

# GPU-Accelerated Branch-and-Bound Algorithm for Differential Cluster Search of Block Ciphers

Wei-Zhu Yeoh<sup>1</sup>, Je Sen Teh<sup>1</sup> and Jiageng Chen<sup>2</sup>

<sup>1</sup> Universiti Sains Malaysia, Malaysia

[yeohweizhu@gmail.com](mailto:yeohweizhu@gmail.com) , [jesen\\_teh@usm.my](mailto:jesen_teh@usm.my)

<sup>2</sup> Central China Normal University, China

[chinkako@gmail.com](mailto:chinkako@gmail.com)

**Abstract.** Differential cryptanalysis of block ciphers requires the identification of differential characteristics with high probability. For block ciphers that have a large block size and a large number of rounds, identifying these differential trails is a computationally intensive task. Matsui first proposed a branch-and-bound algorithm to search for differential trails in 1994. There have been numerous improvements made to the branch-and-bound algorithm since then, such as improving its efficiency by bounding the number of active s-boxes, incorporating a meet-in-the-middle approach, and adapting it to different block cipher architectures like ARX. Although mixed-integer linear programming (MILP) technique has been widely used recently to evaluate the differential resistance of block ciphers, MILP is still an inefficient technique for clustering differential trails (also known as the differential effect). The branch-and-bound method is still a tool better suited for the task of trail clustering. However, it still requires enhancements before being feasible for block ciphers with large block sizes, especially for a large number of rounds. Motivated by the need for a more efficient branch-and-bound algorithm to search for block cipher differential clusters, we propose a GPU-accelerated branch-and-bound approach. The proposed GPU-accelerated algorithm substantially increases the performance of the differential cluster search. We were able to derive a branch enumeration and evaluation kernel that is 5.95 times faster than its CPU counterpart. Then to showcase the practicality of the proposed approach, it is applied on TRIFLE-BC, a 128-bit block cipher. By utilizing the proposed GPU kernel together the incorporation of a meet-in-the-middle approach, we were able to improve the performance of the algorithm by approximately 60 times the original recursive algorithm based on a 20-round TRIFLE-BC. Also, differential clusters with sizes of approximately 2 million for 43 rounds were constructed, leading to slight improvements to the overall differential probabilities. This result depicts the practicality of the proposed GPU framework in constructing clusters consisting of millions of differential trails, which could be used to improve cryptanalytic findings against other block ciphers in the future.

**Keywords:** Cryptanalysis · automatic search · differential characteristic · differential cluster · block cipher · GPU · branch-and-bound

## 1 Introduction

Differential cryptanalysis [BS91] is one of the most wide-known cryptanalytical methods, the resistance to which has become one of the basic requirements for modern block ciphers [BKL<sup>+</sup>07][GPPR11][BPP<sup>+</sup>17]. The success of differential cryptanalysis relies on identifying high probability differentials. The search for high probability differentials of modern block ciphers is a non-trivial task especially for block ciphers with large block sizes and number of rounds.

Automatic search for differential characteristics could be used instead of performing manual searching. Matsui [Mat95] proposed a classic branch-and-bound technique to search for differential characteristics and linear trails. This technique was used at that time to study DES. Since then, there were numerous improvements that have been made to the branch-and-bound algorithm. In [BV14] an ARX version of the branch-and-bound searching algorithm was proposed and the algorithm was also subsequently improved in [CTX<sup>+</sup>16] by the introduction of a sorted partial differential distribution table. In addition, [CMST15] incorporated a meet-in-the-middle approach to the differential cluster search and updated the pruning rules to bound the number of active s-boxes to further improve upon the search efficiency.

In [MWGP12], a mixed-integer linear programming (MILP) approach was proposed as an alternative to the branch-and-bound algorithm. The MILP model requires identifying relevant linear inequalities which are then fed into a MILP solver which produces the minimal number of active s-boxes for a particular block cipher. The MILP framework had been extended by [SHS<sup>+</sup>14] to be applicable to bit-oriented block ciphers. [SLM<sup>+</sup>14] demonstrated the capability of MILP to enumerate differential characteristics to form differential clusters or linear hulls. However, the aforementioned method is impractical for identifying differential clusters for block ciphers with large block sizes and rounds. In addition, none of the related-works attempt to utilize specialized hardware acceleration to perform the search.

General purpose graphical processing unit (GPGPU) technology that utilizes specialized GPU hardware could be used to improve the efficiency of the branch-and-bound search. This would alleviate some of the computational load needed to identify differential clusters for large block ciphers. However, the GPU requires tasks to be divided into smaller tasks so that the subdivided tasks could be processed across a large number of processing units simultaneously. The GPU architecture also has its own array of optimization problems such as memory limitations, work divergence, low number of available subdivided tasks, and many more. Therefore, any GPU-accelerated searching algorithm needs to be optimized with respect to the architecture of the GPU to obtain a reasonable performance boost. In this paper, we propose a GPU framework to improve the efficiency of the branch-and-bound searching algorithm for differential clusters.

## 1.1 Our Contributions

In this paper, we proposed an enhanced automatic search for differential clusters accelerated by GPU hardware. The proposed algorithm also incorporated a meet-in-the-middle (MITM) approach [CMST15] to further improve the efficiency of the GPU-based search. Although there have been numerous GPU-accelerated branch-and-bound proposals [LEB12][BHG17][MCMT12], these proposals solved different sets of sub-problems unrelated to cryptanalysis. Therefore, this work is a novel approach leveraging GPU hardware acceleration for this specific sub-problem. The proposed algorithm achieved a substantial speedup up to a factor of approximately 5.95. The proposed technique can theoretically be extended to utilize a grid of CPU-GPU computing nodes in a real-world environment to enhance the efficiency even further.

To showcase the practicality and feasibility of the proposed GPU-accelerated algorithm, an investigation of the differential cluster effect is performed on the 128-bit block cipher, TRIFLE-BC [NAD<sup>+</sup>19] as a proof of concept. TRIFLE-BC is chosen as the target cipher because it is used as the underlying primitive of the lightweight authenticated cipher TRIFLE, which is one of the round-1 candidates of the ongoing lightweight cryptography standardization effort by NIST [NIS19a]. By applying the newly proposed GPU-accelerated automatic search for differential clusters, the computational time needed to compute the differential clusters for a large number of rounds of 128-bit TRIFLE-BC was significantly shortened. This effectively allowed us to find differentials with the highest probability to

date, by clustering differential characteristics. Thus, this work also contributes towards the NIST standardization efforts for lightweight cryptography in terms of cryptanalysis findings.

However, we wish to note that the cryptanalytic findings for TRIFLE-BC is not the main contribution of this work but rather to depict the practicality of the proposed approach in discovering large clusters for full-sized (non-lightweight) block ciphers with a large number of rounds. This work is one of the first successful attempts in implementing an automated search for differentials for a block cipher with 128-bit block size at a very large number of rounds (43 rounds). This is significant because the computational requirements of a branch-and-bound differential search is proportionate to the block size and also increases exponentially as the number of rounds grows. Previous automated search for differential clusters have focused on block cipher with sizes of 64 bits and below [CMST15][SLM<sup>+</sup>14]. For literature that involve identifying differentials for 128-bit block ciphers, the number of rounds searched are noticeably lower (typically  $\leq 20$ ) and are only capable of identifying a singular differential characteristics [BN10] [EAY19]. Although the framework proposed in [SHY16] was able to identify clusters for SPECK128 and LEA-128, it is not applicable to most ARX ciphers due to its reliance on the independent addition assumption. Also, it could be noted that all prior findings could be potentially improved by applying the proposed GPU framework.

## 1.2 Outline

The remainder of this paper starts off with a brief introduction to GPU architecture and CUDA technology, followed by a recap on TRIFLE's specification and prior differential cryptanalysis efforts. In Section 3, we provide descriptions for both the conventional automatic branch-and-bound search for block cipher differential clusters along with the improved version of the algorithm that serves as the basis for GPU-acceleration. In Section 4, the GPU-accelerated differential cluster algorithm will be detailed. Its performance will be benchmarked against its CPU-counterpart to gauge the efficiency and speed-up attained by the proposed algorithm. The capabilities and limitations of the proposed algorithm are also discussed. In Section 5, we investigate the differential cluster effect of TRIFLE-BC. Section 7 concludes the paper.

## 2 Preliminaries

In this section, background information on GPU architecture, CUDA and TRIFLE are provided to aid readers' understanding of the remaining sections of this paper.

### 2.1 GPU architecture and CUDA

A graphics processing unit (GPU) is specialized hardware designed for highly multithreaded and parallelized data processing workflow. The primary function of a GPU is to manipulate computer graphics and perform image processing. However, the massively parallel processing architecture of GPUs has also enabled them to outperform central processing units (CPUs) in other non-graphical processing algorithms that involve a massive amount of data. With the introduction of the Compute Unified Device Architecture (CUDA) in 2006 by NVIDIA, the parallel processing power of GPUs becomes readily available for solving many other computationally complex problems.

CUDA is a general-purpose parallel computing platform and application programming interface (API) designed by NVIDIA for NVIDIA GPU cards. GPUs are based on the single instruction, multiple threads (SIMT) execution model whereby multiple distinct threads perform the same operation on multiple data concurrently. By dedicating more transistors

to data processing (arithmetic logic unit, ALU) and consequently de-emphasizing data caching and flow control, parallel computation becomes more efficient. The aforementioned structure is schematically illustrated in Figure 1. This unique property of GPUs allows them to efficiently solve data-parallel computational problems that are arithmetic-heavy but with lower memory access frequency.

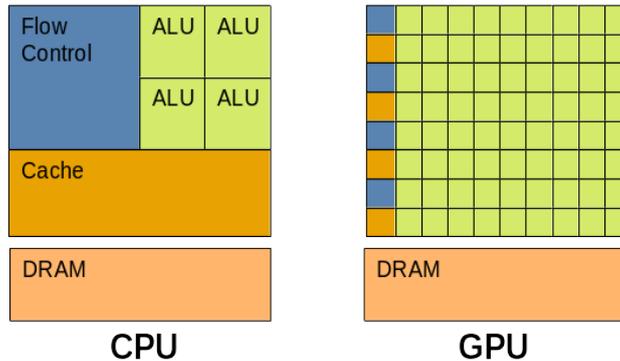


Figure 1: The illustration of structural differences between CPU and GPU.

As illustrated by Figure 2, CUDA threads run on a separate physical **device** (GPU) to accelerate parallel tasks given by the co-running **host** program (CPU). The host and device analogy will be used throughout the paper. A kernel is a CUDA device function that will be executed in parallel by different CUDA threads on the device. A single kernel consists of a single grid that may hold a maximum of  $2^{31} - 1$  number of blocks, whereas each block can contain a maximum of  $2^{10}$  threads.

When a kernel is launched, the blocks that reside within the kernel are assigned to idle streaming multiprocessors (SM). The multiprocessors execute parallel threads within the assigned block in groups of 32 called warps. A warp executes one common instruction at a time. If threads of a warp diverge due to conditional instruction, each branch path will be executed in different warp cycles. Therefore, the use of conditional branches should be minimized to maximize the multiprocessors' efficiency. Since an SM executes a warp of 32 threads at a time, it is advisable to choose the number of threads per block to be a multiple of 32 to optimize GPU utilization.

CUDA threads are able to read data from multiple types of memory during their execution. Each thread has its own local memory. Threads reside within the same thread block can access a shared memory space called the shared memory. There are three types of memory visible to all threads namely global memory, read-only constant memory, and read-only texture memory. Global memory is the slowest memory and requires read/write to be coalesced in 32, 64, or 128-byte memory to achieve maximum efficiency. Constant memory is optimized for broadcasting whereby the maximum efficiency is reached when all threads of the same warp request the same memory address. Texture memory is optimized for 2D spatial locality [Pad11], whereby threads of the same warp reading memory locations that are close to each other will lead to maximum efficiency. Since the different memory types are better suited for different tasks, the memory access pattern of a CUDA program should also be optimized accordingly to maximize efficiency.

The CUDA model maintains separate memory spaces for host and device memory. Thus, the program requires manual management of a device's memory space (allocation and deallocation) during the CUDA runtime. To alleviate the complexity of memory management, unified memory may be used to unify the host and device memory spaces. Unified managed memory provides a single coherent memory address visible to both CPU

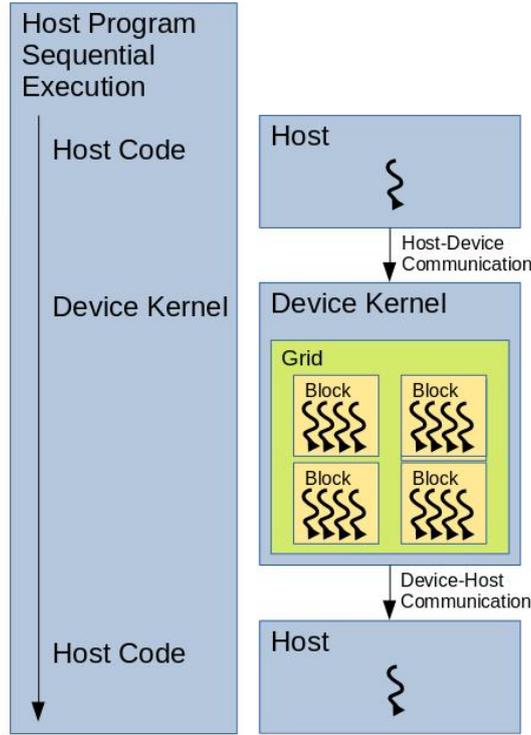


Figure 2: Heterogeneous programming architecture of a typical GPU-accelerated algorithm. (Note that serial host code executes on the CPU while parallel device code executes on the GPU)

and GPU. Pinned memory is used as a staging area by CUDA for the transfer of data between device and host memory. If a large amount of memory transfer is needed and the transfer happens often, it is advised to pin down the memory to avoid the cost of the transfer between page-able and pinned memory. Pinned memory also enables the asynchronous (non-blocking) execution of kernel and data transfer.

This section has only covered information that are relevant to the proposed work. There are a lot more features left unexplored such as concurrent kernel launches, asynchronous execution, and multi-device execution. For a more detailed guide and reference in optimizing for CUDA, refer to [NVI19].

## 2.2 TRIFLE

### 2.2.1 Notation

The following mathematical notations will be used throughout the paper:

- $\{0, 1\}^*$  denotes the set of all strings.
- $\{0, 1\}^n$  denotes the set of strings of length  $n$ .
- $|M|$  denotes the length (number of bits) in string  $M$ .
- $M_1||M_2$  denotes concatenation of string  $M_1$  and string  $M_2$ .

- $\oplus$  denotes field addition and  $\otimes$  field multiplication.
- $\text{OZP}(X)$  applies an optional  $10^*$  padding on  $n$  bits. If  $|X| < n$ , then  $\text{OZP}(X) = 0^{n-|X|-1}||1||X$ . If  $|X| = n$ , then  $\text{OZP}(X) = X$ .
- $\lfloor X \rfloor$  is an integer floor function that produces an integer  $i$  closest to  $X$  such that  $i \leq X$ .
- $\ggg$  denotes bitwise right rotations.
- $W_{bit}(X)$  denotes the number of 1 bits in a given binary string  $X$  while  $W_{nibble}(X)$  denotes the number of non-zero 4-bit values in a binary string  $X$ .
- $AS$  is used to represent the number of active s-boxes.
- $P_c$  represents the probability of a differential cluster and  $P_t$  is the probability of a single differential trail.
- $\Delta X$  is an XOR difference,  $\Delta U_i$  is the  $i^{\text{th}}$  nibble value inside  $\Delta X$ , and  $\Delta AU_i$  is the  $i^{\text{th}}$  active nibble value (non-zero difference) inside  $\Delta X$ .

### 2.2.2 Description of TRIFLE

TRIFLE is one of the round-1 candidates of lightweight authenticated encryption standardization effort organized by NIST [NIS19a]. TRIFLE is a block cipher-based authenticated encryption scheme with a block size of 128 bits. It receives an encryption key  $K \in \{0, 1\}^{128}$ , nonce  $N \in \{0, 1\}^{128}$ , associated data  $A \in \{0, 1\}^*$  and message  $M \in \{0, 1\}^*$  as inputs, and produces an encrypted ciphertext  $C \in \{0, 1\}^{|M|}$  and an authentication tag  $T \in \{0, 1\}^{128}$  as outputs. The corresponding verification and decryption scheme receives a key, nonce, associated data, ciphertext and a tag as inputs, and produces the decrypted plaintext if the authentication tag is valid.

TRIFLE employs a MAC-then-Encrypt scheme whereby a cipher block chaining-esque (CBC) authentication is performed on the nonce, associated data and plaintext to produce the authentication tag. The authentication tag is used as the initialization vector (IV) in output feedback (OFB) mode to produce the ciphertext. For a more detailed TRIFLE specification, refer to [NAD<sup>+</sup>19].

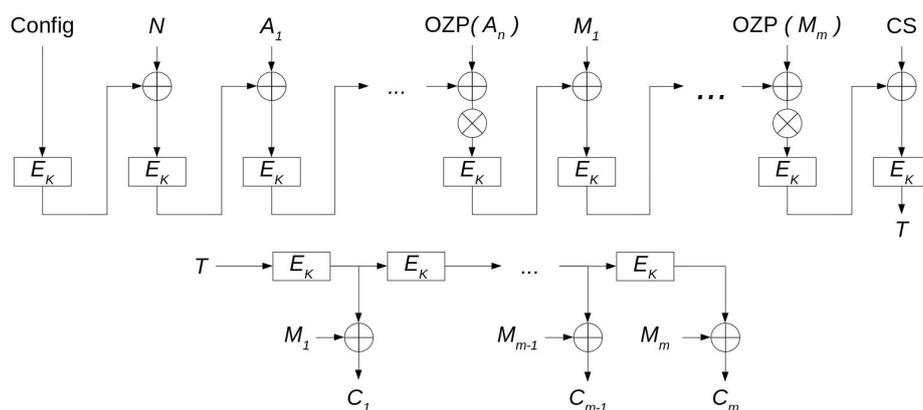


Figure 3: Encryption Scheme of TRIFLE.

The underlying block cipher used by TRIFLE, TRIFLE-BC (represented as  $E_k$  in Figure 3) is a 50-round 128-bit SPN block cipher. It receives a 128-bit plaintext  $X_{127}||X_{126}||\dots||X_0$

where  $X_i$  is a bit, and a 128-bit key  $K_7||K_6||\dots||K_0$  where  $K_i$  is a 16-bit word and produces a 128-bit ciphertext. Each round of TRIFLE-BC consists of four consecutive functions namely SubNibbles, BitPermutation, AddRoundKey, and AddRoundConstant. The four functions are detailed as follows:

**SubNibbles.** TRIFLE-BC uses an invertible 4-bit to 4-bit s-box  $S : \mathbb{F}_2^4 \rightarrow \mathbb{F}_2^4$ . The same 4-bit s-box is used throughout the cipher and is applied to every nibble of the cipher state. The mapping of the s-box is given in Table 1 using hexadecimal notation.

Table 1: The TRIFLE-BC s-box.

$x$	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
$S(x)$	0	C	9	7	3	5	E	4	6	B	A	2	D	1	8	F

**BitPermutation.** The bit permutation used in TRIFLE maps bits from bit position  $i$  of the cipher state to bit position  $P(i)$ . The bit permutation  $P(i)$  is defined as

$$P(i) = \lfloor i/4 \rfloor + (i\%4) \times 32.$$

**AddRoundKey.** In this stage, a 64-bit round key  $(K_4, K_5, K_1, K_0)$  is extracted from the key state. The extracted round key is then applied to the cipher state in an interleaved manner whereby

$$\begin{aligned} V_{31}||V_{30}||\dots||V_0 &\leftarrow K_1||K_0 \\ U_{31}||U_{30}||\dots||U_0 &\leftarrow K_4||K_5 \\ X_{4i+1} &\leftarrow X_{4i+1} \oplus V_i (0 \leq i \leq 31) \\ X_{4i+2} &\leftarrow X_{4i+2} \oplus U_i (0 \leq i \leq 31). \end{aligned}$$

The key state will be updated using simple word-wise rotations for the key state and bit-wise rotations within individual subdivided key states similar to the one defined in GIFT-128 [BPP<sup>+</sup>17]. The key schedule is defined as

$$K_7||K_6||\dots||K_0 \leftarrow K_1 \ggg 2 || K_0 \ggg 12 || K_7||K_6||\dots||K_2.$$

**AddRoundConstant.** A 6-bit round constant is XOR-ed into 6 different bit-states while a constant value of 1 is XOR-ed into the most significant bit  $X_{127}$  as shown in

$$\begin{aligned} X_{127} &\leftarrow X_{127} \oplus 1 \\ X_{23} &\leftarrow X_{23} \oplus C_5 \\ X_{19} &\leftarrow X_{19} \oplus C_4 \\ X_{15} &\leftarrow X_{15} \oplus C_3 \\ X_{11} &\leftarrow X_{11} \oplus C_2 \\ X_7 &\leftarrow X_7 \oplus C_1 \\ X_3 &\leftarrow X_3 \oplus C_0. \end{aligned}$$

The 6-bit round constant is then updated using the SKINNY's 6-bit affine LFSR function [BJK<sup>+</sup>16] defined as

$$C_5||C_4||\dots||C_0 \leftarrow C_4||C_3||\dots||C_0 || (C_5 \oplus C_4 \oplus 1).$$

### 2.2.3 Differential properties of TRIFLE-BC

By analyzing the differential distribution table of TRIFLE's s-box presented in Table 2, it is found that each  $\Delta U$  that has a hamming weight of a single bit ( $W_{bit} = 1$ ) can be mapped back differentially to  $\Delta V$  with  $W_{bit} = 1$ . These 1-bit to 1-bit differential relationships ( $1 \rightarrow 8, 2 \rightarrow 1, 4 \rightarrow 2$ , and  $8 \rightarrow 4$ ) hold with a probability of  $2^{-3}$ . The 1-bit  $\Delta V$  will be permuted and propagated to the next round to become yet another  $\Delta U$  with  $W_{bit} = 1$  due to the nature of bit permutation that shuffles bits without affecting the total number of active bits in the block cipher.

Therefore, for any  $n$  arbitrary rounds of TRIFLE, there exists a differential characteristic  $\Delta X(X_0, X_1, \dots, X_{31}) \rightarrow \Delta Y(Y_0, Y_1, \dots, Y_{31})$  such that  $W_{bit}(X_i^j) = 1$  where  $0 \leq i < 32, 0 \leq j < n$  and  $P(\Delta X \rightarrow \Delta Y) = 3^{-3n}$ . Moreover, there exist 4 differentials  $\Delta U \rightarrow \Delta V$  ( $7 \rightarrow 4, B \rightarrow 2, D \rightarrow 1$ , and  $E \rightarrow 8$ ) where  $W_{bit}(\Delta U) > 1, W_{bit}(\Delta V) = 1$  and  $P(\Delta U \rightarrow \Delta V) = 2^{-2}$ . This set of differentials can be used to improve the first round of the aforementioned single-bit differential characteristics to increase the probability to  $3^{-3n+1}$  for any  $n$  arbitrary rounds without affecting the intermediary single-bit differences propagation.

A similar improvement can be made to the final round of the single-bit differential characteristics by maximizing the probability of the final-round differential. Since there exists a  $\Delta V$  for every  $\Delta U$  with  $W_{bit}(\Delta U) = 1$  such that  $P(\Delta U \rightarrow \Delta V) = 2^{-2}$ , these differential relationships can be used at the final round without affecting the intermediary single-bit differences propagation as well.

Thus, the single-bit differential characteristics with improved first and final rounds that have a probability of  $3^{-3n+2}$  exist for any  $n$  arbitrary rounds of TRIFLE provided that  $n \geq 3$ . In fact, there are exactly 128 (128 different starting bit position) such characteristics for every round. The same differential behavior and its resulting differential characteristics had been observed and discussed in [FT19] and [NIS19b].

Based on the aforementioned improved single-bit differential characteristics, a key recovery strategy had been discussed in [FT19] that recovers the key for 11 rounds of TRIFLE with a time complexity and data complexity of  $2^{104}$  and  $2^{63}$  respectively. The authors proposed using a 42-round improved single-bit differential in their key recovery strategy on TRIFLE-BC. However, the authors made an error of using the 41-round ( $2^{-3(41)+2} = 2^{-121}$ ) differential probability in their calculation instead of 42 ( $2^{-3(42)+2} = 2^{-124}$ ). Therefore, the differential attack of TRIFLE-BC in [FT19] should be able to recover the secret key of a 43-round TRIFLE-BC (instead of 44 rounds) with the time and data complexity of  $2^{126}$ .

The differential discussed in this subsection only considers the probability of a single characteristic. The differential probability can be potentially improved by incorporating probability gains from the clustering effect (also referred to as the differential effect) shown in [NB13], whereby multiple differential characteristics with the same  $\Delta X \rightarrow \Delta Y$  are considered for the probability of a given differential.

## 3 Differential characteristics (cluster) search

Matsui proposed a branch-and-bound algorithm [Mat95] for searching linear paths and differential characteristics. The algorithm had been used on DES and found the best characteristic for it at the time. The algorithm relies on pruning bad branches that have less probability than the best probability found so far  $\overline{B}_n$ . The initial value of  $\overline{B}_n$  also helps break off bad branches in the early parts of the algorithm. Thus, when  $\overline{B}_n$  approaches the real value of the best probability  $B_n$  where  $\overline{B}_n \leq B_n$ , the search speed is improved as well. The algorithm also uses the knowledge of  $\overline{B}_{n-i}$  that is computed from round 0 to round  $i$  to estimate the probability of the current branch being searched. It will effectively cut off branches with probabilities that are estimated to be worse than  $\overline{B}_n$ .

Table 2: The TRIFLE-BC differential distribution table.

$\Delta U \backslash \Delta V$	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	16	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	2	2	2	0	2	0	4	2	2	0
2	0	2	0	0	0	2	0	0	0	4	0	2	2	2	2	0
3	0	2	2	0	2	2	0	4	0	2	0	2	0	0	0	0
4	0	0	2	4	0	0	0	2	0	2	2	2	0	2	0	0
5	0	0	0	0	0	4	2	2	0	2	2	0	0	2	0	2
6	0	0	2	2	2	0	0	2	2	0	2	0	0	0	4	0
7	0	0	2	2	4	0	0	0	0	2	0	2	2	0	0	2
8	0	0	0	2	2	2	4	2	0	0	0	2	0	0	2	0
9	0	2	2	0	0	0	0	0	2	0	2	4	2	2	0	0
A	0	0	0	2	0	2	0	0	0	0	4	2	2	0	2	2
B	0	2	4	0	0	0	2	0	0	2	0	0	2	2	0	2
C	0	2	0	0	2	2	2	0	2	0	0	0	0	4	2	0
D	0	4	0	2	0	0	2	0	2	0	0	0	2	0	2	2
E	0	0	0	2	2	0	2	2	4	2	0	0	0	0	0	2
F	0	2	2	0	2	2	0	0	2	0	2	0	0	0	0	4

Since then, several improvements had been made to Matsui’s algorithm. A cluster search algorithm such as [CMST15] improved upon the Matsui’s algorithm by searching for differential clusters after a main differential characteristic is identified. A differential cluster search includes all differential characteristics that share the same input  $\Delta X$  and output  $\Delta Y$  differences but with different intermediary differences. The searched cluster will improve upon the probability of the overall differential. This approach has successfully identified differentials with improved probability for block ciphers such as LBlock and TWINE [CMST15]. The resulting differentials have higher probabilities than prior ones. It is also worth noting that [CMST15] work used the number of active s-boxes as the pruning rules to eliminate bad branches.

Moreover, the work in [BV14] and [CTX<sup>+</sup>16] use a type of automatic search known as the threshold search. Although both of the aforementioned automated approaches are meant for ARX ciphers, the tendency of the algorithm to identify differences with high probabilities is worth noting as well.

A combination of the number of active s-boxes and the differential probability threshold will be used as the pruning rules for our proposed GPU-based automatic search. The combination of both allows greater flexibility during the search and also effectively cuts off more bad branches quickly if configured correctly. The proposed algorithm is shown in Algorithm 1.

## 4 GPU-accelerated automatic search for differential characteristics and their clusters

The search for differential clusters is computationally intensive. To facilitate the searching of differential clusters for block ciphers with a large block size and a large number of rounds (such as TRIFLE), the processing power of GPUs is leveraged to obtain a substantial performance boost to the conventional branch-and-bound searching algorithm. Although GPU-accelerated branch-and-bound algorithm had been studied in [LEB12] for knapsack, [MCMT12] for flow-shop scheduling, and [BHG17] for multiproduct batch plants

---

**Algorithm 1** Differential characteristics (cluster) searching algorithm with constraints on probability and number of active s-boxes.

---

**Input:** Input difference  $\Delta X$  and output difference  $\Delta Y$ .

**Output:** Probability  $P_c$  of  $\Delta X \rightarrow \Delta Y$  cluster.

**Adjustable Parameters:**

1.  $AS\_BOUND$  : Maximum of number of active sboxes for  $\Delta Y$ .
2.  $PROB\_BOUND$  : Maximum probability of  $\Delta X \rightarrow \Delta Y$ .
3.  $P_{AS}$  : Estimated probability of a nibble  $\Delta U \rightarrow \Delta V$ .

**procedure** CLUSTER\_SEARCH\_ROUND\_ $i$  ( $1 \leq i < n$ )

**for** each candidate  $\Delta Y_i$  **do**

$p_i \leftarrow Prob(\Delta X_i, \Delta Y_i)$

$AS_{i+1} \leftarrow W_{byte}(\Delta Y_i)$

**if**  $AS_{i+1} \leq AS\_BOUND$  **then**

$p_{i+1} \leftarrow (P_{AS})^{AS_{i+1}}$

$p_r \leftarrow (P_{AS})^{n-i-1}$

**if**  $[p_1, \dots, p_i, p_{i+1}, p_r] \geq PROB\_BOUND$  **then**

        call procedure CLUSTER\_SEARCH\_ROUND\_ $(i + 1)$

**end if**

**end if**

**end for**

**end procedure**

**procedure** CLUSTER\_SEARCH\_ROUND\_ $n$

**for** each candidate  $\Delta Y_n$  **do**

**if**  $\Delta Y_n == \Delta Y$  **then**

$p_n \leftarrow Prob(\Delta X_n, \Delta Y_n)$

$P_c \leftarrow P_c + [p_1, \dots, p_n]$

**end if**

**end for**

**end procedure**

---

optimization sub-problems, there exists no prior work that uses GPU to accelerate the branch-and-bound search for differential cryptanalysis. Thus the proposed work is the novel application of such an approach.

The proposed GPU-accelerated algorithm is a variant of the depth-first search algorithm whereby the algorithm will first visit nodes (possible branches) in the following round before backtracking to visit other nodes. The difference is that once a node is visited, all of its corresponding child branches are enumerated. This enables the task of enumeration for the relevant child branches and subsequently the evaluation of the pruning rules to be parallelized and solved by the GPU. All this can be performed while keeping the memory footprint to a manageable range by enumerating one branch at a time rather than all possible branches of a particular depth at once (breadth-first search). The exception exists for the final round of the search whereby all of the branches are visited and evaluated simultaneously. The behaviour of the modified depth-first search algorithm is illustrated in Figure 4.

However, if the total number of possible child branches for a particular difference pattern is too low, then it will cause the GPU kernel to have low efficiency due to low occupancy (insufficient tasks to be distributed across multiprocessors). In the proposed algorithm for TRIFLE, this scenario occurs when the number of active s-boxes for a particular difference is  $< 4$ . To alleviate this problem, differences with a low number of

possible branches are instead enumerated and evaluated by the CPU-variant procedure. The GPU kernel and its CPU-variant are discussed in subsection 4.1. The complete algorithm for the proposed GPU-accelerated branch-and-bound differential cluster search without enumeration kernels and method details is provided in Algorithm 2. Note that the correctness of the proposed algorithm had been verified by comparing the results of Algorithms 1 and 2.

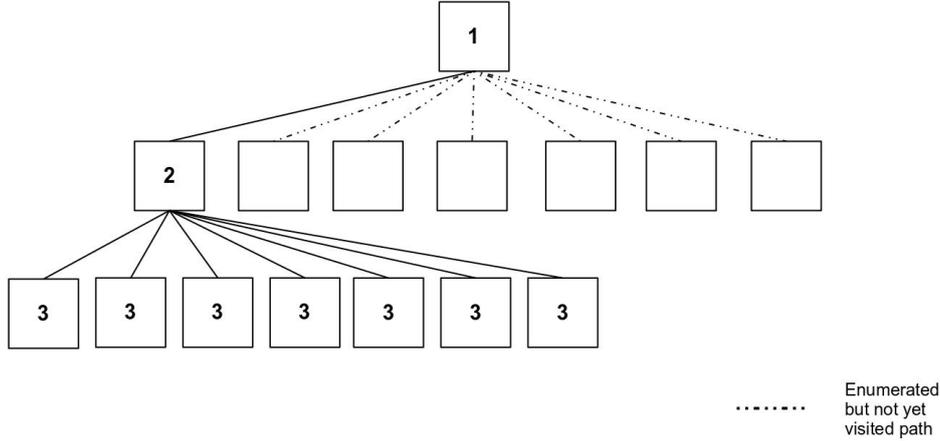


Figure 4: The searching strategy for the proposed algorithm.

#### 4.1 Enumeration using GPU Kernel or CPU Method

The GPU Kernel has been specifically optimized for TRIFLE's structure which has a constant branching number of 7 for  $\Delta V$ . This means that  $\forall \Delta U$  that goes through the TRIFLE's s-box, there are precisely 7 possible choices of  $\Delta V$ . Despite the specific optimization used, the kernel should be able to be generalized to any SPN block cipher while still retaining a similar efficiency by estimating the correct number of branches and assigning workload among the threads accordingly.

The configuration of the proposed GPU architecture will utilize 1D blocks for each kernel launch. Since each block within a grid contains its own block threads, each thread is assigned a unique thread id based on its position in a given grid. This thread id assignment facilitates the process of work distribution and reduction. For TRIFLE, the number of possible branches of  $\Delta X_i$  is  $7^{AS_i}$ . When  $AS_i = 4$ , there are 2401 tasks to be distributed. 19 blocks ( $> 9$  SMs in NVIDIA GTX-1060) are declared for a grid and each block contains 128 threads (32|128) totalling up to 2432 threads (excess threads are terminated during runtime immediately).

Let  $NB_1, NB_2, NB_3, NB_4$  be the number of possible branches, and  $I_1, I_2, I_3, I_4$  be the  $n^{th}$  numbered branches in the four active  $\Delta U$  branches respectively. Thread id,  $T_i$  can also be computed as

$$ID(I_1, I_2, I_3, I_4) = (I_1 \times NB_0) + (I_2 \times \prod_{i=0}^1 NB_i) + (I_3 \times \prod_{i=0}^2 NB_i) + (I_4 \times \prod_{i=0}^3 NB_i), \quad (1)$$

where  $NB_0 = 1$ . The work assignment (the branch taken by each individual thread) is done by computing  $ID^{-1}(T_i)$ . For  $AS_i > 4$ , the work assignment will still occur for the first four

---

**Algorithm 2** GPU-accelerated differential (cluster) searching algorithm.

---

**Input:** Input difference  $\Delta X$  and output difference  $\Delta Y$ .  
**Output:** Probability  $P_c$  of  $\Delta X \rightarrow \Delta Y$  cluster.

```

procedure CLUSTER_SEARCH
  allocate device memory
  allocate and pin host memory
  call procedure CLUSTER_SEARCH_ROUND_0
  copy  $P_i$  from device to host
   $P_c \leftarrow (\sum_{i=1}^{T_{total}} P_i) + P_h$ 
end procedure

procedure CLUSTER_SEARCH_ROUND_ $i$  ( $1 \leq i < n$ )
  //Enumerate all possible branches of  $\Delta Y_i$ 
   $AS_i \leftarrow W_{byte}(\Delta X_i)$ 
  if  $AS_i > 3$  then
    call procedure ENUMERATION_DEVICE_ $i$ 
  else
    call procedure ENUMERATION_HOST_ $i$ 
  end if
  //Prune or proceed based on enumerated branches and their evaluation results
  for each computed  $\Delta Y_i^j$  do
    if  $(\Delta Y_{condition})_i^j == \text{TRUE}$  then
      if  $i + 1 < N$  then
        call procedure CLUSTER_SEARCH_ROUND_( $i + 1$ )
      else
        if  $AS_{i+1} > 3$  and  $AS_i > 3$  then
          call procedure ENUMERATION_DEVICE_ $n$ 
        else
          call procedure ENUMERATION_HOST_ $n$ 
        end if
      end if
    end if
  end for
end procedure

```

---

active  $\Delta U$  branches, but the remaining active  $\Delta U$  branches are exhaustively enumerated by each working thread individually. The last round follows the same logic of Algorithm 1 whereby after a branch (now a trail) is enumerated,  $\Delta Y_n == \Delta Y$  is checked, then  $P_i$  is incremented accordingly. To avoid race conditions, each thread has its own probability accumulator,  $P_i$ . The final cluster probability,  $P_c = \sum_{i=1}^{T_{total}} P_i + P_h$  is computed in the host procedure where  $P_h$  is the host probability accumulator.

Special attention needs to be given to memory management. All of the necessary device memory allocation and host memory pinning are done during program initialization. Both the allocated memory and pinned memory are reused whenever possible since allocation and de-allocation of the memory are expensive and will impact the overall efficiency of the proposed algorithm. DDT and permutation lookup tables are specifically loaded into the shared memory each time the kernel is launched because the improved latency of the shared memory will ease the frequent access of the DDT and permutation table. The complete algorithm for the kernel is summarized in Algorithm 4.

The GPU kernel can only be used when there is a large number of branches to maintain high GPU utilization. For  $AS \leq 3$ , a CPU-version of enumeration method is used instead.

---

**Algorithm 3** Host (CPU) enumeration and evaluation method.
 

---

**Input:** Input Difference  $\Delta X$ .**Output:** Enumerated branches, its evaluation result and probabilities  $P_i$ .**Adjustable Parameters:**

1.  $AS\_BOUND$  : Maximum of number of active sboxes for  $\Delta Y$ .
2.  $PROB\_BOUND$  : Maximum probability of  $\Delta X \rightarrow \Delta Y$ .
3.  $P_{AS}$  : Estimated probability of a nibble  $\Delta U \rightarrow \Delta V$ .

```

procedure ENUMERATION_HOST_ $i$  ( $1 \leq i \leq n$ )
  for each candidate  $(\Delta AV_1, \Delta AV_2, \dots, \Delta AV_{AS\_BOUND})$  do
    if  $i \neq n$  then
       $(\Delta Y_{condition})_i^{candidate\_index} \leftarrow FALSE$ 
       $p_i \leftarrow Prob(\Delta X_i, \Delta Y_i)$ 
       $AS_{i+1} \leftarrow W_{byte}(\Delta Y_i)$ 
      if  $AS_{i+1} \leq AS\_BOUND$  then
         $p_{i+1} \leftarrow (P_{AS})^{AS_{i+1}}$ 
         $p_r \leftarrow (P_{AS})^{n-i-1}$ 
        if  $[p_1, \dots, p_i, p_{i+1}, p_r] \geq PROB\_BOUND$  then
           $(\Delta Y_{condition})_i^{candidate\_index} \leftarrow TRUE$ 
        end if
      end if
    else if  $\Delta Y_i == \Delta Y$  then
       $P_i \leftarrow P_i + p_i$ 
    end if
  end for
end procedure

```

---

The CPU-version follows the general logic of the GPU kernel without parallelized processing. The complete CPU enumeration method is shown in Algorithm 3.

## 4.2 Meet-in-the-middle searching approach

The meet-in-the-middle (MITM) approach described in [CMST15] is used to further improve the efficiency of the search. Since the number of branches grows exponentially as the number of rounds increases, the search for large number of rounds could be completed much quicker if the number of rounds to search is split between  $\alpha$  rounds and  $\beta$  rounds instead of searching directly for  $(\alpha + \beta)$  rounds. This is underlying property behind the efficiency improvements achieved by MITM.

The steps involved in the MITM approach starts off by dividing the search into forward  $\alpha$  rounds and backward  $\beta$  rounds. For the forward search, the proposed algorithm mentioned in Algorithm 2 is used. The difference is that during the  $\alpha^{th}$  (final) round, instead of evaluating  $\Delta Y_\alpha$ , the  $\Delta Y_\alpha$  and its probability is accumulated in an array for matching purposes. Since the amount of information needed to store all of the possible permutations of 128-bit data far exceeds the practical memory storage option currently available, an encoding method is used to index into the array. The encoding is computed by using the format of  $[Pos_{\Delta AV_i}, \Delta AV_i, Pos_{\Delta AV_{i+1}}, \Delta AV_{i+1}, Pos_{\Delta AV_{i+2}}, \Delta AV_{i+2}]$ . The total number of nibbles to be stored is currently limited to a maximum of 3 (12 bits). Since each nibble requires 5 bits to represent its nibble position, thus the total number of bits needed to represent 3 nibbles among 32 possible nibble positions is 27 bits. This amounts to an array size of 134217728 that requires 1.07 GB of memory when using a

64-bit double-precision floating point format to store the probability.

Meanwhile, the backward search requires the computation of a reversed DDT and the corresponding reversed permutation table. During the  $\beta^{th}$  (final) round,  $\Delta Y_\beta$  is encoded using the same method described earlier to index into the storage array to check for matching trails. Matching trails contribute toward the final cluster probability  $P_c$ . The MITM approach detailed in this section is illustrated in Figure 5.

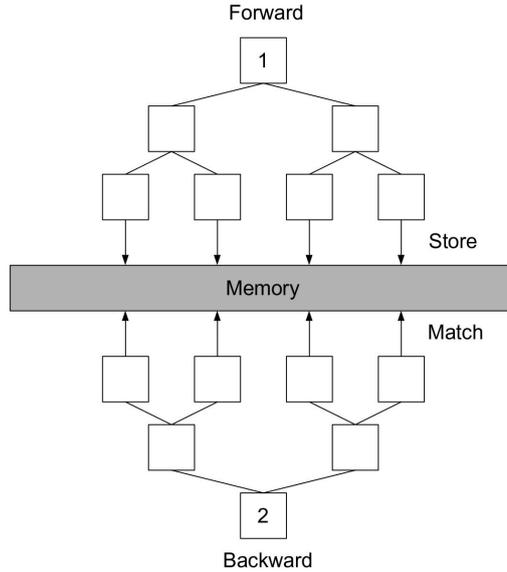


Figure 5: Meet-in-the-middle approach.

### 4.3 Performance comparison of GPU and CPU-based automatic search for differential algorithms

The CPU and GPU algorithms are implemented using C++ and CUDA/C respectively. The performance results are obtained by running the implementations on a single Linux desktop computer with Intel 6<sup>th</sup> generation Skylake Core i5-6600K CPU clocked at 3.5 GHz, NVIDIA Pascal GeForce GTX-1060 with 3 GB memory, and 16 GB of RAM.

A fixed problem set which satisfies a specific  $W_{byte}(\Delta X)$  criteria has been computed on both the GPU-accelerated kernel and CPU-enumeration method. The results obtained (including the time spent on memory transfer) are recorded in Table 3 and is an average of a hundred instances. These results show the potential of the performance improvement of the GPU-accelerated functions which can be up to a factor of 5.95 over the CPU-enumeration method. Also, if the GPU possesses higher on-chip memory whereby the necessary computing differential caches are able to fit, it is possible for the proposed algorithm to reach a speedup of up to 27.07 as shown in Table 4. The 3 GB GPU that used in the practical experiment setup cannot accomplish such a task because to hold 22-rounds of maximum possible enumerated branches used in  $\alpha$ -round of the MITM approach, it would require approximately 11 GB of memory, but a more powerful workstation GPU with  $\geq 16$  GB memory could easily accomplish this.

A series of practical tests of the proposed algorithm is performed on various rounds of TRIFLE-BC. The results are recorded in Table 5 and these results are bounded by  $PROB\_BOUND = P_t \times 2^{-21}$  and are an average of ten instances. It can be seen that although the algorithm depict a speed-up of 5.95, as the number of rounds increases, the

performance also increases and stabilizes at approximately 2.5. This result is obtained because the computation is not GPU-accelerated when the number of active s-boxes is between 1 and 3. It can also be noted that the MITM approach greatly increases the performance of the searching algorithm over the traditional recursive method for up to a factor of approximately 60 at round 20.

Table 3: Search time ( $\mu s$ ) comparison of CPU enumeration and GPU kernel enumeration.

$W_{byte}(\Delta X)$	GPU-Accelerated	CPU-Enumeration	Speedup
4	35.5	173.7	4.89
5	141.2	716.6	5.08
6	861.4	4589.0	5.33
7	5974.9	32 200.4	5.39
8	41 561.5	247 393.0	5.95

Table 4: Search time ( $\mu s$ ) comparison of CPU enumeration and GPU kernel enumeration (without output memory synchronization).

$W_{byte}(\Delta X)$	GPU-Accelerated	CPU-Enumeration	Speedup
4	34.2	173.7	5.08
5	70.3	716.6	10.19
6	286.9	4589.0	16.00
7	1656.7	32 200.4	19.44
8	9140.1	247 393.0	27.07

Table 5: Search time ( $ms$ ) of various rounds of TRIFLE-BC.

Round(s)	MITM GPU-Accel	GPU-Accelerated	CPU-Enum	Recursive
5	1135.9	661.5	787.2	855.6
10	2197.3	8644.5	19 564.6	21 808.7
15	8795.6	62 928.7	156 725.0	165 202.3
20	15 675.2	363 274.2	908 978.1	941 195.2

#### 4.4 Limitations and capabilities of the proposed algorithm.

The proposed algorithm presented in this paper is not without its limitations. Firstly, the kernel is only utilized when AS of  $\Delta X$  is  $\geq 4$ . It should be theoretically possible to bundle several small work units into a large compiled work unit to be sent to kernel for processing. The added benefit of this is the higher performance gains for cases of  $AS\_BOUND < 8$ , which could achieve a speedup equivalent to using  $AS\_BOUND = 8$ . Doing so will definitely incur more overhead. Thus, the feasibility of such an idea may be studied in future work.

In addition, the maximum number of active s-boxes for any  $\Delta X$  had to be limited. In this case, it is limited to below 9 AS for TRIFLE. Since the memory growth is exponential as the number of AS increases, for the case where  $AS \geq 9$ , it can be seen that it requires far more than 16-GB of RAM and is thus infeasible for the target machine. A similar problem is faced by the MITM approach whereby the differences in the middle meeting point must be limited to a maximum of 3 AS, while the differences outside the meeting point remain unaffected. Due to these limitations, some differential characteristics that contain intermediary differences with the number of AS larger than the specified amount will be unreachable by the proposed algorithm. However, the memory limitation could be alleviated to some extent by writing the necessary intermediary differences to a file.

Luckily for block ciphers with smaller block sizes, these properties are less detrimental to the overall paths covered by the search.

This method also requires a large amount of memory especially as compared to a recursive version of the algorithm shown in Algorithm 1. The dependency on the GPU hardware requires some tweaking on the number of blocks and the number of threads per block so that the GPU utilization could be maximized. Currently, the proposed algorithm requires some customization to be applicable to other SPN block ciphers. Its feasibility for other types of block ciphers such as ARX and Feistel will be investigated in future work. Further work is also needed to generalize the proposed algorithm for SPN block ciphers with minimal modifications.

With that said, the proposed algorithm is able to use GPU hardware to shorten the searching runtime drastically. This enables the automated search to be conducted for block ciphers with large block sizes (128-bit) at a high number of rounds ( $\geq 30$ ). This has not been practically attempted in previous works. It is also possible for the work to be adapted to block ciphers with super s-boxes. The possibility of distributing the workload of the proposed algorithm across a grid or grids of CPU-GPU computing nodes makes it possible to enhance the efficiency of the search even further. For example, by enumerating all the second or third level branches in a breadth-first manner, these branches can be divided into individual work items that can be distributed across CPU-GPU computing nodes. This also requires the modification of the proposed algorithm to be able to utilize more CPU cores to better utilize the available computing resources.

#### 4.5 Summary of the proposed algorithm

In short, by utilizing a GPU to accelerate the automatic search for differential clusters, a substantial performance boost up to a factor of 5.95 could be obtained, however the practical runtime shows that the performance increase is around the range of 2.5 for a large number of rounds with  $AS\_BOUND = 4$ . Although the proposed algorithm is able to search for differential clusters of block ciphers with large block size and a high number of rounds, memory limitation is incurred by both the immediate complete enumeration of branches once a node is visited and MITM approach. This memory limitation affects the number of potential branches to be explored and consequently some combinations of differences are unreachable. A small portion of the enumeration tasks still needs to be performed on the host. The tasks split between host and device are summarised as below:

##### Tasks performed on host (CPU)

- Memory management and synchronization.
- Launching of the GPU kernel.
- Enumeration and evaluation of branches for  $\Delta X$  when the enumerable range is too small, for TRIFLE it is when  $AS < 4$ .

##### Tasks performed on device (GPU)

- Enumeration of branches for  $\Delta X$  for a large dataset in parallel.
- Evaluate the enumerated branches.

## 5 Differential clustering effect of TRIFLE-BC

The proposed algorithm has been used to study the differential cluster effect in TRIFLE-BC. The 128 improved single-bit differences propagation trails described in Subsection 2.2.3

Table 6: Differential for 20-round TRIFLE-BC bounded by  
 $AS\_BOUND = 4$  and  $PROB\_BOUND = P_t * 2^{-31}$

$\Delta X$	$\Delta Y$	$P_t$	$P_c$	# of Trails
0000 0000 0000 0000	0000 0000 0000 0000	$2^{-58}$	$2^{-57.97}$	44221306
00b0 0000 0000 0000	0000 0001 0000 0001			
0000 0000 0000 0000	0000 0100 0000 0100	$2^{-58}$	$2^{-57.97}$	36278752
0000 d000 0000 0000	0000 0000 0000 0000			
0000 0000 0000 0000	0000 0000 0000 0000	$2^{-58}$	$2^{-57.97}$	47516983
0000 0000 0700 0000	0000 0002 0000 0002			
0000 0000 0000 0000	0000 0200 0000 0200	$2^{-58}$	$2^{-57.97}$	32376080
0000 0000 0000 000e	0000 0000 0000 0000			
0000 0000 0000 0000	0000 0000 0000 0000	$2^{-58}$	$2^{-57.996}$	27608539
0000 0b00 0000 0000	0000 0400 0000 0400			

Table 7: Differential for 43-round TRIFLE-BC bounded by  
 $AS\_BOUND = 4$  and  $PROB\_BOUND = P_t * 2^{-21}$

$\Delta X$	$\Delta Y$	$P_t$	$P_c$	# of Trails
0000 0000 0000 b000	0000 0000 0010 0000	$2^{-127}$	$2^{-126.931}$	2202258
0000 0000 0000 0000	0010 0000 0000 0000			
0000 0000 0000 0000	0000 0002 0000 0002	$2^{-127}$	$2^{-126.931}$	2197251
b000 0000 0000 0000	0000 0000 0000 0000			
0000 0000 0000 0000	0020 0000 0020 0000	$2^{-127}$	$2^{-126.931}$	2226164
0007 0000 0000 0000	0000 0000 0000 0000			
0000 0000 0000 0000	0000 0000 0000 0000	$2^{-127}$	$2^{-126.931}$	2034427
0000 0007 0000 0000	0000 0200 0000 0200			
0000 0000 0000 0000	0000 0000 0000 0000	$2^{-127}$	$2^{-126.995}$	1504844
0000 0b00 0000 0000	0000 0400 0000 0400			

are clustered using the proposed algorithm. Cluster searches are conducted for 43-round TRIFLE using  $AS\_BOUND = 4$  and  $PROB\_BOUND = P_t * 2^{-21}$ . The time required to complete the search is one and a half days using the desktop computer described in Subsection 4.3.

A equivalent search is conducted for 20-round TRIFLE using  $AS\_BOUND = 4$  and  $PROB\_BOUND = P_t * 2^{-31}$ . A slightly higher bound is used here in an attempt to cluster more differential trails. The time needed to complete the search is half a day using the desktop computer described in Subsection 4.3.

Since the differential probabilities are similar, we select only 5 differentials with 4 being the best probability and 1 differential being the differential described in [FT19] to show in Table 6 and Table 7. We found that the effect of clustering these paths do not significantly improve the probability. However, large differential clusters could be enumerated, consisting of up to 44 and 2.2 million trails for 20-round and 43-round TRIFLE-BC respectively. The differential used in [FT19] for a key recovery attack against TRIFLE can be improved slightly from  $2^{-58}$  to  $2^{-57.996}$ .

The improved efficiency of the searching algorithm allows for practical identification of large clusters. Although the large clusters found in TRIFLE did not contribute to significant improvements in terms of differential probability, this may not be the case for other block ciphers, especially block ciphers with smaller block size. When the block size is larger, the differential probability is distributed into more trails, whereby the number of possible trails is a factor of  $2^{64}$  more than lightweight block ciphers. Meanwhile, when the block size is smaller, the probability of each trail is, by comparison, much larger. Thus, the proposed searching algorithm can be used to more accurately determine the security

margin of these ciphers, and also provide a detailed look at their clustering effects.

## 6 Conclusion

In this work, a new GPU-accelerated branch-and-bound algorithm for differential cluster search of block ciphers has been proposed. This method improves upon the existing branch-and-bound method by utilizing the parallel processing power of GPU hardware for speedup. The proposed algorithm can achieve a substantial speedup up to a factor of approximately 5.95. The proposed method will be useful in differential cryptanalysis to identify large differential clusters that can contribute to higher differential probabilities. The GPU-accelerated algorithm can be adapted to suit other SPN block ciphers by changing the permutation and differential distribution table and customizing the kernel thread number based on the GPU hardware capability. However, other block cipher structures such as Feistel and ARX more work still has to be done with respect to the feasibility of the proposed approach. With the assistance of this developed GPU framework, we provide a detailed look at the clustering effect of the authenticated cipher TRIFLE, which also served to showcase the practicality of the proposed framework.

## Acknowledgement

To be included.

## References

- [BHG17] Andrey Borisenko, Michael Haidl, and Sergei Gorlatch. A GPU parallelization of branch-and-bound for multiproduct batch plants optimization. *The Journal of Supercomputing*, 73(2):639–651, February 2017.
- [BJK<sup>+</sup>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, pages 123–153. Springer Berlin Heidelberg, Berlin, Heidelberg, 2016.
- [BKL<sup>+</sup>07] A. Bogdanov, L. R. Knudsen, G. Leander, C. Paar, A. Poschmann, M. J. B. Robshaw, Y. Seurin, and C. Vikkelsoe. PRESENT: An Ultra-Lightweight Block Cipher. In Pascal Paillier and Ingrid Verbauwhede, editors, *Cryptographic Hardware and Embedded Systems - CHES 2007*, volume 4727, pages 450–466. Springer Berlin Heidelberg, Berlin, Heidelberg, 2007.
- [BN10] Alex Biryukov and Ivica Nikolić. Automatic Search for Related-Key Differential Characteristics in Byte-Oriented Block Ciphers: Application to AES, Camellia, Khazad and Others. In David Hutchison, Takeo Kanade, Josef Kittler, Jon M. Kleinberg, Friedemann Mattern, John C. Mitchell, Moni Naor, Oscar Nierstrasz, C. Pandu Rangan, Bernhard Steffen, Madhu Sudan, Demetri Terzopoulos, Doug Tygar, Moshe Y. Vardi, Gerhard Weikum, and Henri Gilbert, editors, *Advances in Cryptology – EUROCRYPT 2010*, volume 6110, pages 322–344. Springer Berlin Heidelberg, Berlin, Heidelberg, 2010.
- [BPP<sup>+</sup>17] Subhadeep Banik, Sumit Kumar Pandey, Thomas Peyrin, Yu Sasaki, Siang Meng Sim, and Yosuke Todo. GIFT: A Small Present. In Wieland

- Fischer and Naofumi Homma, editors, *Cryptographic Hardware and Embedded Systems – CHES 2017*, volume 10529, pages 321–345. Springer International Publishing, Cham, 2017.
- [BS91] Eli Biham and Adi Shamir. Differential cryptanalysis of DES-like cryptosystems. *Journal of Cryptology*, 4(1):3–72, 1991.
- [BV14] Alex Biryukov and Vesselin Velichkov. Automatic Search for Differential Trails in ARX Ciphers. In David Hutchison, Takeo Kanade, Josef Kittler, Jon M. Kleinberg, Friedemann Mattern, John C. Mitchell, Moni Naor, Oscar Nierstrasz, C. Pandu Rangan, Bernhard Steffen, Madhu Sudan, Demetri Terzopoulos, Doug Tygar, Moshe Y. Vardi, Gerhard Weikum, and Josh Benaloh, editors, *Topics in Cryptology – CT-RSA 2014*, volume 8366, pages 227–250. Springer International Publishing, Cham, 2014.
- [CMST15] Jiageng Chen, Atsuko Miyaji, Chunhua Su, and Jesen Teh. Improved Differential Characteristic Searching Methods. In *2015 IEEE 2nd International Conference on Cyber Security and Cloud Computing*, pages 500–508, New York, NY, USA, November 2015. IEEE.
- [CTX<sup>+</sup>16] Kaihao Chen, Xiaoming Tang, Peng Xu, Man Guo, Weidong Qiu, and Zheng Gong. An Improved Automatic Search Method for Differential Trails in TEA Cipher. *International Journal of Network Security*, 18(4):644–649, July 2016.
- [EAY19] Muhammad ElSheikh, Ahmed Abdelkhalek, and Amr M. Youssef. On MILP-Based Automatic Search for Differential Trails Through Modular Additions with Application to Bel-T. In Johannes Buchmann, Abderrahmane Nitaj, and Tajjeeddine Rachidi, editors, *Progress in Cryptology – AFRICACRYPT 2019*, volume 11627, pages 273–296. Springer International Publishing, Cham, 2019.
- [FT19] Liu Fukang and Isobe Takanori. Iterative Differential Characteristic of TRIFLE-BC, 2019.
- [GPPR11] Jian Guo, Thomas Peyrin, Axel Poschmann, and Matt Robshaw. The LED Block Cipher. *Cryptographic Hardware and Embedded Systems – CHES 2011*, pages 326–341, 2011.
- [LEB12] Mohamed Esseghir Lalami and Didier El-Baz. GPU Implementation of the Branch and Bound Method for Knapsack Problems. In *2012 IEEE 26th International Parallel and Distributed Processing Symposium Workshops & PhD Forum*, pages 1769–1777, Shanghai, China, May 2012. IEEE.
- [Mat95] Mitsuru Matsui. On correlation between the order of S-boxes and the strength of DES. In Gerhard Goos, Juris Hartmanis, Jan van Leeuwen, and Alfredo De Santis, editors, *Advances in Cryptology — EUROCRYPT’94*, volume 950, pages 366–375. Springer Berlin Heidelberg, Berlin, Heidelberg, 1995.
- [MCMT12] N. Melab, I. Chakroun, M. Mezmaç, and D. Tuyttens. A GPU-accelerated Branch-and-Bound Algorithm for the Flow-Shop Scheduling Problem. In *2012 IEEE International Conference on Cluster Computing*, pages 10–17, Beijing, China, September 2012. IEEE.
- [MWGP12] Nicky Mouha, Qingju Wang, Dawu Gu, and Bart Preneel. Differential and Linear Cryptanalysis Using Mixed-Integer Linear Programming. In Chuan-Kun Wu, Moti Yung, and Dongdai Lin, editors, *Information Security and Cryptology*, volume 7537, pages 57–76. Springer Berlin Heidelberg, Berlin, Heidelberg, 2012.

- [NAD<sup>+</sup>19] Datta Nilanjan, Ghoshal Ashrujit, Mukhopadhyay Debdeep, Patranabis Sikhar, Picek Stjepan, and Sadhukhan Rajat. TRIFLE, March 2019.
- [NB13] Mouha Nicky and Preneel Bart. Towards Finding Optimal Differential Characteristics for ARX: Application to Salsa20. *Cryptology ePrint Archive, Report 2013/328*, 2013.
- [NIS19a] NIST. Lightweight Cryptography, Round-1 Candidates, April 2019.
- [NIS19b] NIST. Round 1 Lightweight Cryptography | Official Comments - TRIFLE, 2019.
- [NVI19] NVIDIA. CUDA C Programming Guide Version 9.0, October 2019.
- [Pad11] David Padua, editor. *Encyclopedia of Parallel Computing*. Springer US, Boston, MA, 2011.
- [SHS<sup>+</sup>14] Siwei Sun, Lei Hu, Ling Song, Yonghong Xie, and Peng Wang. Automatic Security Evaluation of Block Ciphers with S-bP Structures Against Related-Key Differential Attacks. In Dongdai Lin, Shouhuai Xu, and Moti Yung, editors, *Information Security and Cryptology*, volume 8567, pages 39–51. Springer International Publishing, Cham, 2014.
- [SHY16] Ling Song, Zhangjie Huang, and Qianqian Yang. Automatic Differential Analysis of ARX Block Ciphers with Application to SPECK and LEA. In Joseph K. Liu and Ron Steinfeld, editors, *Information Security and Privacy*, volume 9723, pages 379–394. Springer International Publishing, Cham, 2016.
- [SLM<sup>+</sup>14] Sun Siwei, Hu Lei, Wang Meiqin, Wang Peng, Qiao Kexin, Ma Xiaoshuang, Shi Danping, Song Ling, and Fu Kai. Towards Finding the Best Characteristics of Some Bit-oriented Block Ciphers and Automatic Enumeration of (Related-key) Differential and Linear Characteristics with Predefined Properties, 2014.

---

**Algorithm 4** Device (GPU) enumeration and evaluation method.
 

---

**Input:** Input Difference  $\Delta X$ .

**Output:** Enumerated branches, its evaluation result and probabilities  $P_i$ .

**Adjustable Parameters:**

1.  $AS\_BOUND$  : Maximum of number of active s-boxes for  $\Delta Y$ .
2.  $PROB\_BOUND$  : Maximum probability of  $\Delta X \rightarrow \Delta Y$ .
3.  $P_{AS}$  : Estimated probability of a nibble  $\Delta U \rightarrow \Delta V$ .

**Assumption:**

1. Non-active nibble (s-boxes) will have a difference value of zero. Thus, an attempt to differentially substitute it will yield  $0 \rightarrow 0$  with a probability of 1.

**procedure** ENUMERATION\_DEVICE\_  $i$  ( $1 \leq i \leq n$ )

synchronize necessary information with device memory (asynchronously)

 call  $KERNEL\_i$  (asynchronously)

synchronize device information with host memory (asynchronously)

cuda stream synchronized (wait for device to complete its computation)

**end procedure**
**procedure** KERNEL\_  $i$  ( $1 \leq i \leq n$ )

copy permutation table, sorted DDT (Descending Frequency), and branch size table to shared memory

 $T_i \leftarrow (blockIdx.x \times blockDim.x + threadIdx.x)$ 

//Work assignment

 $Value \leftarrow T_i, Divide\_Value \leftarrow 1$ 
**for** each active nibble values,  $\Delta AU_i$  where  $1 \leq i \leq 4$  **do**
 $I_i \leftarrow \lfloor Value / Divide\_Value \rfloor \bmod NB_i$ 
 $\Delta AV_i \leftarrow$  sorted DDT[ $\Delta AU_i$ ][ $I_i$ ]

 update  $p_i$ 
 $Divide\_Value \leftarrow Divide\_Value \times NB_i$ 
**end for**

 //Enumerating all remaining branches if  $AS\_BOUND > 4$ 

 //Note that the for loop will still be entered even if  $AU_5 = 0$ 
**for** each candidate ( $\Delta AV_5, \Delta AV_6, \dots, \Delta AV_{AS\_BOUND}$ ) **do**
**if**  $i \neq n$  **then**
 $global\_offset \leftarrow (\prod_{j=1}^{AS\_BOUND} NB_j \times T_i + candidate\_index)$ 
 $(\Delta Y_{condition})_i^{global\_offset} \leftarrow FALSE$ 
 $p_i \leftarrow Prob(\Delta X_i, \Delta Y_i)$ 
 $AS_{i+1} \leftarrow W_{byte}(\Delta Y_i)$ 
**if**  $AS_{i+1} \leq AS\_BOUND$  **then**
 $p_{i+1} \leftarrow (P_{AS})^{AS_{i+1}}$ 
 $p_r \leftarrow (P_{AS})^{n-i-1}$ 
**if**  $[p_1, \dots, p_i, p_{i+1}, p_r] \geq PROB\_BOUND$  **then**
 $(\Delta Y_{condition})_i^{global\_offset} \leftarrow TRUE$ 
**end if**
**end if**
**else if**  $\Delta Y_i == \Delta Y$  **then**
 $P_i \leftarrow P_i + p_i$ 
**end if**
**end for**
**end procedure**


---