Accelerating Mutual Information Analysis Based Power Analysis Attacks using the GPU

Malin Prematilake*, Buddhi Wickramasinghe†, Olitha Vithanage‡, Roshan Ragel§, Hasindu Gamarachchi¶
Department of Computer Engineering, Faculty of Engineering
University of Peradeniya
Email: *mainprematilake@gmail.com, †bcwicki@gmail.com, ‡olitha92@gmail.com, §ragelrg@gmail.com, ¶hasindu2008@gmail.com

***Abstract*—Side Channel Attacks are a popular modern cryptanalysis technique used by adversaries in embedded devices to break the security key. In these types of attacks, the attackers are keen on identifying the weaknesses of the physical implementation of the cryptosystem and utilize such vulnerabilities to extract the key. Power Analysis Attack is a form of Side Channel Attack in which, the adversary exploits power consumed by a cryptographic device during encryption to obtain the key. Mutual Information Analysis (MIA) is a concept introduced in information theory that measures the dependence between two random variables. In MIA based Power Analysis Attack, mutual information between two random variables is taken as the side channel distinguisher. Here, the two variables are physical leakages of the device and the power model based on key estimates. Since this method has more advantages to attackers compared to other methods, it is vital for cryptanalysts to find better countermeasures against this. But, due to the lack of efficient implementations it is hard for cryptanalysts to do that kind of research. In this paper, we present a methodology to accelerate MIA based Power Analysis Attacks using a GPU (Graphical Processor Unit) like NVIDIA Compute Unified Device Architecture (CUDA). Our proposed method promises to better utilize the capabilities of NVIDIA CUDA and obtain a speedup of more than 100 times compared to its sequential version.**

I. Introduction

Cryptography is a methodology used to communicate messages securely, in such a way that only the intended parties will be able to read them. In cryptography, sending party will convert the plain text message into a different form known as cipher text using a secret key. This is called encryption. At the other end, the receiver will convert the cipher text back to plain text using the same key or a different one. This reverse process of encryption is known as decryption. Usually, any cryptographic algorithm is publicly available. Therefore, secrecy of a cryptosystem is entirely dependent on the secret key. In this study, an encryption standard known as AES (Advanced Encryption Standard) is attacked [1].

Cryptanalysis is the process of identifying weaknesses in a cryptographic algorithm and retrieving the secret key using those weaknesses. There is a wide variety of methods which can be used for cryptanalysis. Those methods include traditional methods such as brute force attack and more modern methods such as differential cryptanalysis, linear cryptanalysis and integral attacks [2].

II. Power Analysis Attacks and Side channel attacks

Side Channel Attacks are also a methodology used for cryptanalysis. These attacks utilize on physical implementation weaknesses of a cryptosystem to retrieve the secret key.

A hardware based [2] or software based cryptosystem [2] is actually implemented in a physical device, access to which may be available to an adversary. Although the device may be tamper resistant, an adversary might be able to collect some valuable side channel information during the encryption or decryption process. Therefore, side channel attacks have gained more popularity.

Power Analysis Attack is a form of Side Channel Attack. Here the adversary exploits power consumption data of a cryptosystem during the encryption process to get the secret key. Power consumed by a cryptographic device depends on the data processed and operations carried out within the device [2]. For example, a significant difference in power consumption can be observed when a bit changes its value. The idea of Power Analysis Attacks is to compare physical power leakages of a cryptographic device with a set of key dependent leakage predictions in order to identify which key must have most probably given rise to the actual leakage [3]. The set of key dependent leakage predictions is usually known as a power model. Comparison between the two sets of data is done through a statistical method known as a side channel distinguisher. Types of power analysis attacks vary depending on the distinguisher.

There are mainly two types of Power Analysis Attacks,

- Simple Power Analysis (SPA) Attack
- Differential Power Analysis (DPA) Attack

and based on the distinguisher used, DPA can have several sub types such as Correlation Power Analysis (CPA) Attack and Mutual Information Analysis (MIA) based Attack [2]. Out of them, SPA is the simplest form, in which the attack is conducted using a very little number of samples [2]. On the other hand, DPA makes use of large number of power traces and statistical techniques to conduct the attack [2]. MIA based Power Analysis Attack is the most recent form of Power Analysis Attack introduced in 2008 [4].

A. Mutual Information Analysis based Power Analysis Attack

In this type of attack, mutual information between two random variables is taken as the side channel distinguisher. This method is known to have many advantages over correlation based attack [3]. A Power Analysis Attack using MIA does not require any prior knowledge or assumptions about the cryptographic device. Also, the relationship between physical leakages and power model is not necessary to be linear. Therefore, it can be used in scenarios where data dependencies are not very significant.

1) Mutual Information: Mutual Information is a concept introduced in information theory. It is a measure of dependence between two random variables. Mutual information between two random variables X and Y is, how much information about X can be obtained by observing Y [3]. When X and Y are discrete random variables, mutual information between them is given as,

$$I(X,Y) = \sum_{k=x}^{y} P(X=x, Y=y).\log(\frac{P(X=x,Y=y)}{P(X=x).P(Y=y)})$$

Here, $P(X=x, Y=y)$ is the joint probability distribution of X and Y. $P(X=x)$ and $P(Y=y)$ are marginal probability distributions of X and Y respectively. Key to calculating mutual information between two random variables is, calculating the probability density functions of joint probability between X and Y and marginal probabilities of X and Y. There are several methods of calculating probability functions, namely histogram method, kernel density estimation, data clustering and vector quantization [3].

2) MIA in the context of Power Analysis Attacks: MIA can be used as a side channel distinguisher for Power Analysis Attacks. Here, the required two random variables are physical leakages of the device and power model built based on key estimates. Key estimates are the set of all possible values that a given key can take. The power model is usually referred to as the hypothetical leakage values. There are different kinds of methods which can be used to obtain the hypothetical leakage values. Out of them, Hamming weights calculated for key estimates are used as the leakage model in this context.

III. NVIDIA CUDA

CUDA is a parallel computing platform and an application programing interface developed by NVIDIA corporation. It can be used to offload high intensive computations to the GPU [5]. The computation is distributed in a grid of thread blocks. All blocks contain the same number of threads that execute a program on the device known as the kernel [6]. A three dimensional block ID is used to identify each block. Each thread within a block can be identified by a three dimensional ID for easy indexing of the processed data and the data to be processed. Execution configuration (or the block and grid dimensions) can be set at run time and is based typically on the dimensions and size of the processed data.

Threads within a block can co-operate via a shared user managed cache memory(16 KB or 48 KB), where it lacks a similar mechanism for co-operation between blocks. This is a concern which makes programs such as histogram difficult and inefficient. Access to global memory (device's DRAM) has a high latency (order of 400-600 clock cycles) which makes I/O operations to the global memory expensive. Focused design on the kernel and execution configuration helps avoid the above mentioned latency. The performance of global memory access can be reduced severely unless access to adjacent memory locations are coalesced.

MIA is a compute intensive process. However it is made up of a set of repetitive operations. But instead of repeating them in a sequential manner, they can be executed simultaneously independent from each other. Since GPUs are built specifically for parallel programming, the MIA process can be executed much more efficiently using the GPU. In addition, MIA based power analysis attacks have many benefits for attackers compared to other attacks as mentioned above. Hence more research is now being done to discover countermeasures against this. But since MIA is a compute intensive process, testing countermeasures has become a time consuming task, which in turn reduces the efficiency of researchers. Hence speeding it up is very helpful for such work. Therefore, in this study, we have proposed an approach to make MIA based implementation much faster than the implementations that has been done so far.

IV. Related Work

Mangard [2] provides an overview of power analysis attacks.His book provides a sound knowledge of power analysis attacks, how to prepare the test bed and how to carry out an attack.

Gierlichs et al. [4] proposed for the first time that mutual information analysis can be used as a side channel distinguisher for power analysis attacks. The authors have also confirmed the proposed method using a practical experiment on an AES software implementation with a 128-bit key. However, since the experiments have been conducted only to prove the theoretical proposal, performance measurements have not been conducted. Thus, it does not discuss its efficiency compared to other methods or how it can be improved.

Charvillon and Standaert [7] in their research show how traditional statistical methods can be used with the model proposed in [4]. It is a study of how the theoretical findings given in [4] can be applied in practical situations. Experiments conducted here address the question of when to use MIA based power analysis. As mentioned in [3], correlation based power analysis attacks are much efficient when the noise factor is ignored. Therefore, here the authors have proven that MIA based power analysis attacks are more useful when the leakage model is not entirely precise. But this research only studies how efficiency can be improved by using two types of methods (histogram based and kernel density estimation based) for calculating mutual information.

Shams [5] discusses how MIA based power analysis attacks can be accelerated using a Graphical Processor Unit (GPU). An NVIDIA CUDA based device has been used in this study. Here the author has used the histogram method and how

data dependency when creating histograms limits the use of GPU in histogram calculation is clearly highlighted in this. According to the experimental results, the GPU implementation has achieved around 25 times performance improvement compared to the CPU implementation. In Hudde [8], the author has tried to improve the MIA based attack using NVIDIA CUDA. But the author has gained only 4 to 12 times of computational speed up compared to its sequential CPU counterpart. Although both these authors have been able to achieve a performance improvement compared to a CPU implementation, we expect a much better improvement through this work.

V. Problem and Motivation

Power analysis attack is one of the most studied types of side channel attacks, but the number of researches done on MIA based power analysis attack is very small and the research done on accelerating these attacks using the GPU is smaller. When we study the related work, we see that although some work has been done on MIA based power analysis attacks, there is still room left for improvement. Although Hudde [8] has achieved a speed up of 12 times and Shams [5] has achieved a speed up of 25 times, current GPU architectures provides newer capabilities which should enable a better speed up. These new capabilities cannot be utilized in old implementations. Hence a newer GPU implementation would be useful. Improving the efficiency of MIA based power analysis attacks allows cryptanalysts to identify weaknesses in current encryption standards, improve them and introduce better countermeasures. In addition, we believe that these improvements can be used in other applications where MIA is of use like in identifying cancer cells [9] and in comparing the similarities between two images [10].

VI. The MIA Algorithm

Calculation of MI value between a power model value and a power consumption data value is based on calculating marginal probability of the two variables and the joint probability between those two variables. Out of several methods proposed in [3], histogram method was chosen for the calculation of probability distributions as it is widely used and much simpler than the other methods. The algorithm for obtaining the mutual information is of five parts.

1) Generating the power model values for all bytes in each plain text sample
2) Normalizing the power consumption data
3) Calculating the marginal probabilities of the two random variables
4) Calculating the joint probability of the two random variables
5) Calculating the mutual information value

Step 1 can be done in several ways. However, in this study, the hamming weight between the plain text and the cipher text is used [7] and [3]. Steps 2 and 3 are straight forward. However, calculating the joint probability is a very computationally intensive process. Since the encryption algorithm under study is AES with a key size of 128, there are 16 bytes (key bytes) that needs to be estimated. For each key byte, there are 256 possible values. The data set used in this study consists of 200 plain text samples with 100,000 sample wave points for each sample. Since MIA values need to be calculated for each pair of data sets (i.e.[power model, power consumption data]), a total of $100,000 * 16 * 256 = 2^{12} * 10^5$ calculations must be done. This is more cleary show in Fig.1 given below.

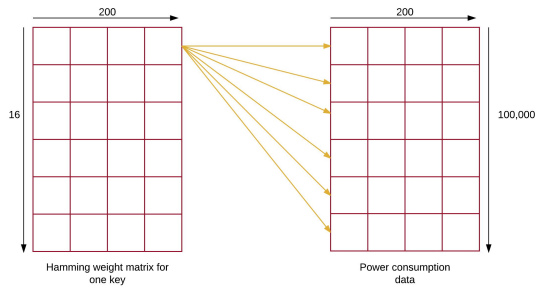

Fig. 1. Total number of MI calculations needed

Algorithm 1 Calculating joint probability

```
jProb is the joint probability of the pair
MIval is the mutual information value of the pair
marg1 is the marginal probability of power model data
marg2 is the marginal probability of wave data

pm ← matrix[256 × 16 × 200] Power model data
wd ← matrix[10^5 × 200] Power consumption data

for i ∈ {1, ..., 256} do //no. of keys
    for j ∈ {1, ..., 16} do //no. of key bytes per key
        for k ∈ {1, ..., 100000} do //wave data points
            array1 ← pm[i * 256 + j]
            array2 ← wd[k]
            jProb ← jointProb(array1, array2)
            MIval ← MI(marg1, marg2, jProb)
```

Algorithm 1 above describes how joint probability is calculated. As can be seen, calculating the joint probability is the process that requires the most time and resources due to its repetitiveness. When the basic GPU implementation was profiled using the NVIDIA Visual Profiler (NVVP) as in [11] and it showed that 70% of the computation time is spent for calculating the joint probability. Hence, attention was given to reduce the execution time of this process.

VII. Methodology and Implmentation

By implementing a parallelized version of the sequential algorithm that was initially developed, several approaches that can help to reduce the execution time were identified.

1) Better memory usage
2) Using approximated power values instead of using perfect values

A. Better memory usage

1) Design: In the sequential implementation, memory for intermediate variables (e.g. matrices) were allocated dynamically. Although this is easier in terms of implementation, it also increases the execution time. This is because allocating memory inside a CUDA kernel (dynamic memory allocation) takes a considerably larger amount of time compared to allocating it before the kernel call. Hence allocating all the memory that is needed before calling the particular CUDA kernels should be more efficient. However as mentioned before, a more complex implementation is required by this. The complexity is increased by the fact that the matrices are stored in row major order, regardless of their dimensions.

Probabilities for hamming weight matrix and the power consumption data matrix is created separately. In addition, the number of states of each dataset (i.e. the array) is stored in a separate array as well. Using the two arrays the size of each joint probability array (as shown in Fig.2) and the total size required by the joint probability matrix can be calculated (i.e. the last element of Total_Size_Array). Hence the total size needed can be allocated prior to the kernel call. As shown in Fig.2, each element of Total_Size_Array represents the total size of the probability matrix up to a given row. Hence the difference of two adjacent cells provides the size of each row of joint probability matrix.

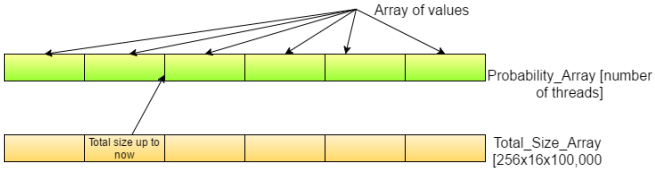

Fig. 2. Indexing of the probability matrix

2) Implementation: One of the toughest challenges in using MIA is using memory of the GPU efficiently. This is because histogram based MIA algorithm requires a lot of memory for intermediate variables. The largest of them requires an average of 58.5GB of memory, even for processing data for breaking one key out of 256 possibilities. This is for a dataset of 200 plain text samples with 100,000 sample wave points for each sample. In addition, the memory required depends on the nature of the data as well. That is, for two data sets of the same size, different amounts of memory will be needed. This is because when higher the number of different values in the set, higher the number of different bins that needs to be created for each of them.

A single thread is utilized for calculating one MIA value (i.e. one thread takes an array each from the two probability matrices and calculates the particular joint probability and the MIA value). Hence more threads utilized in one kernel call, the less time consuming the calculation will be. However, if more threads are used, then obviously the size of the matrices must be increased in order to accommodate the data. Hence, there is a limit to the number of threads that can be used. For example, the average number of threads that can be used for a dataset with 200 plain text samples with 100,000 data points is 800,000, where the data related to only 8 of the 16 key bytes of one key guess value could be processed in one kernel call.

Hence, the kernel for calculating the joint probability values and the MIA values is called repeatedly using a loop. For the above example the kernel must be called 512 times to calculate all of the MIA values. The average execution time for the data set in the above example is 14.34 minutes.

B. Using approximated values instead of using perfect values

1) Design: As it was mentioned in the previous section, a separate kernel for calculating joint probability and the resulting MI value was developed. This kernel had to be called iteratively, since joint probability distributions corresponding to all combinations of keys, key bytes and wave data points could not be stored in the GPU memory at once.

The main reason for running out of memory was the size of joint probability distributions being very large. According to the definition of joint probability, its size is given by the multiplication of the sizes of corresponding marginal probability distributions. Reducing the size of these distributions using a histogram approximation method was considered [8]. Previously, a bin was allocated to each distinct value in the data set when creating the histogram. A histogram for a power consumption dataset in this implementation had a average bin count of about 500. The same for a hamming data set was around 8. Hence the resulting joint probability distribution consisted of around 4000 bins. Instead of using one bin per value, it was decided to allocate one bin for a range of values. The two designs are given in Fig.3. In the modified design of this example, two distinct values fall into one bin. For instance, all 0s and 1s are in one bin, 2s and 3s are in one bin and so on. Thus, it is clearly seen that the number of bins has been halved. Although this might reduce the accuracy comparatively, the sizes of probability distributions will be reduced. This in turn allows utilization of more threads for joint probability and mutual information calculation.

An important point to note is that, histogram based density estimation is highly dependent on data. Therefore, when approximating the bins, accuracy will be different depending on the data. Hence, the implementation had to be done so that it is possible to change the range of bins and test.

2) Implementation: In the previous implementation where one bin is allocated to each distinct value of the dataset, first the data set was normalized so that all values were converted into integers between 0 and (maximum_value - minimum_value). Values were stored in an integer type array. The number of distinct values (Number_of_Bins) contained in each dataset (i.e. (maximum_value - minimum_value)+1) was also calculated for each data set and stored in a separate array. Histogram for a dataset is a double type array with a size equal to Number_of_Bins. The straightforward method of calculating a histogram using these data structures was given in Algorithm 2 below. Here, the Number_of_Bins is equal to the number of distinct data values in the dataset.

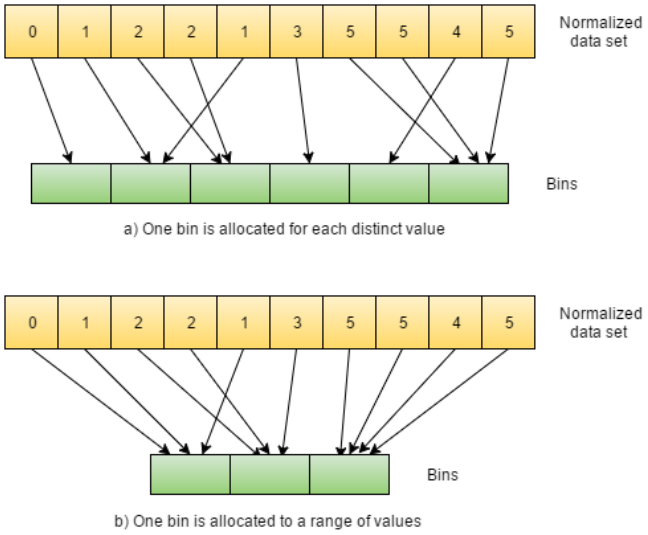

Fig. 3. Allocation of values to bins in the two designs

Algorithm 2 Basic histogram development

```
1: for Each bin number i do
2:     result[i] = 0;
3: for Each data point j do
4:     result[data[j]] ++;
```

In the new implementation, the normalizing procedure was modified in a way that the same function (i.e. Algorithm 2) can be used for histogram calculation. First, the values were converted into integers from 0 to (maximum_value - minimum_value), same as before. In this case, a constant was defined as WIDTH, which gives the number of distinct integers that falls into one bin. Number_of_Bins of the resulting histogram was calculated as given below.

$$Number\_of\_bins = ceil(((maxvalue - minvalue) + 1)/WIDTH)$$

An array with size equal to that of dataset, which stores the bin number that each data point falls into was created during the normalizing procedure. This method was directly plugged into the histogram calculation method. It should be noted that, this approximation method was only used when calculating marginal probability distribution of power consumption data. This is because, the number of bins corresponding to a power consumption dataset is several times greater than that of hamming weight data. Thus, it has a greater impact on the size of joint probability distribution. Also, since the range of values in hamming data is small, approximation will reduce the accuracy of them greatly.

After the approximation method was developed, this was combined with the memory reduction techniques discussed in the previous section. The new implementation reduced the memory requirements of joint probability calculation by a considerably large amount. Hence, the number of threads that can be allocated for the kernel execution also increased. More precisely, a grid consisting of 100,000 blocks, each having 128 threads was utilized for joint probability calculation. As a result, it was possible to process MI values related all 16 key bytes of 8 key guesses using one kernel call. The dataset used was the same as before, with 200 plain text samples with 100,000 data points.

VIII. Results and Discussion

All tests were carried out on a computer with an Intel Core i5-4570 processor, 32GB of RAM and a NVIDIA Tesla K40 GPU with 12GB of memory. It had Linux installed as the operating system. Several sets of power traces collected previously were used in these tests.

When the execution time of different GPU implementations mentioned above are compared, a gradual decrease can be seen as shown in Fig.4. Here, a data set of 200 plain text samples with 100,000 wave points for each sample is used. An average time of 4 hours and 30 minutes is required by the CPU sequential version. Compared to this, only an average of 58 minutes is required by the basic GPU version (this the first version that was created without using any of the above mentioned techniques. the CPU algorithm was parallelized as it is). However, this version does not use any of the two methods mentioned above to make it more efficient. Therefore most of the memory available in the GPU could not be utilized. Hence only 256 threads could be used in one kernel call. As a result this version is only 4.65 times faster than its sequential counterpart.

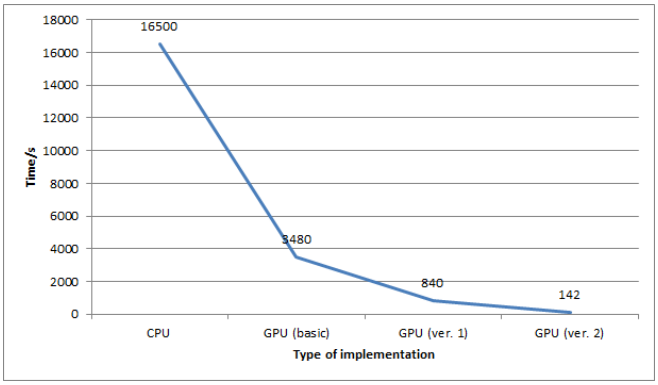

Fig. 4. Execution time of different implementations

However, less time is required by the implementation described in section VIII.a . As mentioned, it utilizes in a much more efficient manner and therefore more threads can be deployed. As a result more calculations can be done in one kernel call. For a dataset of the size mentioned above, it requires 840 seconds in average. This is a improvement of 19.2 times of the execution time, compared to the sequential version.

Execution time is further reduced by using approximated values as discussed in section VIII.b, where data related to

8 key guesses (out of 256) could be processed in one kernel call. Since this approach reduces the memory consumption further more, by combining this and the previous approach, the execution time is reduced down to an average of 142 seconds. Hence a speed up of 114 times is achieved.

By changing the width of the bin of used in the above implementation the execution time can be reduced. As shown in Fig.5 the execution time is inversely proportional to the bin width. However, this could not be improved further as the results (the key) began to be faulty for the datasets that were used.

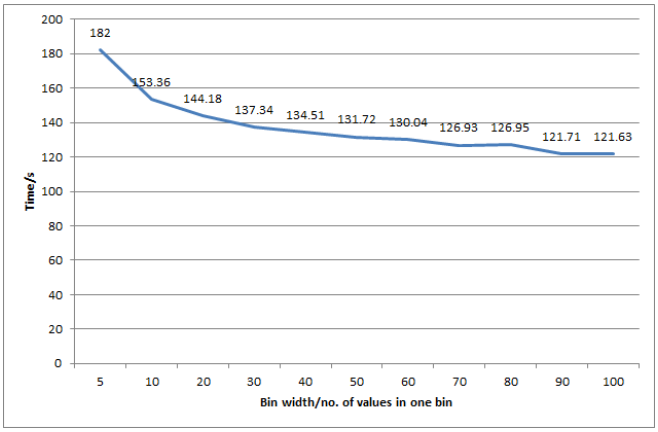

Fig. 5. Variation of execution time against different bin widths

IX. Conclusion

In this paper, an improved method for MIA based power analysis attacks using NVIDIA CUDA is presented. Though MIA is more compute intensive compared to other methods such as CPA (Correlation Power Analysis) attacks, still the fact that it does not need a linear relationship between the power consumption data and the power model makes it much more useful. It has been shown in this paper how this method could be accelerated using GPUs. Some studies have been done in the past, such as [5] and [8] but our implementation is more than four times faster than the fastest among them, mainly because our implementation uses several methods to reduce the memory usage and increase the number of parallel computations per time unit. However, we believe that more improvements could be made to this method to make it much more efficient. For this, attention must be given to reducing the execute time of joint probability density function calculation.

X. Future work

Although the histogram method has been used in this phase of the study, there are other methods for performing MIA more efficiently and out of them kernel density estimation has shown better performance than the others [3]. Kernel density estimation is non-parametric way to find the probability density function of a random variable [12]. This method considers each data point seperatly, unlike the histogram method where the data is normalised. Hence this method is expected to give much more accurate results.

Acknowledgment

The authors would like to thank the NVIDIA Corporation on behalf of NVIDIA Research Center at the Department of Computer Engineering, University of Peradeniya for letting us use the Tesla K40 GPU to successfully carry out the research.

References

[1] J. Daemen and V. Rijmen, *The design of Rijndael: AES-the advanced encryption standard*. Springer Science & Business Media, 2013.

[2] S. Mangard, E. Oswald, and T. Popp, *Power analysis attacks: Revealing the secrets of smart cards*. Springer Science & Business Media, 2008, vol. 31.

[3] L. Batina, B. Gierlichs, E. Prouff, M. Rivain, F.-X. Standaert, and N. Veyrat-Charvillon, "Mutual information analysis: a comprehensive study," *Journal of Cryptology*, vol. 24, no. 2, pp. 269–291, 2011.

[4] B. Gierlichs, L. Batina, P. Tuyls, and B. Preneel, "Mutual information analysis - a generic side-channel distinguisher," in *International Workshop on Cryptographic Hardware and Embedded Systems*. Springer, 2008, pp. 426–442.

[5] R. Shams and N. Barnes, "Speeding up mutual information computation using nvidia cuda hardware," in *Digital Image Computing Techniques and Applications, 9th Biennial Conference of the Australian Pattern Recognition Society on*. IEEE, 2007, pp. 555–560.

[6] C. Nvidia, "Compute unified device architecture programming guide," 2007.

[7] N. Veyrat-Charvillon and F.-X. Standaert, "Mutual information analysis: how, when and why?" in *Cryptographic Hardware and Embedded Systems-CHES 2009*. Springer, 2009, pp. 429–443.

[8] H. C. Hudde, "Gpu assisted mutual information analysis attacks on aes," *Bachelor Thesis, Ruhr-Universität Bochum*, 2010.

[9] G. T. Klus, A. Song, A. Schick, M. Wahde, and Z. Szallasi, "Mutual information analysis as a tool to assess the role of aneuploidy in the generation of cancer-associated differential gene expression patterns." in *Pacific Symposium on Biocomputing*, vol. 6, no. 42-51. Citeseer, 2001, p. 176.

[10] D. B. Russakoff, C. Tomasi, T. Rohlfing, and C. R. Maurer Jr, "Image similarity using mutual information of regions," in *European Conference on Computer Vision*. Springer, 2004, pp. 596–607.

[11] NVIDIA. (2015) Profiler user's guide. [Online]. Available: http://docs.nvidia.com/cuda/profiler-users-guide/index.html

[12] M. Rosenblatt *et al.*, "Remarks on some nonparametric estimates of a density function," *The Annals of Mathematical Statistics*, vol. 27, no. 3, pp. 832–837, 1956.