DTU Library

Analysis of differential distribution of lightweight block cipher based on parallel processing on GPU

Chen, Zhanwen; Chen, Jiageng; Meng, Weizhi; Teh, Je Sen; Li, Pei; Ren, Bingqing

Published in:
Journal of Information Security and Applications

Link to article, DOI:
10.1016/j.jisa.2020.102565

Publication date:
2020

Document Version
Peer reviewed version

Analysis of differential distribution of lightweight block cipher based on parallel processing on GPU

Zhanwen Chen[a], Jiageng Chen[a,*], Weizhi Meng[b], Je Sen Teh[c], Pei Li[a], Bingqing Ren[d]

[a]School of Computer, Central China Normal University, China
[b]Department of Applied Mathematics and Computer Science, Technical University of Denmark, Denmark
[c]School of Computer Sciences, Universiti Sains Malaysia, Penang, Malaysia
[d]Central China Normal University Wollongong Joint Institute, China

Abstract

As the fast development of IoT technology, various security solutions have to be considered when the corresponding solutions are being deployed. Due to the lightweight nature of the IoT devices such as the RFID tags and so on, traditional encryption schemes such as AES which are relatively heavy in the sense of operations cannot be applied here. Lightweight block ciphers have since become a default standard when considering security protections on such lightweight IoT devices. Compared with the security analysis approaches by taking advantage of the differential or linear cryptanalysis, the security margin of the lightweight block ciphers can be further derived more accurately due to the small internal state. In this paper, we investigate the security margin of the lightweight block cipher structure especially the SPN design by taking advantage of the parallel computing power of modern GPU architecture. We show how to accelerate the computing of the statistical distinguisher, which is the crucial point for analyzing the security of the cipher design. Our proposed methods gain notable advantage against traditional CPU architecture in terms of time complexity and possess extensibility for other block ciphers.

Keywords: Differential cryptanalysis, Graphics processing units, Parallel

*Corresponding author
Email address: `chinkako@gmail.com` (Jiageng Chen)

cryptanalysis, SPN cipher, Security margin

1. Introduction and Previous Works

The security of block ciphers heavily depends on the work of cryptanalysis to gain confidence regarding their security margin as well as the corresponding efficiency. Currently there are two main approaches in cryptanalysis, namely, differential cryptanalysis [1] and linear cryptanalysis[2]. Differential cryptanalysis was proposed by Eli Biham and Adi Shamir back in 1991 while cryptanalyzing DES. It was applied to attack DES from the perspective of plaintext and ciphertext differences. The evolution of a differential in each round is crucial to the success of an attack on a cipher and to date, differential cryptanalysis is still regarded a powerful and universal method in attacking block ciphers. Variants of conventional differential cryptanalysis like [3] and [4] has also been used for a long time. For cipher designers, a well designed cipher should be able to resist differential attacks to provide a reference on how to construct or improve a secure cipher. Classic differential cryptanalysis is based on the differential characteristics that are derived by connecting single round differential paths to form a large round differential path. Differential characteristics are relatively easy to compute using various approaches such as the branch and bound algorithm proposed by Matsui in 1993 [5]. However, it cannot represent the true differential distribution but can only provide a rough bound on the security margin. Rather, the concept of a differential, which takes all intermediate paths into consideration, provides a more accurate measure of a cipher's security margin. It will help increase the differential probability, and better reflects the true differential distribution. On the other hand, it is even more difficult to compute the differential given a large block size (for example 32-bit or larger). Recently, [6] proposed the idea of taking advantage of multiple differential paths to further improve the differential distinguisher. This concept can be extended to the extreme case where given one input difference, we compute all output differences (2^{n-1} where n is the block size). This further increases the computational complexity for identifying differentials.

Differential characteristic. A differential characteristic for a single round can be represented by a pair (α, β) where α denotes the input difference and β denotes the output difference such that difference α leads to β (denoted by $\alpha \overset{\mathscr{R}}{\to} \beta$). Differential characteristics with high probability $P(\alpha \overset{\mathscr{R}}{\to} \beta)$ can be exploited in the statistical attack. For multiple rounds, characteristics of each round are concatenated to produce a specific differential path $\alpha \overset{\mathscr{R}}{\to} \delta_1 \overset{\mathscr{R}}{\to} \delta_2 \overset{\mathscr{R}}{\to} ... \overset{\mathscr{R}}{\to} \beta$. The probability of concatenating these characteristics is the product of the probabilities of each single-round characteristic. While performing traditional differential cryptanalysis, most researchers identify differential characteristics with high probability like Wang *et al* in [7] whereby certain differences appear again after several rounds with high probability (called iterative characteristics). These iterative characteristics are used to help reduce the workload of cryptanalysis. However iterative characteristics may not always be found and more importantly, a differential characteristic considers only one specific path from start to end, neglecting information from other paths that could potentially expose more severe weaknesses of a block cipher.

Differential. When analyzing the block cipher, attackers only know the plaintext and ciphertext differences, and nothing else. It means that for a 3 round block cipher, attackers require knowledge of differential: $\alpha \overset{\mathscr{R}}{\to} ? \overset{\mathscr{R}}{\to} ? \overset{\mathscr{R}}{\to} \beta$ where '?' denotes the unknown and irrelevant difference value of intermediate rounds. This is called a *differential* that contains all the characteristics with same input and output differences. Conventionally, an individual differential characteristic restricts analysis to a specific differential path with a fixed evolution procedure. Thus, we can overcome this disadvantage by clustering all the characteristics and the probability of $P(\alpha \overset{\mathscr{R}^n}{\to} \beta)$ can be calculated in the following way:

$$P(\alpha \to \beta) = \sum_{\delta_n} ... \sum_{\delta_2} \sum_{\delta_1} P(\alpha \to \delta_1 \to \delta_2 ... \to \delta_n \to \beta) \tag{1}$$

Such probability is also the theoretical probability of β's appearance for input difference α when collecting samples to recover keys. From the viewpoint of a cipher designer, preventing attacks based on individual characteristics do not guarantee sufficient security. Instead, more emphasis should be given to preventing attacks against differentials.

Full distribution. Based on the differential, we want to go even further to obtain a more precise result for cryptanalysis. Blondeau *et al* investigate the statistical accuracy of the multiple differential cryptanalysis in [6]. They use more than one differential to allow the attacker to extract more information from the samples because for an input difference there exists multiple possible output differences given a large number of rounds. Usually if we take advantage of all the output differentials for some fixed input, we call this a *full distribution* It is more effective because for each input difference it leads to a unique distribution which include all the differential information and helps an attacker analyse the cipher. By using statistical approaches such as the χ^2 or LLR test, we can get an improved effect on the distinguisher.

Although using the full distribution leads to a more powerful cryptanalysis approach, its computational cost increases rapidly. Therefore, distinguishers based on differential characteristics are still used as the main approach since it is usually within our computational capabilities, making it a practical solution as compared to the full differential distribution approach. To compute the full distribution in a practical manner, parallel cryptanalysis can be adopted. We have attempted to compute full distribution cryptanalysis on a computer with 128 logic CPU cores but it still takes much time. Therefore we leverage upon the GPU's parallel advantage to mitigate this problem. [8] has used the PlayStation 3 to solve ECDLPs due to the game console's graphic processing ability. [9], [10], [11], [12] have shown that GPU has been accepted as a new implementation platform to improve the performance for block ciphers. The results of [13] and [14] show that the GPU possesses a much powerful parallel performance than CPU and can largely fasten the speed of encryption algorithms. Cryptanalysis field also starts to explore the potential on the new platform like [15], [16] where theoretically existing attacks that are hard to implement in real-world in the past can eventually be achieved thanks to GPU. Parallel computing on GPUs has also gained wide adoption in various areas such as deeping learning, Bitcoin mining and so on. Compared with a CPU framework, GPU supports more parallel threads and is cheaper. Thus, we want to propose a common method that achieves full distribution differential cryptanalysis on GPU for the SPN (substitution-permutation network) Structure.

Our contribution. In this paper we take advantage of GPU parallel computing power to speed up the computation of differential distinguishers for SPN cipher. First, we introduce an algorithm to to calculate the full distribution of SPN ciphers by parallel computation. Based on the aforementioned algorithm, three upper layer methods are introduced to solve two problems: limited GPU memory space, and achieving efficient attacks on large sized SPN block ciphers. For our experiments, we chose PRESENT[17], an ultralight SPN cipher designed by Bogdanov *et al*, as the target cipher and reduce its block size to calculate its security margin based on full differential distribution. We then evaluate the performance of our cryptanalysis. We use a reduced block size because searching for the full distribution for the original PRESENT with 64-bit block size is still impractical even with the help of GPU power. Hence we reduce the block size to 8,12,16,20,24,28-bit so that the search space of the distribution is reduced. By studying the security margin of the reduced versions we can obtain evidence that helps us to predict the security margin of PRESENT. Based on experimental results, we found that the GPU approach leads to a significant advantage over regular CPU computation.

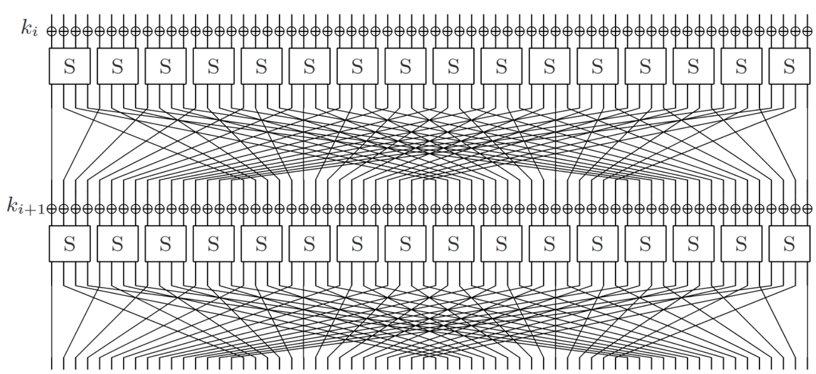
Outline. Section 2 introduces background information regarding differential cryptanalysis and GPU programming. Then in Section 3 we propose the general GPU-based algorithm for a full differential distribution search. Section 4 introduces three upper-layer methods based on the proposed base algorithm. They are used to solve the problem of inadequate memory space and improve computational efficiency. Performance analysis is given in Section 6 where the computational cost of GPU and CPU is compared. Finally we conclude the paper in Section 7.

2. Preliminaries

2.1. SPN Cipher and block size reduced PRESENT

Substitution-Permutation Network(SPN) is a common block cipher design strategy in order to achieve fast diffusion and confusion. It is widely used by many famous block ciphers such as AES, PRESENT and so on. And massive analysis results showed that this structure can indeed provide strong security margin given the primitives are flawless.

There are three operations in a basic encryption operation of SPN: **Key addition layer, substitution box (S-Box) layer, permutation layer.** In key addition layer, round key generator provides a round key based on the key update algorithm. Input of this round is XORed with the round key and go to the S-Box layer. S-Box is a substitution on the input data. The rule of substitution is defined when a specific SPN cipher is proposed. The input and output length is not required. Designers can create a S-Box of 4-bits length or 8-bits length or any other length. And how to map the input to output is also up to the cipher designed. S-Box is the only non-liner part that introduces the differential evolution.



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

Figure 1: PRESENT cipher

Figure 1 shows the structure of the original PRESENT cipher. The standard cipher PRESENT although being widely considered to be lightweight, the 64-bit block size is still too large for our experiment purpose. Luckily, PRESENT follows a very symmetric design structure and we can tweak the cipher by shrinking the block size without changing the cipher property. Actually Toy Present cipher has already been proposed for this purpose [18]. During the experiment we reduce the block size to 8, 12, 16, 20 and 24-bits for our experiment purpose.

2.2. Full distribution and security margin

For a block cipher with block size b and r round, differential(plaintext) space is $N = 2^b$. We denote all differences as $\delta_1, \delta_2, ..., \delta_N$. Full distribution means while doing differential cryptanalysis, we need to calculate N differentials after r round: (assume input difference is δ_2)

$$\begin{cases} \delta_2 \to \delta_1 \\ \delta_2 \to \delta_2 \\ \quad\quad ... \\ \delta_2 \to \delta_N \end{cases} \tag{2}$$

After the calculation we acquire an two dimension array of length N and elements in it is $(\delta_i, P(\delta_i))$, which represents the full distribution. Then a statistical test inspired by [19] is applied to calculate the data complexity of distribution. A statistical test is used to create distinguisher to distinguish two distributions D_0 and D_1, where D_0 is the full distribution we get from the experiment and D_1 is the uniform distribution. Then we calculate data complexity n as follows:

$$n = \frac{d}{\sum_{z \in Z} \frac{\epsilon_z^2}{p_z}} \approx \frac{d}{2D(D_0 \| D_1)} \tag{3}$$

The error probability for distinguisher is $P_e \approx \Phi(-\sqrt{d}/2)$. In our test, we set P_e to be 0.1. D is the Kullback-Leibler distance, which is calculated by:

$$D(D_0 \| D_1) = \sum_{z \in Z} Pr_{D_0}[z] log \frac{Pr_{D_0}[z]}{Pr_{D_1}[z]} \tag{4}$$

Data complexity n means how many samples the distinguisher needs to distinguish between the two distributions. Intuitively, n increase with round number r. Security margin is defined as followed: for a cipher with block size b and r, if $n > 2^b$ then the test needs more samples than the whole sample space. In other words, we cannot distinguish a distribution between the cipher and a theoretical uniform distribution. The smallest value of r that provides an indistinguishable case can be used to evaluate the security margin.

2.3. GPU feature

Running Program on GPU. In 2007 Nvidia release CUDA, a parallel computing platform and application programming interface (API) model, to enable programmers use their GPU and do general purpose process. [20] suggests that to run a program on GPU, first we need to create a kernel function that tells GPU how to deal with input data. Kernel function is run by GPU and use resources (memory and processors) inside GPU. GPU has its own memory space. If kernel function needs some input data, they can only be taken from GPU's memory. And if the result of kernel function needs to be recorded, kernel function can only store them in GPU's memory. There only exists a pipe that can copy memory between host (computer) and device (GPU). So all the input data is copied from host to device before calling kernel function and results that are stored in GPU memory is copied from device to host after all kernel function finish.

Thread Organization in GPU. The organization of threads can be defined by programmer. Kernel function create a grid, which contains all the threads, and runs inside this grid. Those threads share a global memory inside this grid. Threads are divided into several blocks as shown in figure 2. The number of threads and blocks can be customized according to the programmer. Streaming Multiprocessors(SM) inside GPU is in charge of dispatch threads. There are more than one SM in a GPU. SM can be seen like the core in CPU. While running the program, one block can only be dispatched by one SM. And SM dispatch thread in the unit of warp and one warp contains 32 threads.

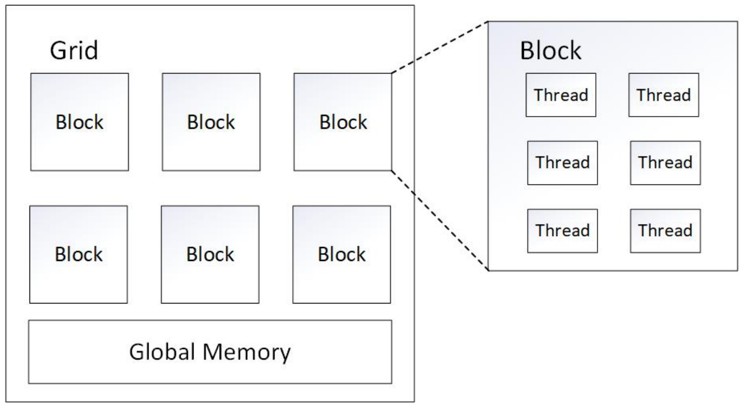


Figure 2: Organization of threads in GPU

3. Using GPU to calculate full distribution

Nvidia's GPU use SIMD (Single Instruction Multiple Data) to improve efficiency. For different data, GPU apply same instructions to them. We exploit this feature by arranging kernel function in this way: kernel function takes only one input difference and calculate the full distribution derived from this difference. Hence every thread is in charge of searching full distribution for one input difference.

3.1. Search the full distribution of one input difference after one round

Process of differential cryptanalysis can be seen as two main operation:

Difference combination. Differential distribution table of S-Box can be built from the substitution rule. Each S-Box provides a set $S_i = s_{i,1}, s_{i,2}, s_{i,3}...$ which contains all the possible output difference from it. Then to decide the difference of whole block, chose one possible difference of each S-Box for one time and combine them as:

$$Diff = s_{1,a}||s_{2,b}||s_{3,c}||... \tag{5}$$

All the possible combination should be recorded so it is a full combination. Each $s_{i,x}$ is generated with a probability $P(s_{i,x})$ hence the probability of combined difference can be calculated as:

$$P(\mathit{Diff}) = \prod P(s_{i,x}) \tag{6}$$

Permutation. Like plaintext, differential after S-Box layer can also be permuted. The same permutation rule is applied to the distribution obtained from *difference combination* and the differential distribution after this round can be get.

Table 1: SP table of the first S-Box in 8-bit version PRESENT

	0x00	0x02	0x08	0x0a	0x20	0x22	0x28	0x2a	0x80	0x82	0x88	0x8a	0xa0	0xa2	0xa8	0xaa
0	16	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	0	4	0	0	0	4	0	4	0	0	0	4	0	0
2	0	0	0	2	0	4	2	0	0	0	2	0	2	2	2	0
3	0	2	0	2	2	0	4	2	0	0	2	2	0	0	0	0
4	0	0	0	0	0	4	2	2	0	2	2	0	2	0	2	0
5	0	2	0	0	2	0	0	0	0	2	2	2	4	2	0	0
6	0	0	2	0	0	0	2	0	2	0	0	4	2	0	0	4
7	0	4	2	0	0	0	2	0	2	0	0	0	2	0	0	4
8	0	0	0	2	0	0	0	2	0	2	0	4	0	2	0	4
9	0	0	2	0	4	0	2	0	2	0	0	0	2	0	4	0
a	0	0	2	2	0	4	0	0	2	0	2	0	0	2	2	0
b	0	2	0	0	2	0	0	0	4	2	2	2	0	2	0	0
c	0	0	2	0	0	4	0	2	2	2	2	0	0	0	2	0
d	0	2	4	2	2	0	0	2	0	0	2	2	0	0	0	0
e	0	0	2	2	0	0	2	2	2	2	0	0	2	2	0	0
f	0	4	0	0	4	0	0	0	0	0	0	0	0	0	4	4

Notice that *permutation* is a one to one map, so it is possible to combine two process above into one step. A SP(substitution permutation) table is created to do previous two steps in one time. Based on the original differential distribution table of S-Box, we pre-calculate the result of permutation for all output differences. Table 1 gives an example of the first S-Box's (in order from left) SP table. In such way we remove the time cost of bit-wise operation *permutation* and increase only a little memory cost (each S-Box has its unique SP table rather than share one differential distribution table).

While searching the full distribution on GPU, each thread is in charge of only one input difference and search for all output differences after a round. Probability for output δ_x is calculated by:

$$P(\delta_x) = \sum_i (P(\delta_i) \times P(\delta_i \rightarrow \delta_x)) \tag{7}$$

Algorithm 1 shows the detailed process in searching full distribution for 1 round, which is also the kernel function.

Algorithm 1 Search full distribution for one input difference after one round

---

Input: Input difference *Diff*; Probability of the input difference P_{in}; Block size l.

Output: Array A as differential distribution of ciphertext.

$A[2^l] = [0..]$
$x_1, x_2, ... \leftarrow$ *Diff* //separate input for every S-Box
for all y_1 in $SP_1[x_1]$ **do**
    for all y_2 in $SP_2[x_2]$ **do**
        ...
        for all y_n in $SP_n[x_n]$ **do**
            $A[y_1 \oplus ... \oplus y_n] \leftarrow P_{in} * SP_1[x_1][y_1] * SP_2[x_2][y_2] * ...$
        end for
        ...
    end for
end for

For more rounds, the algorithm can be executed many times. P_{in} can be obtained from the result of previous round. Finally a full distribution that indicates probability of every differential can be obtained.

3.2. Memory and Thread Organization for Search full distribution

3.2.1. Memory Organization

Memory is the biggest limitation on GPU. 16 GB is already a large memory space for current GPU but is not adequate for large block size cipher's crypt-analysis. Therefore how to make good use of memory space is a key point. In full distribution search algorithm, every S-Box has its unique SP table. Kernel function use these tables to decide the output differences and its probability. They are copied to the *shared memory* of GPU, which is the fastest memory space but it is much smaller compared to the global memory.

To analyze as in Section 3.1, two arrays A_1, A_2 with length 2^b are needed to record full distribution. They are located in the global memory. One is for

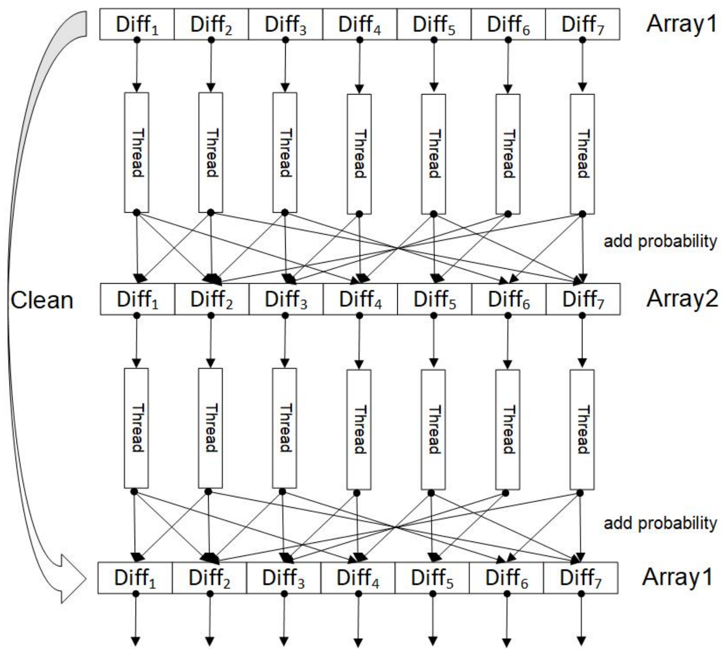


Figure 3: Reusing two arrays to record full distribution

input differential distribution and another is for output differential distribution. After every round the roles are changed. For example A_1 is the input of round 1 and output are stored in A_2. A_1 is cleaned after round 1 and then in round 2 A_2 becomes the input and outputs are written to A_1. These two arrays are recycled in turn to save memory space.

For every thread, it runs a kernel function and searches the full distribution for its input difference. All the threads share the same A_1, A_2 during the cryptanalysis. For each search result D and its probability P, P is added to $A_1[D]$ (or $A_2[D]$). Unavoidably writing conflict may happen when multiple threads want to add probability on same place of array. This is solved by CUDA's built-in function *atomicAdd* that every addition operation cannot be broken so that write conflict is avoided.

3.2.2. Thread Organization

CUDA allows programmers to create any amount of threads as long as it do not exceed the maximum number of thread, which is relatively large when used in cryptanalysis. So we do not consider the case where the number of threads is not enough. It can be seen in Figure 3 that every thread calculate only one input difference. For a cipher with block size b, 2^b threads are needed for our cryptanalysis task. After the thread amount is decided, block and grid still can be constructed in different ways. Due to the feature of GPU introduced in section 2.3, we give two principle for thread organization:

How many threads in block: SM process threads in the unit of wrap (32 threads). Therefore the thread number of block should always be multiple of 32.

How many blocks in grid: As for block number, it depends on the number of SM. One block can only be dispatched by one SM. Different SM works parallel. So the best condition is to make the block number no less than SM number.

4. Improved measures on the proposed search algorithm

4.1. Pruning for large block size

To shorten the time cost for large block size cipher, we propose a method that prune differential path with little probability in every round. Two arrays A_1 and A_2 only record parts of differentials with high probability rather than full distribution. Elements of A_1 and A_2 are recorded in form of {difference, probability}. Every time when a new output differential is found by threads, it is checked that if such differential's record has already been written. If it is then add the probability to existing record otherwise create a record in empty address. When there is no more space to store more records, a threshold probability is set for the current and further rounds that only the differential with higher probability than the threshold can be written to the array.

We set the threshold value to a theoretical value that every differential path has the same probability, which represents the uniform distribution. For a cipher with block size b, the uniform differential probability is $1/2^b$. Assume the cipher also has ideal permutation on characteristics, then probability of any characteristic is even and uniform characteristic probability is $1/2^b \times 1/2^b =$

$1/2^{2b}$. Such probability is chosen as threshold value. And the result of this cryptanalysis method is a semi-full distribution. Distinguisher from section 2.2 can still work on such distribution.

4.2. Meet in the middle approach by branch and bound

Meet in the middle attack is an efficient way to reduce time cost for differential cryptanalysis because in early rounds most input differences' probability is 0 so searching processes finishes fast. It takes several rounds before the input probability spread to the other part of the differential space. If cryptanalysis starts from both plaintext and ciphertext, then we can make use of early rounds twice.

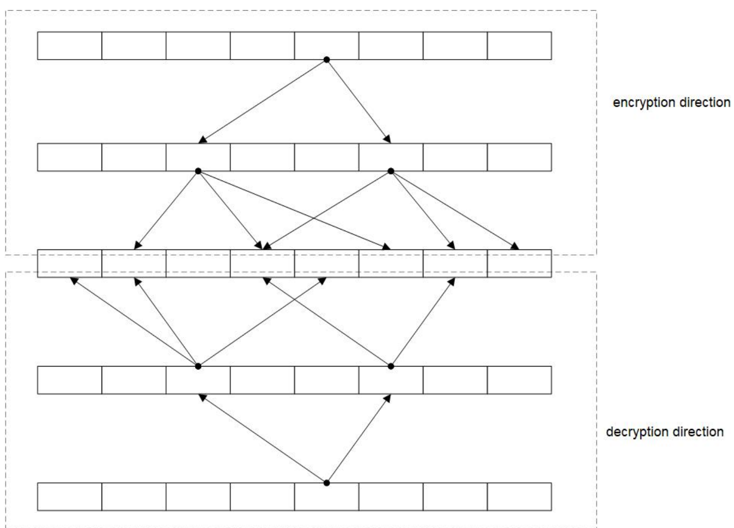


Figure 4: Meet in the middle attack

Supposing all threads are parallel worked, such meet in the middle improves nearly nothing. But things are different if there are thread blocking. While created threads' amount is more than what all SM can process, a waiting queue is produced and some threads are delayed, which largely decrease the parallel efficiency. However in early rounds when most differentials have 0 probability and the full distribution searching is not required, kernel function ends up quickly and spare SM to other threads. In addition, while recording probabilities to the output array in global memory, writing conflict is unavoidable but less writing request can reduce the chance of writing conflict.

One problem is that meet in the middle attack requires to decide which differences pair is chosen before begin the cryptanalysis. A *branch and bound* algorithm is introduced to help us predict which differential may have a high probability. Matsui first gives a branch and bound searching algorithm in citematsui1994correlation. The purpose of it is to quickly find a characteristic with very high probability among all the characteristics. It do not guarantee a high differential probability but research [21] by Chen *et al* shows that Matsui's algorithm also gives a high probability on differential. Algorithm 2 is the combination of Matsui's algorithm and meet in the middle attack, in which Matsui's algorithm is used to predict what plaintext and ciphertext difference (with potential high probability differential) is chosen. Then based on its result we use meet in the middle attack to calculate the differential probability. *branch and bound* returns the output very fast and is a recursion algorithm. Therefore we run this algorithm on CPU rather than GPU.

Algorithm 2 Use branch and bound to search for differential probability

Input: Total round number of cipher R; Block size b.

Output: Plaintext and ciphertext difference α,β; Differential probability P

$\alpha, \beta \leftarrow$ Branch&bound() //Done by CPU
$A_0[2^b], A_1[2^b], B_0[2^b], B_1[2^b] = [0..]$
$A_0[\alpha] = 1$; $B_0[\beta] = 1$
$P \leftarrow 0$
for $i \leftarrow 0$;$i < \frac{R}{2}$;$i++$ **do**
    $A_1 = full\_distribution\_search(A_0, b)$ //Encryption direction
    Swap(A_0,A_1)
    $A_1 = [0..]$
end for
for $i \leftarrow 0$;$i < \frac{R}{2} + 1$;$i++$ **do** //Decryption direction
    $B_1 = full\_distribution\_search(B_0, b)$ //Decryption direction
    Swap(B_0,B_1)
    $B_1 = [0..]$
end for
for $i \leftarrow 0$;$i < 2^b$;$i++$ **do**
    if $A_0[i]! = 0$ and $B_0[i]! = 0$ **then**
        $P+ = x * y$
    end if
end for

4.3. Matrix based differential cryptanalysis

Differential cryptanalysis on full distribution can be seen as matrix multiplication as well. Assume a full distribution $(\delta_1, \delta_2, \delta_3...)$ with probabilities $(p_1, p_2, p_3...)$, full distribution of next the round can be calculated in the following way:

$$p(\delta_i) = \sum_j p_j \times p(\delta_j \rightarrow \delta_i) \tag{8}$$

while $p(\delta_j \rightarrow \delta_i)$ is the differential characteristic of one round, and it is always a fixed value. Therefore the only parameter that changes in previous equation is

p_j. If all the information of $p(\delta_j \rightarrow \delta_i)$ can be calculated in advance, the process of searching full distribution can be represented by a matrix multiplication:

$$\begin{aligned}
&\begin{bmatrix} p_{1,r+1} & p_{2,r+1} & \cdots & p_{n,r+1} \end{bmatrix} \\
&= \begin{bmatrix} p_{1,r} & p_{2,r} & \cdots & p_{n,r} \end{bmatrix} \times \\
&\begin{bmatrix} \delta_1 \rightarrow \delta_1 & \delta_1 \rightarrow \delta_2 & \cdots & \delta_1 \rightarrow \delta_n \\ \delta_2 \rightarrow \delta_1 & \delta_2 \rightarrow \delta_2 & \cdots & \delta_2 \rightarrow \delta_n \\ \vdots & \vdots & \ddots & \vdots \\ \delta_n \rightarrow \delta_1 & \delta_n \rightarrow \delta_2 & \cdots & \delta_n \rightarrow \delta_n \end{bmatrix}
\end{aligned} \tag{9}$$

$p_{i,r}$ is the probability of difference δ_i in round r. $\delta_i \rightarrow \delta_j$ denotes the probability of differential characteristic from δ_i to δ_j in one round. Let equation 9 be written as:

$$P_{r+1} = P_r \times \Delta \tag{10}$$

We can perform the calculation iteratively to obtain the full distribution of round $k+i$ from round i, which is described by the the following relation.

$$P_{k+i} = P_i \times \Delta^k \tag{11}$$

It is obvious that two ways can be used to calculate equation 11. One is to normally start from $P_r \times \Delta$ and multiplied Δ one by one (left to right order). The second one is changing to a new calculation order that Δ^k is calculated first and then multiplied by P_i. One big advantage of using the latter way is that this fasten the process of searching the full distribution compared to the original method used in section 3, because it does not need to calculate round by round. After Δ^2 is calculated, Δ^k can be transformed to $(\Delta^2)^{\frac{k}{2}}$, $(\Delta^4)^{\frac{k}{4}}$ and so on. As a result, ideally the time of calculation can be reduced to $log_2 k$. Thus the original method in section 3 can be used to obtain Δ first and then used the matrix way for further rounds.

After transforming differential cryptanalysis into pure matrix multiplication, it is more clear to see the problem in a mathematical way: how to fasten large matrix multiplication on GPU. Section 3 provides a way of searching full distribution in differential view but many characteristics or techniques of GPU cannot be applied directly. As a result efficiency is not always satisfying. However for matrix multiplication, it has been studied for many years and a large number of matrix multiplication solutions are available. Taking advantage of those mature solutions makes the most use of GPU's parallel ability. MatrixMulCUBLAS, a highly recommended algorithm by Nvidia that depends on CUBLAS (CUDA Basic Linear Algebra Subroutines) library, is chosen to conduct matrix multiplication in our experiments because it costs little time when matrix has a large size.

In some cases matrix based method may face the memory limitation and Δ is too large for the GPU's memory, which is pretty common considering that nowadays GPUs do not have memory space as large as RAM or hard disk. Despite that the memory space of GPUs cannot be increased at will, matrix approach provides some characteristics that can help with this problem.

Sparse matrix. Δ contains the 1-round differential characteristic information for all input differential. We did a test on 16-bit PRESENT that a number of input differentials are chosen and their 1-round full distributions are computed independently. It turns out that about 99% output differentials have 0 probability. In other words Δ is a very large but sparse matrix with about 99% elements being 0. Therefore storage format for sparse matrix like COO, CSR, CSC save a lots of memory space need, and corresponding calculation approaches are still feasible for parallel operation.

Matrix partition. Although Δ can be compressed as sparse matrix, after several times of multiplication Δ^r will become a dense matrix and the compressing method loses its effectiveness. Under this circumstance, Δ^r can only be stored in RAM or hard disk. But matrix partition enables us to split a large matrix into some sub-matrices and those sub-matrices follow the calculation rule as the elements in matrix. Assuming that the GPU memory can only store two $n \times n$ matrices, Δ^r can be split to an assemble of $n \times n$ matrices. For each time two sub-matrices are transferred to GPU and the result is written back to RAM or hard disk.

5. Security margin of tested cipher

We use Tesla V100-PCIE-16GB to search the full distribution of several block size reduced PRESENT cipher. Then the statistical methodology is applied based on the full distribution result to calculate the data complexity, which indicates the security margin of the tested cipher. For each version, we choose plaintext difference in such principle:

a. There is only one active S-Box in the first round.

b. Value of input difference in the active S-Box is chosen randomly

We search the full distribution up to 8 rounds and detailed information about log_2C (Denote data complexity as C) is shown in table 2. Logarithm is applied to make it more clearly to see whether distinguisher can successfully distinguish the two distributions while the value is directly compared with block size. Bold number represents the least round number when distinguisher cannot distinguish practical and theoretical distributions. The security margin of round number can be further derived from table 3.

Table 2: log_2C of tested cipher

Version	Input Diff	round 1	round 2	round 3	round 4	round 5	round 6	round 7	round 8
8-bits	0x7	-0.213	1.064	3.762	7.972	**12.120**	15.469	19.762	23.992
8-bits	0x80	-0.213	1.702	5.756	**9.880**	13.856	17.541	22.209	25.872
12-bits	0x8	-1.003	0.362	3.404	7.017	10.402	**14.986**	19.441	24.415
12-bits	0x50	-0.964	0.079	3.335	7.765	**12.807**	17.264	22.175	27.204
12-bits	0x400	-0.964	0.028	2.902	6.586	11.519	**16.587**	21.621	26.255
16-bits	0xc	-1.483	-0.964	0.971	5.114	8.0956	12.263	**17.201**	21.634
16-bits	0x40	-1.483	-0.575	1.841	6.116	10.782	15.026	**19.181**	23.229
16-bits	0x200	-1.483	-0.575	1.79	6.170	10.960	15.066	**19.342**	23.552
16-bits	0x7000	-1.510	-0.842	0.938	5.173	9.031	12.227	**16.721**	22.141
20-bits	0x3	-1.863	-1.362	0.121	3.617	8.911	14.786	**20.607**	26.562
20-bits	0x40	-1.863	-1.294	0.817	5.975	11.941	18.139	**23.985**	29.904
20-bits	0xb00	-1.863	-1.420	-0.174	2.773	7.609	13.076	18.679	**24.376**
20-bits	0x9000	-1.884	-1.462	-0.337	2.603	7.582	13.166	18.926	**24.863**
20-bits	0xd0000	-1.863	-1.441	-0.221	2.727	7.710	13.402	18.919	**24.790**

Security margin of those block size reduced PRESENT versions are obtained as follow:

Table 3: The maximum number of rounds for an effective distinguisher.

8-bits	12-bits	16-bits	20-bits
5th round	6th round	7th round	8th round

We can observe that the data complexity increases with the round number. Before the round where data complexity's logarithm (log_2C) is less than 0, it grows slowly. While $log_2C > 0$, the increment suddenly changes to around 4. Besides, increment between neighboring rounds is nearly the same (4 for each neighboring round), which indicates that the growth of the data complexity can be exponential. So we predict that the security margin of the original PRESENT is around round 19 when $log_2C \geq 64$. Another factor that affects the data complexity is the plaintext's difference. For a certain version, various differentials may derive different security margins and the largest one is often chosen to evaluate the final security margin.

6. Performance analysis

In previous work we use a 128 core CPU to do same full distribution search on block size reduced PRESENT. Time cost for GPU and CPU to find full distribution up to 8 rounds for randomly chosen plaintext difference is shown below:

Table 4: Time cost of using CPU and GPU

	CPU	GPU
8-bits	2s	4.6ms
12-bits	10s	4.7ms
16-bits	30min	14.5ms
20-bits	12h	5.6s
24-bits	N/A	3.8min
28-bits	N/A	7.6h

It can be easily seen from the table 4 that using GPU increase the speed greatly. GPU can help finish all the work but CPU takes a few days and still cannot derive the results of the 24-bit version. Considering the ecomonical cost of a 128 core computer is even more than a TESLA graphic card, using GPU to do cryptanalysis has more advantages over the CPU structure. It also can be seen that time cost grows largely from 24-bits version to 28-bits version for GPU. It shows that after 24-bit version thread amount engages a bottleneck of GPU's maximum parallel thread number.

For improved measures, time cost of pruning method in Section 4.1 depends on the array length which can be referred to Table 4. Meet-in-the-middle attack costs double time of half rounds and same block size for corresponding ciphers in Table 4. Table 5 shows the time comparison of matrix method and the original method. While using matrix multiplication, 12-bit version is the limit without the matrix partition. For small block size version such as 8-bit and 12-bit, **Matrix 1** shows the dominant advantage over **Original**. **Matrix 2** requires more parallel threads because it contains multiplication of larger matrix (Δ^n). Thus it only excel **Original** in 8-bit version when block size is too small that **Original** cannot make use of all the parallel capacities. After matrix partition is involved from 16-bit, although the problem of inadequate memory is solved, partition process makes two matrix methods slower than **Original**. Due to that **Original, Matrix 1**'s time cost grows linearly along with round number when **Matrix 2**'s grows logarithmically, the former is more effective when the round number is not too large. And given enough memory space, **Matrix 1** is relatively the faster way for differential cryptanalysis. We expect it to be powerful in the further when GPU memory no longer being the bottleneck.

The matrix partition is done by the following ways: For **Matrix 1**, $P_r \times \Delta$ is partitioned to be $\begin{bmatrix} P_r \end{bmatrix} \times \begin{bmatrix} \delta_1 & \delta_2 & \cdots & \delta_n \end{bmatrix}$ where P_r is a $2^b \times 1$ sub-matrix and δ_i being $j \times 2^b$. j is adaptive that is expected to make the most use of memory space in GPU. As **Matrix 2**, it is always a multiplication of two $n \times n$ matrix and Tesla V100 can hold $2^{12} \times 2^{12}$ matrix multiplication at most. So for larger matrix, they are all partitioned to be a square matrix with elements

Table 5: Time cost of matrix multiplication and original method for calculating 8 rounds full distribution

	Original	Matrix 1	Matrix 2
8-bits	4.6ms	0.08ms	0.08ms
12-bits	4.7ms	0.96ms	88ms
16-bits	14.5ms	156.3ms	6min
20-bits	5.6s	52s	N/A

Matrix 1: Calculate in left to right order
Matrix 2: Calculate Δ^8 first

being $2^{12} \times 2^{12}$.

7. Conclusion

In this paper, we studied how to derive multiple differentials by taking advatange of the parallel computing, and specifically, we choose Tesla V100 as our experiment platform. According to our performance test, it shows that GPU can indeed largely speed up the procedure of cryptanalysis compared with CPU platform. Based on the full distribution searching algorithm, we provide some improvements to solve some limitations introduced by the GPU structure. Facing the inadequate memory space of GPU, we save memory cost by abandoning low probability in every round. And if the thread blocking occurs seriously in GPU, a meet in the middle attack method with guidance of branch and bound algorithm can help to increase the efficiency. Furthermore, differential cryptanalysis is transformed into matrix multiplication when mature algorithms of such area are adopted to both solving memory and efficiency problems. Besides, although only differential cryptanalysis is included, the process of linear cryptanalysis is similar to differential cryptanalysis, which means by applying the some transformations the proposed method can be used in linear cryptanalysis as well. Although at present full distribution differential cryptanalysis can only be achieved on block size reduced cipher, with the development of industries more powerful GPU can be used to analyse ciphers with larger block size in the future.

8. Acknowledgement

The final authenticated version of this preprint has been published in the Journal of Information Security and Applications at https://doi.org/10.1016/j.jisa.2020.102565.

References

[1] E. Biham, A. Shamir, Differential cryptanalysis of des-like cryptosystems, in: Advances in Cryptology - CRYPTO '90, 10th Annual International Cryptology Conference, Santa Barbara, California, USA, August 11-15, 1990, Proceedings, 1990, pp. 2–21. doi:10.1007/3-540-38424-3_1.

[2] M. Matsui, A. Yamagishi, A new method for known plaintext attack of feal cipher, in: Workshop on the Theory and Application of of Cryptographic Techniques, Springer, 1992, pp. 81–91.

[3] L. R. Knudsen, Truncated and higher order differentials, in: International Workshop on Fast Software Encryption, Springer, 1994, pp. 196–211.

[4] D. Wagner, The boomerang attack, Fast Software Encryption March 1636 (1998) 245–259.

[5] M. Matsui, On correlation between the order of s-boxes and the strength of DES, in: Advances in Cryptology - EUROCRYPT '94, Workshop on the Theory and Application of Cryptographic Techniques, Perugia, Italy, May 9-12, 1994, Proceedings, 1994, pp. 366–375. doi:10.1007/BFb0053451.
URL https://doi.org/10.1007/BFb0053451

[6] C. Blondeau, B. Gérard, K. Nyberg, Multiple differential cryptanalysis using LLR and χ 2 statistics, in: Security and Cryptography for Networks - 8th International Conference, SCN 2012, Amalfi, Italy, September 5-7, 2012. Proceedings, 2012, pp. 343–360. doi:10.1007/978-3-642-32928-9_19.
URL https://doi.org/10.1007/978-3-642-32928-9_19

[7] M. Wang, Differential cryptanalysis of reduced-round present, in: International Conference on Cryptology in Africa, Springer, 2008, pp. 40–49.

[8] J. Bos, M. Kaihara, T. Kleinjung, A. Lenstra, P. Montgomery, Solving a 112-bit prime elliptic curve discrete logarithm problem on game consoles using sloppy reduction. ijact 2 (3), 212–228 (2012) (2012).

[9] O. Hajihassani, S. K. Monfared, S. H. Khasteh, S. Gorgin, Fast aes implementation: A high-throughput bitsliced approach, IEEE Transactions on Parallel and Distributed Systems.

[10] A. A. Abdelrahman, M. M. Fouad, H. Dahshan, A. M. Mousa, High performance cuda aes implementation: A quantitative performance analysis approach, in: 2017 Computing Conference, IEEE, 2017, pp. 1077–1085.

[11] A. Saxena, V. Agrawal, R. Chakrabarty, S. Singh, J. S. Banu, Accelerating image encryption with aes using gpu: A quantitative analysis, in: International Conference on Intelligent Systems Design and Applications, Springer, 2018, pp. 372–380.

[12] A. A. Abdelrahman, H. Dahshan, G. I. Salama, Enhancing the actual throughput of the aes algorithm on the pascal gpu architecture, in: 2018 3rd International Conference on System Reliability and Safety (ICSRS), IEEE, 2018, pp. 97–103.

[13] A. A. Abdelrahman, M. M. Fouad, H. Dahshan, Analysis on the aes implementation with various granularities on different gpu architectures, Advances in Electrical and Electronic Engineering 15 (3) (2017) 526–535.

[14] J. Ma, X. Chen, R. Xu, J. Shi, Implementation and evaluation of different parallel designs of aes using cuda, in: 2017 IEEE Second International Conference on Data Science in Cyberspace (DSC), IEEE, 2017, pp. 606–614.

[15] M. Cianfriglia, S. Guarino, Cryptanalysis on gpus with the cube attack: design, optimization and performances gains, in: 2017 International Confer-

ence on High Performance Computing & Simulation (HPCS), IEEE, 2017, pp. 753–760.

[16] A. Ahmadzadeh, O. Hajihassani, S. Gorgin, A high-performance and energy-efficient exhaustive key search approach via gpu on des-like cryptosystems, The Journal of Supercomputing 74 (1) (2018) 160–182.

[17] A. Bogdanov, L. R. Knudsen, G. Leander, C. Paar, A. Poschmann, M. J. Robshaw, Y. Seurin, C. Vikkelsoe, Present: An ultra-lightweight block cipher, in: International Workshop on Cryptographic Hardware and Embedded Systems, Springer, 2007, pp. 450–466.

[18] G. Leander, Small scale variants of the block cipher present., IACR Cryptology ePrint Archive 2010 (2010) 143.

[19] T. Baignères, P. Junod, S. Vaudenay, How far can we go beyond linear cryptanalysis, Lecture Notes in Computer Science 3329 (2004) 432–450.

[20] D. Kirk, et al., Nvidia cuda software and gpu parallel computing architecture, in: ISMM, Vol. 7, 2007, pp. 103–104.

[21] J. Chen, A. Miyaji, C. Su, J. Teh, Improved differential characteristic searching methods, in: 2015 IEEE 2nd International Conference on Cyber Security and Cloud Computing, IEEE, 2015, pp. 500–508.