Automated Search for Block Cipher Differentials: A GPU-Accelerated Branch-and-Bound Algorithm

Wei-Zhu Yeoh[1], Je Sen Teh[1], and Jiageng Chen[2]

[1] Universiti Sains Malaysia, Malaysia
yeohweizhu@gmail.com , jesen_teh@usm.my
[2] Central China Normal University, China
chinkako@gmail.com

Abstract. In this paper, we propose a GPU-accelerated branch-and-bound algorithm. The proposed approach 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. To showcase its practicality, the proposed algorithm is applied on TRIFLE-BC, a 128-bit block cipher. By incorporating a meet-in-the-middle approach with the proposed GPU kernel, we were able to improve the search efficiency (on 20 rounds of TRIFLE-BC) by approximately 58 times as compared to the CPU-based approach. Differentials consisting of up to 50 million individual characteristics can be constructed for 20 rounds of TRIFLE, leading to slight improvements to the overall differential probabilities. Even for larger rounds (43 rounds), the proposed algorithm is still able to construct large clusters of over 500 thousand characteristics. This result depicts the practicality of the proposed algorithm in constructing large differentials even for a 128-bit block cipher, which could be used to improve cryptanalytic findings against other block ciphers in the future. [3]

Keywords: Automated search · block cipher · branch-and-bound · cryptanalysis · differential characteristic · differential cluster · GPU

1 Introduction

Differential cryptanalysis is one of the most widely-known cryptanalytical methods, resistance to which has become a basic requirement for modern block ciphers [1,9]. The success of differential cryptanalysis relies on identifying differential characteristics that occur with high probability. The search for these characteristics is a non-trivial task especially for block ciphers with large block sizes and

[3] This is a pre-print version of the paper that has been published in Lecture Notes in Computer Science for ACISP 2020. The final authenticated version is available at https://doi.org/10.1007/978-3-030-55304-3_9.

number of rounds. In addition, differential cryptanalysis also takes into consideration differentials (clusters of single characteristics) for a more accurate estimate of the overall differential probability [4] [10].

Recently automated search for differential characteristics has been used instead of manual searching. Matsui [12] proposed a 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 [3] an ARX version of the branch-and-bound searching algorithm was proposed and the algorithm was also subsequently improved in [6] by the introduction of a sorted partial differential distribution table. In addition, [5] incorporated a meet-in-the-middle approach to the differential cluster search, and updated the pruning rules to bound the number of active of s-boxes to further improve upon the search efficiency.

In [14], 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 [23] to be applicable to bit-oriented block ciphers. [21] 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. Although GPU-accelerated branch-and-bound algorithm had been studied in [11] for knapsack, [13] for flow-shop scheduling, and [4] for multiproduct batch plants optimization sub-problems, there exists no prior work that uses GPU to accelerate the branch-and-bound search for differential cryptanalysis.

Our Contributions. The proposed work is a novel approach leveraging GPU hardware acceleration for the specific sub-problem of differential cluster search. It also incorporates the meet-in-the-middle (MITM) technique [5] to further improve its efficiency. The proposed algorithm can achieve a substantial speedup, up to a factor of approximately 5.95. A comparison based on cloud computing

[4] We use the term differential *cluster* interchangeably with differentials to ensure that there is a clear distinction between differentials and individual characteristics.

also indicates that the GPU-based algorithm can save costs by up to 85% as compared to its CPU-based counterpart in enumerating high number of branches.

To showcase the practicality and feasibility of the proposed GPU-accelerated algorithm, we investigate the differential clustering properties of the 128-bit block cipher, TRIFLE-BC [16] as a proof-of-concept. By applying the proposed GPU-accelerated automatic search for differential clusters, the computational time needed to construct differential clusters for a large number of rounds of 128-bit TRIFLE-BC was significantly shortened. This effectively allowed us to identify differentials with the highest probability to date, thus making this work one of the first successful attempts in implementing an automated differential search for a block cipher with 128-bit block size at a very large number of rounds (43 rounds). Previous automated search attempts have focused on block ciphers with block sizes of 64 bits or less [5,21]. For literature that involve 128-bit block ciphers, the number of rounds searched were noticeably lower (typically $<= 20$), and are only capable of identifying singular differential characteristics [2,7]. Although the framework proposed in [22] 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.

Outline. The rest of this paper is organized as follows: Section 2 introduces the GPU architecture and CUDA technology, followed by TRIFLE and its cryptanalytic results. Section 3 describes the conventional branch-and-bound differential search and its improved version that serves as the basis for this work. The GPU-accelerated algorithm is detailed in Section 4, the performance of which is compared with its CPU-counterpart. Capabilities and limitations of the proposed algorithm are also discussed. Section 5 investigates the differential cluster effect of TRIFLE-BC. Section 6 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.

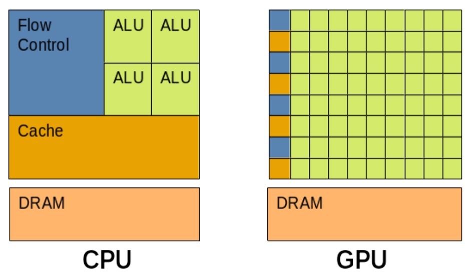

Fig. 1: Structural differences between CPU and GPU.

CUDA threads run on a separate physical **device** (GPU) to accelerate parallel tasks given by the co-running **host** program (CPU) as illustrated by Figure 2. 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,

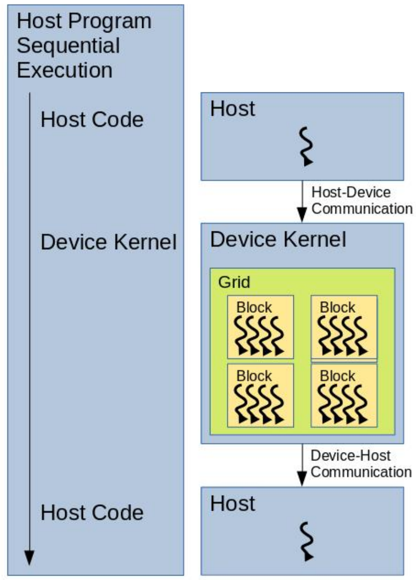

Fig. 2: Heterogeneous programming architecture of a typical GPU-accelerated algorithm. (Note that serial host code executes on the CPU while parallel device code executes on the GPU)

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 [20], 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. 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 and GPU. 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 [19].

2.2 TRIFLE

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.
- OZP(X) applies an optional 10^* padding on n bits. If $|X| < n$, then OZP(X) $= 0^{n-|X|-1}||1||X$. If $|X| = n$, then OZP(X) $= X$.
- $\lfloor X \rfloor$ is an integer floor function that produces an integer i closest to X such that $i \leq X$.
- $>>>$ 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.
- ΔX is an XOR difference, ΔU_i is the i^{th} nibble value inside ΔX, and ΔAU_i is the i^{th} active nibble value (non-zero difference) inside ΔX.

Description. TRIFLE is a round-1 candidate of the lightweight encryption standardization effort by NIST [17]. It is a 128-bit block cipher-based authenticated encryption scheme. 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. The underlying block cipher, TRIFLE-BC is a 50-round 128-bit SPN block cipher. Each round of TRIFLE-BC consists of four consecutive functions namely SubNibbles, BitPermutation, AddRoundKey, and AddRoundConstant. For a more detailed TRIFLE specification, refer to [16].

Differential properties of TRIFLE-BC. By analyzing the differential distribution table of TRIFLE's s-box, it was found that each ΔU that has a hamming weight of a single bit ($W_{bit} = 1$) can be differentially mapped back to ΔV 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 ΔV will be permuted and propagated to the next round to become yet another ΔU with $W_{bit} = 1$ due to the nature of bitwise 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 $\Delta X(X_0, X_1, ..., X_{31}) \rightarrow \Delta Y(Y_0, Y_1, ..., Y_{31})$ such that $W_{bit}(X_i^j) = 1$ where $0 \leq i < 32, 0 \leq j < n$ and $P(\Delta X \rightarrow \Delta Y) = 2^{-3n}$. 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 2^{-3n+1} for any n arbitrary rounds. Since there also exists a ΔV for every ΔU 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. Thus, the single-bit differential characteristics with improved first and final rounds that have a probability of 2^{-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. These observations have also been discussed in [8] and [18].

Based on the aforementioned improved single-bit differential characteristics, a key recovery strategy had been discussed in [8] 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 [8] should 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 [15], whereby multiple differential characteristics with the same $\Delta X \rightarrow \Delta Y$ are considered for the probability of a given differential.

3 Automatic search for differential

Matsui proposed a branch-and-bound algorithm [12] for searching linear paths and differential characteristics. The algorithm had been used on DES to find the best characteristic at the time. The algorithm relied on pruning *bad* branches that have lower probability than the best one 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 used the knowledge of $\overline{B_{n-i}}$ 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 $\overline{B_n}$.

Since then, several improvements have been made to Matsui's algorithm. A cluster search algorithm such as [5] improved upon Matsui's algorithm by searching for differential clusters after identifying a main differential characteristic. The differential cluster search includes all differential characteristics that share the same input ΔX and output ΔY differences but with different intermediary differences. This led to differentials with improved probability for block ciphers such as LBlock and TWINE [5]. It is also worth noting that [5] used the number of active s-boxes as part of the pruning rules to eliminate bad branches. There were also other researchers [3,6] that use a type of automatic search known as the threshold search for ARX ciphers.

A combination of the number of active s-boxes and the differential probability threshold will be used as the pruning rules for the proposed GPU-based automatic search. The combination of both allows for greater flexibility during the search, and also effectively filters branches quickly if configured correctly. This CPU-based recursive algorithm is described in Algorithm 1[5].

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

To facilitate the differential search for block ciphers with a large block size and number of rounds, the processing power of GPUs can be leveraged to provide a substantial performance boost to the conventional branch-and-bound searching algorithm. The proposed GPU-accelerated algorithm is a variant of a depth-first search whereby the algorithm will first visit nodes (possible branches) in successive rounds 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 3.

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

[5] All algorithms are described in Appendix 1.

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 has been verified by comparing the results of Algorithms 1 and 2.

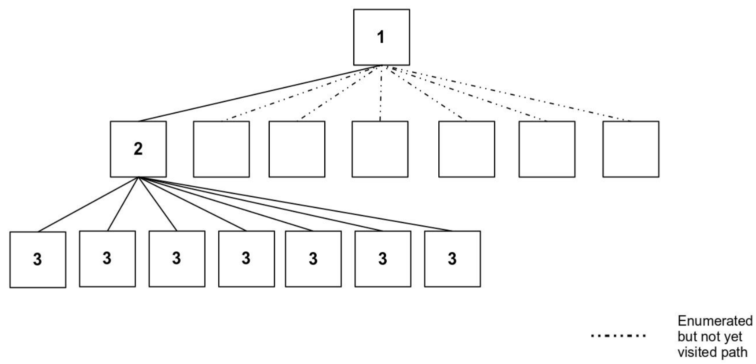

Fig. 3: The searching strategy for the proposed algorithm.

4.1 Enumeration using GPU Kernel and CPU

The GPU kernel has been optimized for TRIFLE's structure which has a constant branching number of 7 for ΔV. This means that $\forall \Delta U$ that goes through the TRIFLE's s-box, there are precisely 7 possible choices of ΔV. Despite this specific customization used, the kernel can 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 ΔX_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 ΔU 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 active ΔU branches, but the remaining active ΔU 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. 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 [5] 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 α rounds and β rounds instead of searching directly for $(\alpha + \beta)$ rounds.

The steps involved in the MITM approach starts off by dividing the search into forward α rounds and backward β rounds. For the forward search, the proposed algorithm mentioned in Algorithm 2 is used. The difference is that during the α^{th} (final) round, instead of evaluating ΔY_α, the ΔY_α 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 β^{th} (final) round, ΔY_β 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 4.

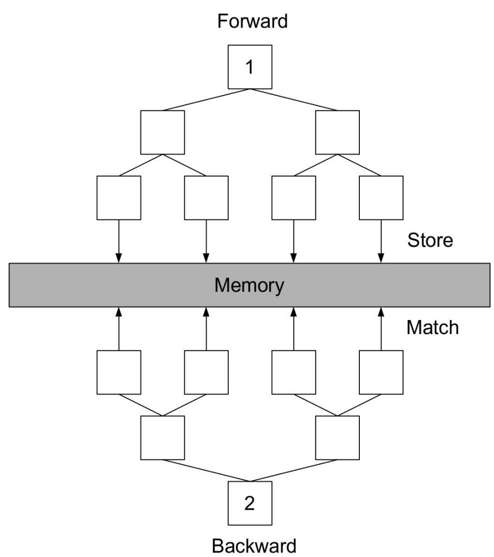

Fig. 4: Meet-in-the-middle approach.

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^{th} 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 $W_{nibble}(\Delta X)$ 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 1 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 2. A similar experiment is performed for a series of Google VM Cloud-based CPU and GPU. The performance results indicate that for $AS = 8$, the cost reduction is estimated to be around 16% to 85% of the original cost compared to the reference XEON CPU. These results depict the potential of the proposed algorithm in terms of cost-saving for large numbers of active s-boxes.

A series of practical tests of the proposed algorithm is performed on various rounds of TRIFLE-BC. The results are recorded in Table 4 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 58 at round 20.

Table 1: Search time (μs) comparison of CPU and GPU kernel enumeration.

$W_{nibble}(\Delta X)$	GPU-Accel	CPU-Enum	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

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 ΔX is ≥ 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.

Table 2: Search time (μs) comparison of CPU and GPU kernel enumeration (without output memory synchronization).

$W_{nibble}(\Delta X)$	GPU-Accel	CPU-Enum	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 3: Cloud computing cost (USD) comparison without memory synchronization for $\Delta X_i \rightarrow \Delta X_{i+1}$ for $W_{nibble}(\Delta X) = 8$.

Device	Time(μs)	Cost/Month	Core Equivalent	Cost%
Xeon Skylake 2.0 GHz*	506 703.0	27.46	1	100
Tesla T4	8531.6	255.50	60	16
Tesla P100	8080.4	1065.80	63	62
Tesla V100	6528.0	1810.40	78	85

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) for a large number of rounds (≥ 30). This has yet to be attempted in previous works. 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. In addition, the algorithm can be easily adapted to search for linear hulls.

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 de-

Table 4: Search time (ms) of various rounds of TRIFLE-BC.

Round(s)	MITM-GPU-Accel	GPU-Accel	CPU-Enum
5	1135.9	661.5	787.2
10	2197.3	8644.5	19 564.6
15	8795.6	62 928.7	156 725.0
20	15 675.2	363 274.2	908 978.1

Table 5: Differential for 20-round TRIFLE-BC

ΔX	ΔY	P_t	P_c	# of Trails
0000 0000 0000 0000 00b0 0000 0000 0000	0000 0000 0000 0000 0000 0001 0000 0001	2^{-58}	$2^{-57.97}$	50901814
0000 0000 0000 0000 0000 d000 0000 0000	0000 0100 0000 0100 0000 0000 0000 0000	2^{-58}	$2^{-57.97}$	39432495
0000 0000 0000 0000 0000 0000 0700 0000	0000 0000 0000 0000 0000 0002 0000 0002	2^{-58}	$2^{-57.97}$	51377914
0000 0000 0000 0000 0000 0b00 0000 0000	0000 0000 0000 0000 0000 0400 0000 0400	2^{-58}	$2^{-57.996}$	30372009

scribed in Subsection 2.2 are clustered using the proposed algorithm. The cluster search was conducted for 43-round TRIFLE-BC using $AS\_BOUND = 4$ and $PROB\_BOUND = P_t \times 2^{-21}$. A equivalent search is conducted for 20-round TRIFLE-BC using $AS\_BOUND = 4$ and $PROB\_BOUND = P_t \times 2^{-31}$. A slightly higher bound is used here in an attempt to cluster more differential trails. The time required to complete the search is two days using the desktop computer described in Subsection 4.3.

Since the differential probabilities are similar, we select only 4 differentials with 3 being the best probability and 1 differential being the differential described in [8] to show in Table 5 and Table 6. 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 51 and 0.5 million trails for 20-round and 43-round TRIFLE-BC respectively. The differential used in [8] 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 se-

Table 6: Differential for 43-round TRIFLE-BC

ΔX	ΔY	P_t	P_c	# of Trails
0000 0000 0000 b000 0000 0000 0000 0000	0000 0000 0010 0000 0010 0000 0000 0000	2^{-127}	$2^{-126.931}$	544352
0000 0000 0000 0000 b000 0000 0000 0000	0000 0002 0000 0002 0000 0000 0000 0000	2^{-127}	$2^{-126.931}$	564220
0000 0000 0000 0000 0007 0000 0000 0000	0020 0000 0020 0000 0000 0000 0000 0000	2^{-127}	$2^{-126.931}$	584356
0000 0000 0000 0000 0000 0b00 0000 0000	0000 0000 0000 0000 0000 0400 0000 0400	2^{-127}	$2^{-126.995}$	381035

curity 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. Rather than just a direct application of GPUs to the problem, we implicitly partitioned the difference branches into chunk sizes which corresponds to a individual thread in the GPU kernel. The implicit partitioning allows the thread to acquire its work unit in a fixed amount of step without thread divergence and synchronization mechanisms to maximize the GPU core utilization. The proposed algorithm can achieve a tremendous speedup especially when enumerating large amount of branches. The speedup enables the search for large differential clusters for block ciphers with a large block size over a large number of rounds. Aided by the proposed GPU framework, we provide a detailed look at the clustering effect of the authenticated cipher TRIFLE, which also served to showcase the practically of the proposed framework. We were able to construct large clusters consisting of hundreds of thousands to millions of individual differential characteristics, even for a large number of rounds of TRIFLE's underlying 128-bit block cipher. 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 for 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. The proposed approach can also be extended to utilize a grid of CPU-GPU computing nodes in a real-world environment for an even higher efficiency gains. In addition, it can be easily adapted to search for linear hulls. Last but not least, the GPU framework described in this paper can be used to provide a more accurate security bound on differential cryptanalysis for block ciphers.

Acknowledgement

This work is supported in part by the Ministry of Education Malaysia under the Fundamental Research Grant Scheme project no. FRGS/1/2019/ICT05/USM/02/1, the National Natural Science Foundation of China under grant no. 61702212, and the Fundamental Research Funds for the Central Universities under grant no. CCNU19TS017.

References

1. Banik, S., Pandey, S.K., Peyrin, T., Sasaki, Y., Sim, S.M., Todo, Y.: GIFT: A Small Present. In: Fischer, W., Homma, N. (eds.) Cryptographic Hardware and Embedded Systems – CHES 2017, vol. 10529, pp. 321–345. Springer International Publishing, Cham (2017). https://doi.org/10.1007/978-3-319-66787-4_16, http://link.springer.com/10.1007/978-3-319-66787-4_16
2. Biryukov, A., Nikolić, I.: Automatic Search for Related-Key Differential Characteristics in Byte-Oriented Block Ciphers: Application to AES, Camellia, Khazad and Others. In: Hutchison, D., Kanade, T., Kittler, J., Kleinberg, J.M., Mattern, F., Mitchell, J.C., Naor, M., Nierstrasz, O., Pandu Rangan, C., Steffen, B., Sudan, M., Terzopoulos, D., Tygar, D., Vardi, M.Y., Weikum, G., Gilbert, H. (eds.) Advances in Cryptology – EUROCRYPT 2010, vol. 6110, pp. 322–344. Springer Berlin Heidelberg, Berlin, Heidelberg (2010). https://doi.org/10.1007/978-3-642-13190-5_17, http://link.springer.com/10.1007/978-3-642-13190-5_17
3. Biryukov, A., Velichkov, V.: Automatic Search for Differential Trails in ARX Ciphers. In: Hutchison, D., Kanade, T., Kittler, J., Kleinberg, J.M., Mattern, F., Mitchell, J.C., Naor, M., Nierstrasz, O., Pandu Rangan, C., Steffen, B., Sudan, M., Terzopoulos, D., Tygar, D., Vardi, M.Y., Weikum, G., Benaloh, J. (eds.) Topics in Cryptology – CT-RSA 2014, vol. 8366, pp. 227–250. Springer International Publishing, Cham (2014). https://doi.org/10.1007/978-3-319-04852-9_12, http://link.springer.com/10.1007/978-3-319-04852-9_12
4. Borisenko, A., Haidl, M., Gorlatch, S.: A GPU parallelization of branch-and-bound for multiproduct batch plants optimization. The Journal of Supercomputing **73**(2), 639–651 (Feb 2017). https://doi.org/10.1007/s11227-016-1784-x, http://link.springer.com/10.1007/s11227-016-1784-x
5. Chen, J., Miyaji, A., Su, C., Teh, J.: Improved Differential Characteristic Searching Methods. In: 2015 IEEE 2nd International Conference on Cyber Security and Cloud Computing. pp. 500–508. IEEE, New York, NY, USA (Nov 2015). https://doi.org/10.1109/CSCloud.2015.42, http://ieeexplore.ieee.org/document/7371529/
6. Chen, K., Tang, X., Xu, P., Guo, M., Qiu, W., Gong, Z.: An Improved Automatic Search Method for Diffierential Trails in TEA Cipher. International Journal of Network Security **18**(4), 644–649 (Jul 2016). https://doi.org/10.6633/IJNS.201607.18(4).05
7. ElSheikh, M., Abdelkhalek, A., Youssef, A.M.: On MILP-Based Automatic Search for Differential Trails Through Modular Additions with Application to Bel-T. In: Buchmann, J., Nitaj, A., Rachidi, T. (eds.) Progress in Cryptology – AFRICACRYPT 2019, vol. 11627, pp. 273–296. Springer International Publishing, Cham (2019). https://doi.org/10.1007/978-3-030-23696-0_14, http://link.springer.com/10.1007/978-3-030-23696-0_14
8. Fukang, L., Takanori, I.: Iterative Differential Characteristic of TRIFLE-BC (2019), https://eprint.iacr.org/2019/727.pdf
9. Guo, J., Peyrin, T., Poschmann, A., Robshaw, M.: The LED Block Cipher. Cryptographic Hardware and Embedded Systems – CHES 2011 pp. 326–341 (2011)
10. Lai, X., Massey, J.L., Murphy, S.: Markov Ciphers and Differential Cryptanalysis. In: Davies, D.W. (ed.) Advances in Cryptology — EUROCRYPT '91, vol. 547, pp. 17–38. Springer Berlin Heidelberg, Berlin, Heidelberg (1991). https://doi.org/10.1007/3-540-46416-6_2, http://link.springer.com/10.1007/3-540-46416-6_2

11. Lalami, M.E., El-Baz, D.: GPU Implementation of the Branch and Bound Method for Knapsack Problems. In: 2012 IEEE 26th International Parallel and Distributed Processing Symposium Workshops & PhD Forum. pp. 1769–1777. IEEE, Shanghai, China (May 2012). https://doi.org/10.1109/IPDPSW.2012.219, http://ieeexplore.ieee.org/document/6270853/
12. Matsui, M.: On correlation between the order of S-boxes and the strength of DES. In: Goos, G., Hartmanis, J., van Leeuwen, J., De Santis, A. (eds.) Advances in Cryptology — EUROCRYPT'94, vol. 950, pp. 366–375. Springer Berlin Heidelberg, Berlin, Heidelberg (1995). https://doi.org/10.1007/BFb0053451, http://link.springer.com/10.1007/BFb0053451
13. Melab, N., Chakroun, I., Mezmaz, M., Tuyttens, D.: A GPU-accelerated Branch-and-Bound Algorithm for the Flow-Shop Scheduling Problem. In: 2012 IEEE International Conference on Cluster Computing. pp. 10–17. IEEE, Beijing, China (Sep 2012). https://doi.org/10.1109/CLUSTER.2012.18, http://ieeexplore.ieee.org/document/6337851/
14. Mouha, N., Wang, Q., Gu, D., Preneel, B.: Differential and Linear Cryptanalysis Using Mixed-Integer Linear Programming. In: Wu, C.K., Yung, M., Lin, D. (eds.) Information Security and Cryptology, vol. 7537, pp. 57–76. Springer Berlin Heidelberg, Berlin, Heidelberg (2012). https://doi.org/10.1007/978-3-642-34704-7_5, http://link.springer.com/10.1007/978-3-642-34704-7_5
15. Nicky, M., Bart, P.: Towards Finding Optimal Differential Characteristics for ARX: Application to Salsa20. Cryptology ePrint Archive, Report 2013/328 (2013), https://eprint.iacr.org/2013/328
16. Nilanjan, D., Ashrujit, G., Debdeep, M., Sikhar, P., Stjepan, P., Rajat, S.: TRIFLE (Mar 2019), https://csrc.nist.gov/CSRC/media/Projects/Lightweight-Cryptography/documents/round-1/spec-doc/trifle-spec.pdf
17. NIST: Lightweight Cryptography, Round-1 Candidates (Apr 2019), https://csrc.nist.gov/projects/lightweight-cryptography/round-1-candidates
18. NIST: Round 1 Lightweight Cryptography | Official Comments - TRIFLE (2019), https://csrc.nist.gov/CSRC/media/Projects/Lightweight-Cryptography/documents/round-1/official-comments/TRIFLE-official-comment.pdf
19. NVIDIA: CUDA C Programming Guide Version 9.0 (Oct 2019), https://docs.nvidia.com/cuda/cuda-c-programming-guide/
20. Padua, D. (ed.): Encyclopedia of Parallel Computing. Springer US, Boston, MA (2011). https://doi.org/10.1007/978-0-387-09766-4, http://link.springer.com/10.1007/978-0-387-09766-4
21. Siwei, S., Lei, H., Meiqin, W., Peng, W., Kexin, Q., Xiaoshuang, M., Danping, S., Ling, S., Kai, F.: 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)
22. Song, L., Huang, Z., Yang, Q.: Automatic Differential Analysis of ARX Block Ciphers with Application to SPECK and LEA. In: Liu, J.K., Steinfeld, R. (eds.) Information Security and Privacy, vol. 9723, pp. 379–394. Springer International Publishing, Cham (2016). https://doi.org/10.1007/978-3-319-40367-0_24, http://link.springer.com/10.1007/978-3-319-40367-0_24
23. Sun, S., Hu, L., Song, L., Xie, Y., Wang, P.: Automatic Security Evaluation of Block Ciphers with S-bP Structures Against Related-Key Differential Attacks. In: Lin, D., Xu, S., Yung, M. (eds.) Information Security and Cryptology, vol. 8567, pp. 39–51. Springer International Publishing, Cham (2014). https://doi.org/10.1007/978-3-319-12087-4_3, http://link.springer.com/10.1007/978-3-319-12087-4_3

Appendix 1 CPU and GPU-accelerated Algorithm for Differential Cluster Search

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

Input: Input difference ΔX and output difference ΔY.
Output: Probability P_c of $\Delta X \rightarrow \Delta Y$ cluster.
Adjustable Parameters:
1. AS_BOUND : Maximum of number of active sboxes for ΔY.
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 ΔY_i **do**
        $p_i \leftarrow \Pr(\Delta X_i, \Delta Y_i)$
        $AS_{i+1} \leftarrow W_{nibble}(\Delta Y_i)$
        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 $[p_1, ..., p_i, p_{i+1}, p_r] \geq PROB\_BOUND$ **then**
                call procedure CLUSTER_SEARCH_ROUND_$(i+1)$
            end if
        end if
    end for
end procedure

procedure CLUSTER_SEARCH_ROUND_n
    for each candidate ΔY_n **do**
        if $\Delta Y_n == \Delta Y$ **then**
            $p_n \leftarrow \Pr(\Delta X_n, \Delta Y_n)$
            $P_c \leftarrow P_c + [p_1, ..., p_n]$
        end if
    end for
end procedure

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

```
procedure CLUSTER_SEARCH
    call procedure CLUSTER_SEARCH_ROUND_0
    P_c ← (∑_{i=1}^{T_total} P_i) + P_h
end procedure

procedure CLUSTER_SEARCH_ROUND_i (1 ≤ i < n)
    AS_i ← W_nibble(ΔX_i)
    if AS_i > 3 then
        call procedure ENUMERATION_DEVICE_i
    else
        call procedure ENUMERATION_HOST_i
    end if
    for each computed ΔY_i^j do
        if (ΔY_condition)_i^j == 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
```

Algorithm 3 Host (CPU) enumeration and evaluation method.

```
procedure ENUMERATION_HOST_i (1 ≤ i ≤ n)
    for each candidate (ΔAV_1, ΔAV_2, ..., ΔAV_{AS_BOUND}) do
        if i ≠ n then
            (ΔY_condition)_i^{candidate_index} ← FALSE
            p_i ← Pr(ΔX_i, ΔY_i)
            AS_{i+1} ← W_nibble(ΔY_i)
            if AS_{i+1} ≤ AS_BOUND then
                p_{i+1} ← (P_AS)^{AS_{i+1}}
                p_r ← (P_AS)^{n−i−1}
                if [p_1, ..., p_i, p_{i+1}, p_r] ≥ PROB_BOUND then
                    (ΔY_condition)_i^{candidate_index} ← TRUE
                end if
            end if
        else if ΔY_i == ΔY then
            P_i ← P_i + p_i
        end if
    end for
end procedure
```

Algorithm 4 Device (GPU) enumeration and evaluation method.

Input: Input Difference ΔX.
Output: Enumerated branches, its evaluation result and probabilities P_i.
Adjustable Parameters:
1. AS_BOUND : Maximum of number of active s-boxes for ΔY.
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 ≤ i ≤ 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 ≤ i ≤ n)
    copy permutation table, sorted DDT (Descending Frequency), and branch size table
    to shared memory
    T_i ← (blockIdx.x × blockDim.x + threadIdx.x)
    //Work assignment
    Value ← T_i, Divide_Value ← 1
    for each active nibble values,ΔAU_i where 1 ≤ i ≤ 4  do
        I_i ← ⌊Value/Divide_Value⌋ mod NB_i
        ΔAV_i ← sorted DDT[ΔAU_i][I_i]
        update p_i
        Divide_Value ← Divide_Value × 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 (ΔAV_5, ΔAV_6, ..., ΔAV_{AS_BOUND}) do
        if i ≠ n then
            global_offset ← (∏_{j=1}^{AS_BOUND} NB_j × T_i + candidate_index)
            (ΔY_condition)_i^{global_offset} ← FALSE
            p_i ← Pr(ΔX_i, ΔY_i)
            AS_{i+1} ← W_nibble(ΔY_i)
            if AS_{i+1} ≤ AS_BOUND then
                p_{i+1} ← (P_AS)^{AS_{i+1}}
                p_r ← (P_AS)^{n−i−1}
                if [p_1, ..., p_i, p_{i+1}, p_r] ≥ PROB_BOUND then
                    (ΔY_condition)_i^{global_offset} ← TRUE
                end if
            end if
        else if ΔY_i == ΔY then
            P_i ← P_i + p_i
        end if
    end for
end procedure
```