:10.1007/978-981-99-8727-6_14.

[HSWW20] Kai Hu, Siwei Sun, Meiqin Wang, and Qingju Wang. An Algebraic Formulation of the Division Property: Revisiting Degree Evaluations, Cube Attacks, and Key-Independent Sums. In Shiho Moriai and Huaxiong Wang, editors, *Advances in Cryptology - ASIACRYPT 2020*, volume 12491 of *LNCS*, pages 446–476. Springer, 2020. doi:10.1007/978-3-030-64837-4_15.

[HWKS98] Chris Hall, David A. Wagner, John Kelsey, and Bruce Schneier. Building PRFs from PRPs. In Hugo Krawczyk, editor, *Advances in Cryptology - CRYPTO '98*, volume 1462 of *LNCS*, pages 370–389. Springer, 1998. doi:10.1007/BFB0055742.

[JN16] Jérémy Jean and Ivica Nikolic. Efficient Design Strategies Based on the AES Round Function. In Thomas Peyrin, editor, *Fast Software Encryption - FSE 2016*, volume 9783 of *LNCS*, pages 334–353. Springer, 2016. doi:10.1007/978-3-662-52993-5_17.

[JNP14] Jérémy Jean, Ivica Nikolić, and Thomas Peyrin. Tweaks and Keys for Block Ciphers: The TWEAKEY Framework. In Palash Sarkar and Tetsu Iwata, editors, *Advances in Cryptology – ASIACRYPT 2014*, pages 274–288, Berlin, Heidelberg, 2014. Springer Berlin Heidelberg. doi:10.1007/978-3-662-45608-8_15.

[JR93] Burton S. Kaliski Jr. and Matthew J. B. Robshaw. Fast Block Cipher Proposal. In Ross J. Anderson, editor, *Fast Software Encryption - 1993*, volume 809 of *LNCS*, pages 33–40. Springer, 1993. doi:10.1007/3-540-58108-1_3.

[KDGD20] Michael E. Kounavis, Sergej Deutsch, Santosh Ghosh, and David Durham. K-Cipher: A Low Latency, Bit Length Parameterizable Cipher. In *IEEE Symposium on Computers and Communications - ISCC 2020*, pages 1–7. IEEE, 2020. doi:10.1109/ISCC50000.2020.9219582.

[KKS00] John Kelsey, Tadayoshi Kohno, and Bruce Schneier. Amplified Boomerang Attacks Against Reduced-Round MARS and Serpent. In Bruce Schneier, editor, *Fast Software Encryption - FSE 2000*, volume 1978 of *LNCS*, pages 75–93. Springer, 2000. doi:10.1007/3-540-44706-7_6.

[Knu98] Lars Knudsen. DEAL-a 128-bit block cipher. *complexity*, 258(2):216, 1998. doi:10.1007/3-540-48519-8_5.

[KW02] Lars R. Knudsen and David A. Wagner. Integral Cryptanalysis. In Joan Daemen and Vincent Rijmen, editors, *Fast Software Encryption - FSE 2002*, volume 2365 of *LNCS*, pages 112–127. Springer, 2002. doi:10.1007/3-540-45661-9_9.

[KYB24] Mustafa Khairallah, Srinivasan Yadhunathan, and Shivam Bhasin. Lightweight Leakage-Resilient PRNG from TBCs Using Superposition. In Romain Wacquez and Naofumi Homma, editors, *Constructive Side-Channel Analysis and Secure Design - COSADE 2024*, volume 14595 of *LNCS*, pages 197–217. Springer, 2024. doi:10.1007/978-3-031-57543-3_11.

[LAAZ11] Gregor Leander, Mohamed Ahmed Abdelraheem, Hoda Alkhzaimi, and Erik Zenner. A Cryptanalysis of PRINTcipher: The Invariant Subspace Attack. In Phillip Rogaway, editor, *Advances in Cryptology - CRYPTO 2011*, volume 6841 of *LNCS*, pages 206–221. Springer, 2011. doi:10.1007/978-3-642-22792-9_12.

[LH94] Susan K. Langford and Martin E. Hellman. Differential-Linear Cryptanalysis. In Yvo Desmedt, editor, *Advances in Cryptology - CRYPTO '94*, volume 839 of *LNCS*, pages 17–25. Springer, 1994. doi:10.1007/3-540-48658-5_3.

[LLL21] Meicheng Liu, Xiaojuan Lu, and Dongdai Lin. Differential-Linear Cryptanalysis from an Algebraic Perspective. In Tal Malkin and Chris Peikert, editors, *Advances in Cryptology - CRYPTO 2021*, volume 12827 of *LNCS*, pages 247–277. Springer, 2021. doi:10.1007/978-3-030-84252-9_9.

[LMMR21] Gregor Leander, Thorben Moos, Amir Moradi, and Shahram Rasoolzadeh. The SPEEDY Family of Block Ciphers Engineering an Ultra Low-Latency Cipher from Gate Level for Secure Processor Architectures. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2021(4):510–545, 2021. doi:10.46586/TCHES.V2021.I4.510-545.

[LNV25] Eran Lambooij, Patrick Neumann, and Michiel Verbauwhede. Meet-in-the-Middle Attacks on Full ChiLow-32. Cryptology ePrint Archive, Paper 2025/1601, 2025. URL: https://eprint.iacr.org/2025/1601.

[Luc01] Stefan Lucks. The Saturation Attack - A Bait for Twofish. In Mitsuru Matsui, editor, *Fast Software Encryption - FSE 2001*, volume 2355 of *LNCS*, pages 1–15. Springer, 2001. doi:10.1007/3-540-45473-X_1.

[LXC+23] Yong Liu, Zejun Xiang, Siwei Chen, Shasha Zhang, and Xiangyong Zeng. A Novel Automatic Technique Based on MILP to Search for Impossible Differentials. In Mehdi Tibouchi and Xiaofeng Wang, editors, *Applied Cryptography and Network Security - ACNS 2023*, volume 13905 of *LNCS*, pages 119–148. Springer, 2023. doi:10.1007/978-3-031-33488-7_5.

[LZH+24] Huina Li, Haochen Zhang, Kai Hu, Guozhen Liu, and Weidong Qiu. AlgSAT - A SAT Method for Verification of Differential Trails from an Algebraic Perspective. In Tianqing Zhu and Yannan Li, editors, *Information Security and Privacy - ACISP 2024*, volume 14895 of *LNCS*, pages 450–471. Springer, 2024. doi:10.1007/978-981-97-5025-2_23.

[Mat93] Mitsuru Matsui. Linear Cryptanalysis Method for DES Cipher. In Tor Helleseth, editor, *Advances in Cryptology - EUROCRYPT '93*, volume 765 of *LNCS*, pages 386–397. Springer, 1993. doi:10.1007/3-540-48285-7_33.

[MKPA22] Mohammad Mahzoun, Liliya Kraleva, Raluca Posteuca, and Tomer Ashur. Differential Cryptanalysis of K-Cipher. In *IEEE Symposium on Computers and Communications - ISCC 2022*, pages 1–7. IEEE, 2022. doi:10.1109/ISCC55528.2022.9912926.

[MP13] Nicky Mouha and Bart Preneel. Towards Finding Optimal Differential Characteristics for ARX: Application to Salsa20. Cryptology ePrint Archive, Paper 2013/328, 2013. URL: https://eprint.iacr.org/2013/328.

[Mur11] Sean Murphy. The Return of the Cryptographic Boomerang. *IEEE Trans. Inf. Theory*, 57(4):2517–2521, 2011. doi:10.1109/TIT.2011.2111091.

[MWGP11] Nicky Mouha, Qingju Wang, Dawu Gu, and Bart Preneel. Differential and Linear Cryptanalysis Using Mixed-Integer Linear Programming. In Chuankun Wu, Moti Yung, and Dongdai Lin, editors, *Information Security and Cryptology - Inscrypt 2011*, volume 7537 of *LNCS*, pages 57–76. Springer, 2011. doi:10.1007/978-3-642-34704-7_5.

[Nik17] Ivica Nikolic. How to Use Metaheuristics for Design of Symmetric-Key Primitives. In Tsuyoshi Takagi and Thomas Peyrin, editors, *Advances in Cryptology - ASIACRYPT 2017*, volume 10626 of *LNCS*, pages 369–391. Springer, 2017. doi:10.1007/978-3-319-70700-6_13.

[NIS18] NIST. Submission requirements and evaluation criteria for the lightweight cryptography standardization process , 2018. URL: https://csrc.nist.gov/CSRC/media/Projects/Lightweight-Cryptography/documents/final-lwc-submission-requirements-august2018.pdf.

[NSS20] Yusuke Naito, Yu Sasaki, and Takeshi Sugawara. Lightweight Authenticated Encryption Mode Suitable for Threshold Implementation. In Anne Canteaut and Yuval Ishai, editors, *Advances in Cryptology - EUROCRYPT 2020*, volume 12106 of *Lecture Notes in Computer Science*, pages 705–735. Springer, 2020. doi:10.1007/978-3-030-45724-2_24.

[NXP23] NXP Semiconductors. AN12527 - LPC55Sxx PRINCE Real-Time Data Encryption. Application note, 2023. URL: https://www.nxp.com/docs/en/application-note/AN12527.pdf.

[Nyb94] Kaisa Nyberg. Linear Approximation of Block Ciphers. In Alfredo De Santis, editor, *Advances in Cryptology - EUROCRYPT '94*, volume 950 of *LNCS*, pages 439–444. Springer, 1994. doi:10.1007/BFB0053460.

[Pey10] Thomas Peyrin. Improved Differential Attacks for ECHO and Grøstl. In Tal Rabin, editor, *Advances in Cryptology - CRYPTO 2010*, volume 6223 of *LNCS*, pages 370–392. Springer, 2010. doi:10.1007/978-3-642-14623-7_20.

[QDW+21] Lingyue Qin, Xiaoyang Dong, Xiaoyun Wang, Keting Jia, and Yunwen Liu. 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. doi:10.46586/TOSC.V2021.I2.249-291.

[Qua17] Qualcomm Technologies, Inc. Pointer Authentication on ARMv8.3. Technical report, Qualcomm Technologies, Inc., 2017. URL: https://www.qualcomm.com/content/dam/qcomm-martech/dm-assets/documents/pointer-auth-v7.pdf.

[SE20] Mohamed Shalan and Tim Edwards. Building OpenLANE: A 130nm OpenROAD-based Tapeout- Proven Flow : Invited Paper. In *2020 IEEE/ACM International Conference On Computer Aided Design (ICCAD)*, pages 1–6, 2020. doi:10.1145/3400302.3415735.

[Sel08] Ali Aydin Selçuk. On Probability of Success in Linear and Differential Cryptanalysis. *J. Cryptol.*, 21(1):131–147, 2008. doi:10.1007/S00145-007-9013-7.

[SLN+22] Kosei Sakamoto, Fukang Liu, Yuto Nakano, Shinsaku Kiyomoto, and Takanori Isobe. Rocca: An Efficient AES-based Encryption Scheme for Beyond 5G (Full version). *IACR Cryptol. ePrint Arch.*, page 116, 2022. URL: https://eprint.iacr.org/2022/116.

[SLR+15] Bing Sun, Zhiqiang Liu, Vincent Rijmen, Ruilin Li, Lei Cheng, Qingju Wang, Hoda Alkhzaimi, and Chao Li. Links among Impossible Differential, Integral and Zero Correlation Linear Cryptanalysis. *IACR Cryptol. ePrint Arch.*, page 181, 2015. URL: http://eprint.iacr.org/2015/181.

[SP04] Taizo Shirai and Bart Preneel. On Feistel Ciphers Using Optimal Diffusion Mappings Across Multiple Rounds. In Pil Joong Lee, editor, *Advances in Cryptology - ASIACRYPT 2004, 10th International Conference on the Theory and Application of Cryptology and Information Security, Jeju Island, Korea, December 5-9, 2004, Proceedings*, volume 3329 of *Lecture Notes in Computer Science*, pages 1–15. Springer, 2004. doi:10.1007/978-3-540-30539-2_1.

[SS04] Taizo Shirai and Kyoji Shibutani. Improving Immunity of Feistel Ciphers against Differential Cryptanalysis by Using Multiple MDS Matrices. In Bimal K. Roy and Willi Meier, editors, *Fast Software Encryption, 11th International Workshop, FSE 2004, Delhi, India, February 5-7, 2004, Revised Papers*, volume 3017 of *Lecture Notes in Computer Science*, pages 260–278. Springer, 2004. doi:10.1007/978-3-540-25937-4_17.

[SSD+18] Danping Shi, Siwei Sun, Patrick Derbez, Yosuke Todo, Bing Sun, and Lei Hu. Programming the Demirci-Selçuk Meet-in-the-Middle Attack with Constraints. In Thomas Peyrin and Steven D. Galbraith, editors, *Advances in Cryptology - ASIACRYPT 2018*, volume 11273 of *LNCS*, pages 3–34. Springer, 2018. doi:10.1007/978-3-030-03329-3_1.

[SSS+23] Danping Shi, Siwei Sun, Ling Song, Lei Hu, and Qianqian Yang. Exploiting Non-full Key Additions: Full-Fledged Automatic Demirci-Selçuk Meet-in-the-Middle Cryptanalysis of SKINNY. In Carmit Hazay and Martijn Stam,

editors, *Advances in Cryptology - EUROCRYPT 2023*, volume 14007 of *LNCS*, pages 67–97. Springer, 2023. doi:10.1007/978-3-031-30634-1_3.

[ST17] Yu Sasaki and Yosuke Todo. New Impossible Differential Search Tool from Design and Cryptanalysis Aspects - Revealing Structural Properties of Several Ciphers. In Jean-Sébastien Coron and Jesper Buus Nielsen, editors, *Advances in Cryptology - EUROCRYPT 2017*, volume 10212 of *LNCS*, pages 185–215, 2017. doi:10.1007/978-3-319-56617-7_7.

[SWW21] Ling Sun, Wei Wang, and Meiqin Wang. Accelerating the Search of Differential and Linear Characteristics with the SAT Method. *IACR Trans. Symmetric Cryptol.*, 2021(1):269–315, 2021. doi:10.46586/TOSC.V2021.I1.269-315.

[SYC+24] Ling Song, Qianqian Yang, Yincen Chen, Lei Hu, and Jian Weng. Probabilistic Extensions: A One-Step Framework for Finding Rectangle Attacks and Beyond. In Marc Joye and Gregor Leander, editors, *Advances in Cryptology - EUROCRYPT 2024*, volume 14651 of *LNCS*, pages 339–367. Springer, 2024. doi:10.1007/978-3-031-58716-0_12.

[TLS16] Yosuke Todo, Gregor Leander, and Yu Sasaki. Nonlinear Invariant Attack - Practical Attack on Full SCREAM, iSCREAM, and Midori64. In Jung Hee Cheon and Tsuyoshi Takagi, editors, *Advances in Cryptology - ASIACRYPT 2016*, volume 10032 of *LNCS*, pages 3–33, 2016. doi:10.1007/978-3-662-53890-6_1.

[TM16] Yosuke Todo and Masakatu Morii. Bit-Based Division Property and Application to Simon Family. In Thomas Peyrin, editor, *Fast Software Encryption - FSE 2016*, volume 9783 of *LNCS*, pages 357–377. Springer, 2016. doi:10.1007/978-3-662-52993-5_18.

[TMC+23] Meltem Sönmez Turan, Kerry McKay, Donghoon Chang, Lawrence Bassham, Jinkeon Kang, Noah Waller, John Kelsey, and Deukjo Hong. Status Report on the Final Round of the NIST Lightweight Cryptography Standardization Process. NIST IR 8454, 2023. URL: https://doi.org/10.6028/NIST.IR.8454.

[Tod15] Yosuke Todo. Structural Evaluation by Generalized Integral Property. In Elisabeth Oswald and Marc Fischlin, editors, *Advances in Cryptology - EUROCRYPT 2015*, volume 9056 of *LNCS*, pages 287–314. Springer, 2015. doi:10.1007/978-3-662-46800-5_12.

[Tri10] Will Tribbey. 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. doi:10.1145/1874391.187410.

[Wag99] David A. Wagner. The Boomerang Attack. In Lars R. Knudsen, editor, *Fast Software Encryption - FSE '99*, volume 1636 of *LNCS*, pages 156–170. Springer, 1999. doi:10.1007/3-540-48519-8_12.

[WBD+24] Jinliang Wang, Christina Boura, Patrick Derbez, Kai Hu, Muzhou Li, and Meiqin Wang. Cryptanalysis of Full-Round BipBip. *IACR Trans. Symmetric Cryptol.*, 2024(2):68–84, 2024. doi:10.46586/TOSC.V2024.I2.68-84.

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

[WNL+23] Jinliang Wang, Chao Niu, Qun Liu, Muzhou Li, Bart Preneel, and Meiqin Wang. Cryptanalysis of SPEEDY. In Leonie Simpson and Mir Ali Rezazadeh Baee, editors, *Information Security and Privacy - 28th Australasian Conference, ACISP 2023, Brisbane, QLD, Australia, July 5-7, 2023, Proceedings*, volume 13915 of *Lecture Notes in Computer Science*, pages 124–156. Springer, 2023. doi:10.1007/978-3-031-35486-1_7.

[Wol] Claire Wolf. Yosys Open SYnthesis Suite. https://yosyshq.net/yosys/.

[WWS23] Dachao Wang, Baocang Wang, and Siwei Sun. SAT-aided Automatic Search of Boomerang Distinguishers for ARX Ciphers. *IACR Trans. Symmetric Cryptol.*, 2023(1):152–191, 2023. doi:10.46586/TOSC.V2023.I1.152-191.

[XZBL16] Zejun Xiang, Wentao Zhang, Zhenzhen Bao, and Dongdai Lin. Applying MILP Method to Searching Integral Distinguishers Based on Division Property for 6 Lightweight Block Ciphers. In Jung Hee Cheon and Tsuyoshi Takagi, editors, *Advances in Cryptology - ASIACRYPT 2016*, volume 10031 of *LNCS*, pages 648–678, 2016. doi:10.1007/978-3-662-53887-6_24.

[YSZ+24] Qianqian Yang, Ling Song, Nana Zhang, Danping Shi, Libo Wang, Jiahao Zhao, Lei Hu, and Jian Weng. Optimizing Rectangle and Boomerang Attacks: A Unified and Generic Framework for Key Recovery. *J. Cryptol.*, 37(2):19, 2024. doi:10.1007/S00145-024-09499-1.

[ZHL23] Zhongyi Zhang, Chengan Hou, and Meicheng Liu. Collision Attacks on Round-Reduced SHA-3 Using Conditional Internal Differentials. In Carmit Hazay and Martijn Stam, editors, *Advances in Cryptology - EUROCRYPT 2023*, volume 14007 of *LNCS*, pages 220–251. Springer, 2023. doi:10.1007/978-3-031-30634-1_8.

[ZHL24] Zhongyi Zhang, Chengan Hou, and Meicheng Liu. Probabilistic Linearization: Internal Differential Collisions in up to 6 Rounds of SHA-3. In Leonid Reyzin and Douglas Stebila, editors, *Advances in Cryptology - CRYPTO 2024*, volume 14923 of *LNCS*, pages 241–272. Springer, 2024. doi:10.1007/978-3-031-68385-5_8.

[ZWZ+22] Lei Zhang, Ruichen Wu, Yuhan Zhang, Yafei Zheng, and Wenling Wu. LLL-WBC: A New Low-Latency Light-Weight Block Cipher. In Yi Deng and Moti Yung, editors, *Information Security and Cryptology - Inscrypt 2022*, volume 13837 of *LNCS*, pages 23–42. Springer, 2022. doi:10.1007/978-3-031-26553-2_2.

A Definition of distance between two ciphers

To define what is the *diversity* of a cipher c to a set of ciphers $\mathcal{C}$, we would first have to define the concept of *distance* between two ciphers. The next few definitions would hopefully clarify them.

Definition 10 (Normalized distance between two substitution layers)**.** Given two substitution layers, SL and SL', with n parallel sboxes. The normalized distance between the two layers is defined as the number of differing sboxes. In other words,

$$Dist(SL, SL') := \frac{\#\{SL_i \neq SL'_i \; \forall \; i \in [0..n-1]\}}{n}$$

Definition 11 (Normalized distance between two linear layers)**.** Given two linear layers, M and M', represented by $m \times m$ binary matrix. The normalized distance between the two matrices is the number of differing elements when comparing element-wise. In other words,

$$Dist(PL, PL') := \frac{\sum_{i,j} M_{i,j} \oplus M'_{i,j}}{m^2}$$

Definition 12 (Distance between two ciphers)**.** Given two ciphers, c and c', the distance between the ciphers is given by the sum of the normalized distance of their components

Definition 13 (Diversity)**.** Given a cipher c and a set of ciphers $\mathcal{C}$, the diversity is defined as the sum of the distances from c to every element in $\mathcal{C}$. i.e.

$$\text{Diversity}(c, \mathcal{C}) := \sum_{c' \in \mathcal{C}} \text{Dist}(c, c')$$

B uKNIT-BC components

B.1 uKNIT-BC substitution layers and their inverses

The 4-bit S-boxes applied to each nibble of the internal state in uKNIT-BC can be represented by $(D \circ S_{\texttt{MANTIS}} \circ B)(x)$ where $S_{\texttt{MANTIS}}$ is the MANTIS S-box:

$$S_{\texttt{MANTIS}}[x] = [\texttt{c}, \texttt{a}, \texttt{d}, 3, \texttt{e}, \texttt{b}, \texttt{f}, 7, 8, 9, 1, 5, 0, 2, 4, 6]$$

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 6.

The inverses of these S-boxes are represented by $(B^{-1} \circ S_{\texttt{MANTIS}} \circ D^{-1})(x)$. B^{-1} and D^{-1} are listed in Table 7.

Table 6: 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) (0132)	(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) (0312)	(03)(1)(2) (0)(1)(23)
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) (0321)	(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) (0321)	(02)(13) (0213)	(03)(12) (02)(13)	(0)(1)(23) (0)(132)	(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)

Table 7: The matrices D^{-1} (top) and B^{-1} (bottom) for all the inverses of S-boxes in each round. Each inverse of S-box is $B^{-1} \circ S_{\texttt{MANTIS}} \circ D^{-1}$.

S-box	Round											
	0	1	2	3	4	5	6	7	8	9	10	11
0	(120)(3) (3120)	(0)(231) (20)(1)(3)	(1320) (0)(231)	(1230) (0)(321)	(1320) (0)(21)(3)	(120)(3) (2310)	(210)(3) (0)(1)(32)	(30)(1)(2) (20)(1)(3)	(120)(3) (2130)	(3210) (0)(31)(2)	(30)(1)(2) (1230)	(20)(31) (210)(3)
1	(2130) (210)(3)	(230)(1) (2130)	(2310) (1230)	(30)(21) (0)(1)(32)	(0)(1)(2)(3) (2310)	(10)(32) (320)(1)	(30)(21) (0)(231)	(0)(321) (310)(2)	(1320) (3210)	(1230) (30)(1)(2)	(210)(3) (320)(1)	(2310) (320)(1)
2	(320)(1) (310)(2)	(10)(32) (120)(3)	(30)(21) (1320)	(120)(3) (0)(321)	(230)(1) (20)(1)(3)	(0)(321) (10)(32)	(2310) (30)(21)	(3210) (0)(321)	(210)(3) (0)(231)	(20)(1)(3) (0)(1)(2)(3)	(0)(21)(3) (2130)	(30)(1)(2) (0)(1)(32)
3	(3210) (0)(231)	(0)(231) (3120)	(0)(31)(2) (320)(1)	(320)(1) (0)(1)(2)(3)	(30)(1)(2) (0)(21)(3)	(0)(1)(32) (310)(2)	(10)(2)(3) (1230)	(20)(1)(3) (3210)	(30)(1)(2) (3120)	(120)(3) (0)(21)(3)	(30)(21) (3210)	(2310) (30)(1)(2)
4	(230)(1) (0)(1)(32)	(2130) (3120)	(0)(321) (0)(321)	(0)(1)(2)(3) (20)(1)(3)	(0)(31)(2) (20)(1)(3)	(20)(31) (0)(1)(32)	(30)(1)(2) (2310)	(2130) (30)(1)(2)	(130)(2) (0)(1)(32)	(130)(2) (3120)	(0)(231) (120)(3)	(0)(231) (3120)
5	(230)(1) (1320)	(1230) (310)(2)	(120)(3) (30)(21)	(20)(31) (130)(2)	(230)(1) (10)(2)(3)	(130)(2) (30)(21)	(1320) (30)(1)(2)	(20)(31) (0)(321)	(320)(1) (30)(1)(2)	(3210) (10)(2)(3)	(2310) (0)(321)	(0)(321) (130)(2)
6	(2310) (210)(3)	(30)(21) (1230)	(1230) (20)(1)(3)	(10)(32) (310)(2)	(3120) (0)(1)(2)(3)	(120)(3) (230)(1)	(20)(1)(3) (0)(321)	(30)(21) (0)(21)(3)	(210)(3) (130)(2)	(0)(321) (230)(1)	(230)(1) (210)(3)	(0)(31)(2) (20)(1)(3)
7	(230)(1) (0)(321)	(2130) (0)(31)(2)	(20)(1)(3) (0)(1)(32)	(10)(32) (310)(2)	(0)(321) (320)(1)	(0)(21)(3) (0)(321)	(20)(31) (310)(2)	(20)(31) (3210)	(0)(321) (2130)	(130)(2) (2310)	(20)(31) (310)(2)	(0)(21)(3) (0)(31)(2)
8	(2130) (2130)	(20)(1)(3) (20)(31)	(10)(2)(3) (30)(21)	(130)(2) (30)(21)	(0)(231) (0)(21)(3)	(310)(2) (3210)	(1230) (0)(1)(32)	(20)(31) (20)(1)(3)	(320)(1) (210)(3)	(210)(3) (0)(21)(3)	(0)(1)(2)(3) (1230)	(1320) (20)(31)
9	(0)(1)(32) (230)(1)	(3120) (30)(1)(2)	(1230) (0)(31)(2)	(130)(2) (310)(2)	(320)(1) (1320)	(20)(1)(3) (1230)	(10)(32) (10)(2)(3)	(10)(32) (320)(1)	(310)(2) (230)(1)	(20)(1)(3) (130)(2)	(2130) (130)(2)	(30)(21) (3210)
10	(0)(1)(2)(3) (10)(32)	(30)(21) (0)(231)	(2130) (1320)	(210)(3) (30)(21)	(20)(31) (0)(321)	(3120) (0)(1)(32)	(230)(1) (310)(2)	(120)(3) (30)(21)	(1320) (30)(1)(2)	(1320) (0)(21)(3)	(20)(1)(3) (2130)	(210)(3) (0)(1)(32)
11	(1230) (0)(31)(2)	(1320) (230)(1)	(2130) (0)(321)	(210)(3) (0)(231)	(2310) (230)(1)	(30)(21) (1320)	(0)(1)(2)(3) (30)(21)	(3210) (30)(21)	(0)(21)(3) (30)(21)	(0)(31)(2) (320)(1)	(0)(31)(2) (1230)	(310)(2) (2310)
12	(3210) (310)(2)	(20)(31) (0)(231)	(210)(3) (1230)	(20)(31) (3120)	(30)(21) (20)(31)	(0)(1)(32) (0)(231)	(30)(1)(2) (3210)	(10)(2)(3) (20)(1)(3)	(0)(231) (120)(3)	(210)(3) (3210)	(130)(2) (10)(2)(3)	(20)(31) (0)(21)(3)
13	(0)(1)(32) (0)(21)(3)	(130)(2) (0)(21)(3)	(230)(1) (20)(1)(3)	(0)(21)(3) (2130)	(3210) (210)(3)	(0)(231) (230)(1)	(230)(1) (30)(1)(2)	(210)(3) (0)(231)	(2130) (230)(1)	(30)(21) (1320)	(30)(1)(2) (0)(321)	(2130) (20)(31)
14	(3210) (0)(231)	(3210) (0)(321)	(2310) (10)(2)(3)	(1320) (0)(31)(2)	(0)(21)(3) (130)(2)	(1320) (320)(1)	(20)(1)(3) (0)(1)(32)	(3210) (2130)	(0)(1)(32) (0)(31)(2)	(1230) (1230)	(0)(21)(3) (0)(231)	(0)(31)(2) (1320)
15	(20)(31) (0)(1)(32)	(0)(31)(2) (130)(2)	(2310) (2310)	(0)(1)(32) (230)(1)	(2130) (0)(1)(32)	(20)(1)(3) (30)(1)(2)	(1320) (230)(1)	(1230) (2130)	(0)(321) (3210)	(0)(231) (20)(31)	(210)(3) (10)(2)(3)	(0)(231) (10)(32)

B.2 uKNIT-BC linear layers and their inverses

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 9 and 10 of Appendix B.2 provide the list of these indices.

The inverses of these linear layers are also sparse, we list them as the same style in Tables 11 and 12.

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 8.

Table 8: 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 9: 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 10: 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

Table 11: The inverses of linear layers 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	41	55	1	28	53	0	28	43	0	29	40	2	31	41	1	42	52	2	30	53	1	43	52	30	43	52	29	47	54	3	42	55
1	1	43	53	2	30	54	31	40	54	2	28	41	3	29	42	3	43	53	3	29	55	3	29	53	31	41	54	30	40	52	0	43	53
2	2	40	54	3	31	52	3	42	53	1	31	43	1	28	40	2	40	55	1	31	54	31	41	54	28	40	55	31	42	53	1	40	52
3	3	42	52	0	29	55	2	30	52	3	30	44	0	30	43	0	41	54	0	28	52	2	30	40	29	42	53	28	43	55	2	41	54
4	29	43	53	3	31	42	3	29	53	1	31	54	3	29	52	3	30	53	2	30	42	3	29	42	1	41	54	0	31	42	28	41	54
5	31	40	54	1	28	41	2	30	41	3	30	52	2	31	54	0	28	54	1	31	41	30	40	55	0	40	55	2	30	40	30	40	52
6	30	41	55	0	29	40	28	43	55	0	29	55	0	30	53	2	29	55	0	4	28	1	28	52	3	42	53	1	28	43	29	42	55
7	28	42	52	2	30	43	1	31	54	2	28	53	1	28	55	1	31	52	0	4	52	0	41	54	2	43	52	3	29	47	31	43	53
8	1	29	43	28	41	53	0	43	55	20	21	60	0	43	53	1	31	42	29	43	55	29	42	53	0	28	55	1	28	55	0	31	43
9	2	31	40	29	40	55	3	29	42	0	40	55	3	42	52	2	29	40	31	41	54	1	28	43	2	30	52	2	30	52	3	29	42
10	0	30	41	31	42	52	30	41	52	1	43	54	1	40	55	3	30	43	4	28	52	2	40	55	3	29	53	3	29	54	1	30	40
11	3	28	42	30	43	54	1	40	54	3	44	52	2	41	54	0	28	41	30	42	53	0	31	54	1	31	54	0	31	53	2	28	41
12	2	31	54	3	42	52	4	16	45	28	41	53	30	43	53	28	41	54	3	29	43	2	30	55	1	31	41	15	39	49	1	30	52
13	3	28	52	2	43	54	29	42	53	29	40	55	29	42	52	30	43	53	1	41	54	28	43	52	2	30	43	2	40	52	0	31	53
14	0	30	55	0	40	55	0	28	55	31	43	54	28	40	55	29	40	55	3	43	55	3	42	53	0	28	40	0	42	53	3	29	55
15	1	29	53	1	41	53	1	31	40	30	44	52	31	41	54	31	42	52	2	42	53	0	31	41	3	29	42	3	47	54	2	28	54
16	5	47	59	4	19	59	7	47	56	4	42	58	6	16	57	7	16	46	11	47	58	4	45	59	6	46	58	7	18	59	5	18	46
17	7	44	56	6	44	58	6	46	58	7	46	57	19	44	56	6	45	56	5	44	56	6	44	56	4	16	44	5	19	58	4	45	57
18	4	46	58	7	17	45	4	45	57	6	45	56	5	45	58	4	17	59	6	46	57	5	47	58	5	17	59	4	16	57	7	16	59
19	6	45	57	16	47	56	5	44	59	5	47	59	4	18	47	18	44	57	40	45	59	7	46	57	18	47	56	6	17	56	19	44	58
20	7	19	56	6	18	58	6	18	58	6	17	56	7	44	56	6	19	56	11	17	58	4	18	59	16	44	57	7	45	59	7	16	47
21	6	17	57	17	45	57	4	16	57	4	19	58	5	17	58	5	44	57	16	40	59	6	19	56	5	17	45	18	45	59	18	46	56
22	5	18	59	5	47	56	7	19	56	7	18	57	6	16	46	16	46	58	5	19	56	7	17	57	7	47	56	6	41	56	6	44	58
23	4	16	58	4	19	46	5	17	59	5	16	59	18	47	59	4	17	47	6	18	57	5	16	58	6	19	58	4	44	57	4	17	57
24	6	17	45	19	46	59	5	17	44	4	19	42	5	17	45	6	19	45	5	19	44	4	18	45	6	19	46	5	46	58	5	46	56
25	4	16	46	6	18	44	6	18	46	5	16	47	7	19	56	7	46	58	11	17	47	6	19	44	4	44	57	16	44	57	16	47	59
26	5	18	47	7	45	57	7	19	47	6	17	45	16	46	57	5	18	57	16	40	45	7	17	46	7	18	56	19	46	58	4	17	45
27	7	19	44	5	16	56	2	41	52	7	18	46	4	47	59	17	47	59	6	18	46	5	16	47	17	45	59	17	41	56	6	19	58
28	17	45	57	18	44	58	16	45	57	18	46	57	4	18	59	19	45	56	19	44	56	19	44	56	19	46	58	7	18	45	17	45	57
29	18	47	59	5	16	47	18	46	58	17	45	56	6	46	57	5	18	44	18	46	57	16	47	58	5	45	59	6	17	41	5	18	56
30	16	46	58	4	46	59	17	44	59	16	47	59	7	19	44	4	47	59	17	47	58	17	46	57	7	18	47	5	19	46	6	19	44
31	19	44	56	7	17	57	19	47	56	19	42	58	17	45	58	7	16	58	16	45	59	18	45	59	4	16	57	4	16	44	7	47	59

Table 12: The inverses of linear layers 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	9	20	62	9	21	63	22	35	61	22	35	63	9	22	35	20	35	60	8	33	60	11	20	34	22	32	63	20	35	60	21	33	63
33	11	22	34	11	22	32	23	32	63	10	23	33	10	23	33	23	32	61	9	21	35	9	22	35	23	34	62	23	32	62	8	22	34
34	23	33	61	10	33	62	20	34	60	21	32	60	11	20	34	21	34	62	22	34	62	8	23	32	20	33	61	11	34	61	10	35	61
35	8	35	63	20	34	61	21	33	62	8	9	61	8	21	32	22	33	63	7	20	63	10	21	33	21	35	60	22	33	63	11	23	62
36	8	21	63	22	32	60	9	32	63	11	35	63	21	32	60	8	35	60	8	23	60	9	22	63	11	34	62	9	20	35	10	20	61
37	10	33	61	10	23	62	11	33	62	2	41	53	20	34	63	11	32	61	10	34	62	10	21	61	10	33	61	10	23	32	9	33	63
38	9	20	32	8	34	61	10	34	60	23	33	62	22	35	62	10	34	62	21	35	61	11	20	62	9	35	60	21	22	33	22	34	60
39	22	34	60	9	21	35	8	35	61	8	9	34	23	33	61	9	33	63	7	20	32	8	23	60	8	32	63	8	11	34	11	23	32
40	11	34	60	21	35	63	10	20	60	11	22	63	10	33	61	8	20	60	9	35	61	9	35	63	8	22	63	10	23	62	10	20	35
41	10	23	61	10	23	33	11	21	62	10	33	62	8	32	60	11	23	61	8	23	33	10	33	61	10	20	61	8	11	61	9	21	63
42	8	21	35	8	20	61	8	22	61	20	21	32	11	34	63	10	21	62	20	32	63	11	34	62	9	21	60	9	20	60	23	32	62
43	20	32	62	11	32	60	9	23	32	8	34	61	9	35	62	9	22	63	10	22	62	8	32	60	11	23	62	21	22	63	8	34	60
44	11	22	60	9	35	63	9	23	63	10	23	62	8	21	60	10	21	34	9	21	61	23	32	60	11	23	34	9	35	60	20	35	61
45	10	23	33	11	22	60	10	20	34	20	32	60	10	23	61	9	22	33	10	22	34	22	35	63	9	21	35	10	32	62	9	21	33
46	21	35	63	8	20	34	8	22	35	9	34	61	11	20	63	11	23	32	23	33	60	20	34	62	8	22	32	8	34	61	11	32	62
47	9	32	62	23	33	62	11	21	33	11	22	35	9	22	62	8	20	35	7	32	63	21	33	61	10	20	33	21	33	63	8	22	60
48	26	38	48	13	39	49	12	36	39	26	37	50	14	24	37	14	26	36	27	38	48	24	36	49	15	36	48	13	36	51	14	26	48
49	13	24	39	14	24	36	13	25	49	14	25	49	15	38	50	15	24	49	25	37	50	25	39	51	12	39	49	15	25	49	27	37	50
50	14	36	50	15	25	48	14	38	50	12	27	51	25	39	51	27	38	50	24	36	49	26	37	48	13	37	50	26	38	48	13	25	36
51	15	27	49	27	37	51	24	27	51	15	26	50	12	27	49	13	37	51	26	39	51	27	38	50	14	38	51	14	27	37	15	38	51
52	14	25	50	12	37	51	13	25	37	13	24	39	12	27	36	13	25	51	15	25	37	13	27	38	25	39	49	27	37	50	25	36	49
53	15	27	37	15	25	38	36	39	48	14	25	36	15	26	50	12	38	50	12	27	38	12	25	39	24	36	48	13	24	51	12	37	50
54	24	39	51	24	36	50	14	26	50	12	27	38	13	39	51	26	36	48	13	24	36	14	24	36	27	38	51	15	25	39	14	26	39
55	12	38	48	13	26	49	15	24	51	15	26	37	24	37	48	15	24	39	14	26	39	15	26	37	26	37	50	12	38	48	15	24	51
56	13	39	51	13	26	39	12	36	48	24	39	48	14	37	48	13	25	37	12	27	48	15	26	48	12	25	39	25	39	49	15	24	38
57	14	25	36	12	27	51	25	37	49	27	38	51	27	36	49	12	27	50	14	26	51	14	24	49	13	26	37	12	26	48	26	39	48
58	12	26	48	25	38	48	14	26	38	13	24	48	13	25	51	24	39	49	13	24	49	13	27	50	15	24	36	13	24	36	13	36	49
59	27	37	49	14	36	50	15	27	51	25	36	49	15	26	38	14	36	48	15	25	50	12	25	51	14	27	38	14	37	50	12	27	50
60	25	36	50	15	38	48	12	39	48	13	39	48	13	25	39	14	26	48	14	39	51	13	38	50	13	26	50	1	43	55	24	38	51
61	13	24	51	26	39	49	13	37	49	15	37	50	12	36	49	12	27	38	13	36	49	12	39	51	14	27	51	24	36	51	12	27	37
62	12	26	38	12	27	37	26	38	50	12	38	51	26	38	50	25	37	51	15	37	50	15	37	48	12	25	49	12	26	38	13	25	49
63	15	37	49	14	24	50	15	24	27	14	36	49	14	24	48	15	39	49	12	38	48	14	36	49	15	24	48	14	27	50	14	39	48

C An alternative representation of uKNIT-BC

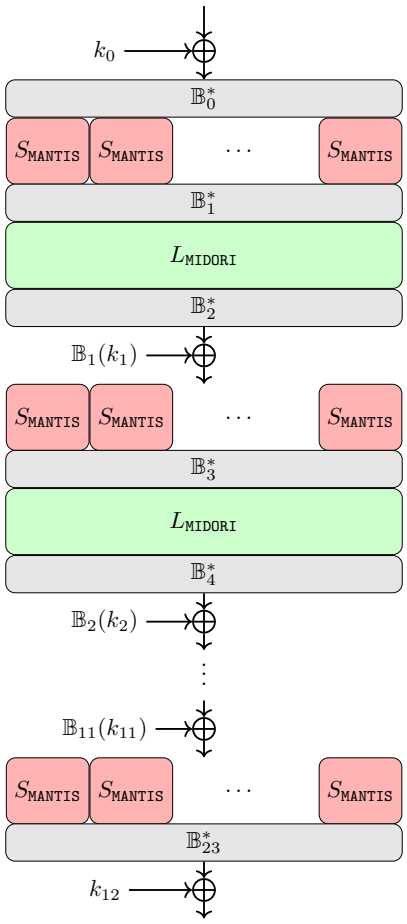
In this section, we provide an alternative representation of uKNIT-BC. Since the linear layers in uKNIT-BC are also "bit-permutation equivalence" to that of in MIDORI, we can represent our cipher using MANTIS's substitution and MIDORI's linear layers with 24 additional bit permutation layers (see Figure below). Let

$$\mathbb{S}_{\texttt{uKNIT-BC}} = S_{\texttt{uKNIT-BC}} \,||\, S_{\texttt{uKNIT-BC}} \,||\, \ldots \,||\, S_{\texttt{uKNIT-BC}}$$

Then uKNIT-BC can be represented by

$$\begin{aligned}
&\mathbb{S}_{\texttt{uKNIT-BC}} \cdot L_{\texttt{uKNIT-BC}} \cdot \mathbb{S}_{\texttt{uKNIT-BC}} \cdot \ldots \cdot \mathbb{S}_{\texttt{uKNIT-BC}} \\
&= (\mathbb{B}_0 \cdot \mathbb{S}_{\texttt{MANTIS}} \cdot \mathbb{D}_0)(\mathbb{A}_0 \cdot L_{\texttt{MIDORI}} \cdot \mathbb{C}_0)(\mathbb{B}_1 \cdot \mathbb{S}_{\texttt{MANTIS}} \cdot \mathbb{D}_1) \cdot \ldots \cdot (\mathbb{B}_{11} \cdot \mathbb{S}_{\texttt{MANTIS}} \cdot \mathbb{D}_{11}) \\
&= \mathbb{B}_0^* \cdot \mathbb{S}_{\texttt{MANTIS}} \cdot \mathbb{B}_1^* \cdot L_{\texttt{MIDORI}} \cdot \mathbb{B}_2^* \cdot \mathbb{S}_{\texttt{MANTIS}} \cdot \mathbb{B}_3^* \cdot \ldots \cdot \mathbb{S}_{\texttt{MANTIS}} \cdot \mathbb{B}_{23}^*
\end{aligned}$$

where $\mathbb{B}_i$ for $i \in \{0..11\}$ and $\mathbb{B}_j^*$ for all $j \in \{0, ..., 23\}$ are 64×64 transposition matrices. The matrices $\mathbb{B}_i$ and $\mathbb{B}_j^*$ can be found in our GitHub.



D 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

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

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

E 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 active

   We 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 active

   This 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,

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

   where $\mathcal{A}(M_b)$ and $\mathcal{A}(M_f)$ are the number of active S-boxes in the two-end key-guess processes, $\mathrm{P}(M_d)$ 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.

F 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 4, uKNIT can provide $\sim 10\%$ lower latency that PRINCEv2, but, interestingly, for the same latency as PRINCEv2 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 [HWKS98] and its security was proven in [DHT17], 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 $\approx 55128.24\mu m^2$.

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.

G Rotations for the SPECK-32 and SPECK-128 variants and a different byte-permutation for AES variant

Table 13: Rotation values for the SPECK-32 and SPECK-128 variants

round		1	2	3	4	5	6	7	8
SPECK-32	α	10	10	3	9	14	5	5	13
	β	13	2	1	12	8	2	15	15
SPECK-128	α	23	21	18	14	10	8	–	–
	β	4	20	16	16	10	20	–	–

Table 14: Byte-permutations for the AES variant searched by our genetic search algorithm. Note that the convention we use to represent the position follows that of the one in [DR02] (i.e. column-first).

Rd \ Pos.	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
1	13	5	8	11	6	2	14	15	9	4	10	1	7	3	0	12
2	6	9	15	2	8	7	1	12	11	4	14	0	13	3	5	10
3	8	5	0	15	4	9	3	13	7	10	12	1	11	14	6	2
4	11	0	4	8	6	14	13	9	12	10	3	1	2	7	15	5
5	7	3	6	0	13	12	2	10	15	1	11	9	14	4	5	8
6	15	11	4	6	2	1	10	7	13	14	8	3	0	9	12	5

H Power consumption and area at different frequencies

Table 15: 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