# Correlation power analysis attack on software implementation of TRIVIUM stream cipher

Rangana De Silva*, Iranga Navarathna and Malitha Kumarasiri

Department of Computer Engineering,
University of Peradeniya, Sri Lanka
Email: ranganades@gmail.com
Email: iranganavaratna@gmail.com
Email: malithakumarasiri93@gmail.com
*Corresponding author

Chai Wen Chuah

Department of Information Security and Web Technology,
Tun Hussein Onn University of Malaysia, Malaysia
Email: cwchuah@uthm.edu.my

Janaka Alawatugoda

Rabdan Academy,
Dhafeer Street, Abu Dhabi, UAE
Email: jalawatugoda@ra.ac.ae

**Abstract:** Power analysis attacks are a category of attacks against cryptographic implementations. In this case, the power consumption of a cryptosystem is analysed to extract its secret values such as secret keys and key streams. This has become a huge threat to modern day cryptosystems. Therefore, identifying cryptographic implementations which are vulnerable to power analysis attacks is very important. Many studies have been carried out on power analysis attacks on block cipher implementations, but relatively less number of studies have been carried out on power analysis attacks on stream cipher implementations. This paper presents a power analysis attack on a software implementation of TRIVIUM stream cipher. In order to analyse the power consumption, correlation power analysis (CPA) is done, and the keystream is successfully recovered.

**Keywords:** side-channel attacks; correlation power analysis attacks; stream ciphers; TRIVIUM.

**Reference** to this paper should be made as follows: De Silva, R., Navarathna, I., Kumarasiri, M., Chuah, C.W. and Alawatugoda, J. (2022) 'Correlation power analysis attack on software implementation of TRIVIUM stream cipher', *Int. J. Information and Computer Security*, Vol. 19, Nos. 3/4, pp.379–401.

**Biographical notes:** Rangana De Silva is a researcher in CEB Smart Meter Project at the Department of Computer Engineering in University of Peradeniya, Sri Lanka. He obtained his BSc in Engineering from the University of Peradeniya, Sri Lanka. He obtained his internship at the Secure Systems Group, Aalto University, Finland. He worked as a research assistant on runtime scope enforcement against data oriented programing attacks. His main research interest is on side channel attacks against cryptographic systems.

Iranga Navarathna is a software engineer at the Geveo Australasia, Sri Lanka. He obtained his BSc in Engineering from the University of Peradeniya, Sri Lanka. He obtained his internship at the Vega Innovations, Sri Lanka. He worked as a research intern on the AiGrow project which develops self-driven greenhouses. His main research interest is on embedded system security.

Malitha Kumarasiri is a software engineer at the CodeGen International, Sri Lanka. He obtained his BSc in Engineering from the University of Peradeniya, Sri Lanka. He obtained his internship at the Synopsys Lanka (PVT) Ltd, Sri Lanka as a research and development engineer in Electronic Design Automated Tools Development team. His main research interests are on machine learning and data mining.

Chai Wen Chuah is a Senior Lecturer at the Faculti Sains Komputer Dan Teknologi Maklumat of Tun Hussein Onn University of Malaysia. She obtained her Bachelor's degree from the Tun Hussein Onn University of Malaysia, Master's degree from the University of Science, Malaysia and PhD from the Queensland University of Technology, Australia. She has received a number of research grants. Her main research interest is secure stream ciphers.

Janaka Alawatugoda is an associate researcher and Assistant Professor at the Research and Innovation Centers, Rabdan Academy, UAE, and an Adjunct Research Fellow of the Institute for Integrated and Intelligent Systems, Griffith University, Australia. He obtained his BSc from the University of Peradeniya, Sri Lanka and PhD from the Queensland University of Technology, Australia. He is a Fellow of the British Computer Society, member of the Computer Society (Sri Lanka), a member of the Association for Computing Machinery and a member of the International Association for Cryptologic Research. His main research interest is cryptography.

## 1 Introduction

Cryptanalysis is the art of deciphering ciphertext without having access to the secret key. Cryptanalysis is used to discover the vulnerabilities of the ciphers. Hence, it is valuable when it comes to developing secure encryption algorithms. Most of the cryptanalysis involves mathematical analysis of the encryption algorithms while some others target weaknesses in the implementation of the encryption process. The attacks which focus on the weaknesses of the physical implementation of cryptosystems are known as side-channel attacks.

Side-channel attacks are a type of attack in cryptanalysis. The main channel in a cryptographic operation is where the plaintext is encrypted using a secret key, and a ciphertext is generated. When an attacker observes the encryption process and tries to find out information to eventually find the secret key of the operation through physical effects of the encryption, it is called a side-channel attack. These attacks exploit vulnerabilities in the physical implementation of the system to decrypt the ciphertext. Some of the side-channels which can be exploited are power consumption, electromagnetic leaks, timing information, sound, and heat. These physical properties have a relationship with the encryption process. Therefore, the secret key of a system can be derived by analysing these properties.

A side-channel attack that analyses the power consumption of an embedded system is carried out in this research. The attack is best known as 'power analysis attacks'. The first reported power analysis attack on a cryptographic operation was published by Kocher et al. (1999). The initial attack was implemented on a block cipher, DES (National Institute of Standards and Technology, 1993) and public key encryption, RSA (Rivest et al., 1978). The secret keys of these cryptographic implementations were easily revealed through power analysis attacks. Power analysis became an important category in breaking the cryptographic implementations with the inception of these attacks. Block ciphers caught more attention in the beginning, and only a limited number of research is done on stream ciphers at present. Stream ciphers make non-repeatable streams of key bits, and a different key-bit is used to encrypt every bit of the plaintext. Whereas in block ciphers, the same key is used for every block of the plaintext. Hence, power analysis on stream ciphers is a bit difficult than on block ciphers. The secret key or some information relevant to the key can be deduced by analysing the power consumption of the device while it is doing encryption or decryption. This attack is undertaken by analysing power traces obtained by a set of different plaintexts and using power analysis algorithms such as differential power analysis (DPA), correlation power analysis (CPA) or simple power analysis (SPA). The secret key can be deduced using one of the power analysis algorithms mentioned above. Since the plaintexts or in some cases the corresponding power traces for different intermediate values are taken, this falls under a known plaintext attack. Noted that, the assumption of known plaintext attack is allowing attackers to access some part of plaintext and the corresponding ciphertext. By simple XOR operation, the attacker will retrieve the particular part of the keystream. However, the attacker will not know the secret key, the internal state as well as the remaining keystream. Hence, CPA is used in this research to analyse the power consumption and deduce the secret key.

### *1.1 CPA attacks*

CPA (Brier et al., 2004) is a type of DPA attack. Here, conclusions on the secret key are deduced using the correlation coefficient of statistics. CPA uses the Pearson correlation as a distinguisher to identify the most likely correct hypothesis of the (sub) key candidate. If this is identified, then information about the secret key of the system can be obtained. In comparison to other power analysis methods such as DPA and SPA, CPA requires less number of power traces to find information about the secret key. Only one power trace is required per plaintext in CPA whereas an average power trace from many power traces is obtained in DPA. The operation flow of CPA attack is illustrated in Figure 1.

The correlation coefficient $\rho(X,Y)$ can be used to calculate the linear dependency between two sets of variables such as $X$ and $Y$. This value lies between –1 and $1(-1 \leq \rho \leq 1)$. A value closer to 1 means that there is a strong positive relationship between $X$ and $Y$ whereas, a value closer to –1 suggests a strong negative correlation between $X$ and $Y$. If the value of the correlation coefficient is 0, it means there is no correlation between data points of $X$ and $Y$. This process can be denoted mathematically by the equation given below.

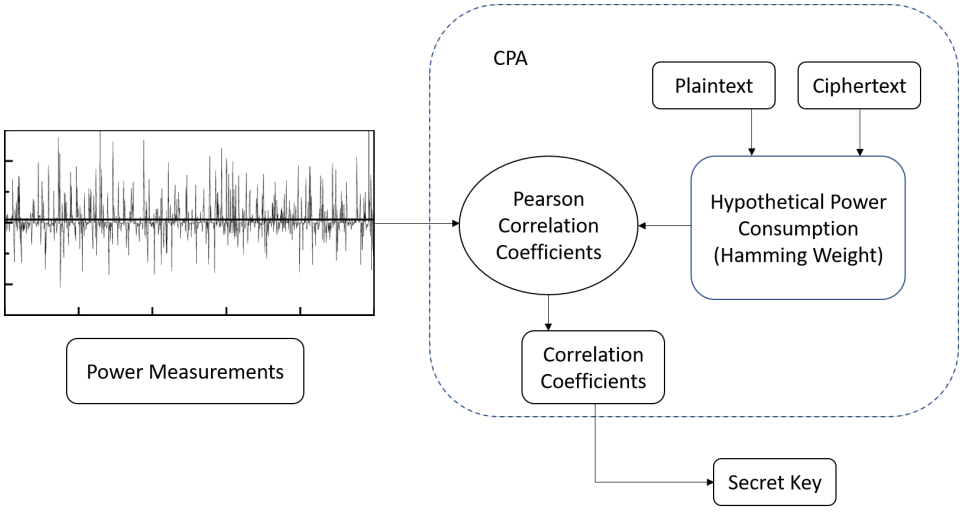
$$\rho(X,Y) = \frac{Cov(X,Y)}{\sqrt{Var(X).Var(Y)}}. \tag{1}$$

This $\rho$ can be estimated using the following equation,

$$r = \frac{\sum_{n=1}^{n}(x_i - \overline{x})(y_i - \overline{y})}{\sqrt{\sum_{n=1}^{n}(x_i - \overline{x})^2 (y_i - \overline{y})^2}}. \tag{2}$$

In CPA attacks, the above equations can be used to find the correlation between the actual power consumption of the signal generated in the cryptographic device and hypothetical power traces. After calculating the correlation, the hypothetical power traces with the highest correlation is selected, and the key corresponding to these hypothetical power traces is identified as the key of the cryptographic device. Many power analysis attacks using CPA have been carried out to identify vulnerabilities in ciphers. Some of the ciphers known to be vulnerable to power analysis attacks are AES (Lo et al., 2017), Speck (Gamaarachchi et al., 2017) and MICKEY (Liu et al., 2010). However, most of the power analysis attacks which have been carried out have been on block ciphers. Hence, this research targets a stream cipher known as TRIVIUM.

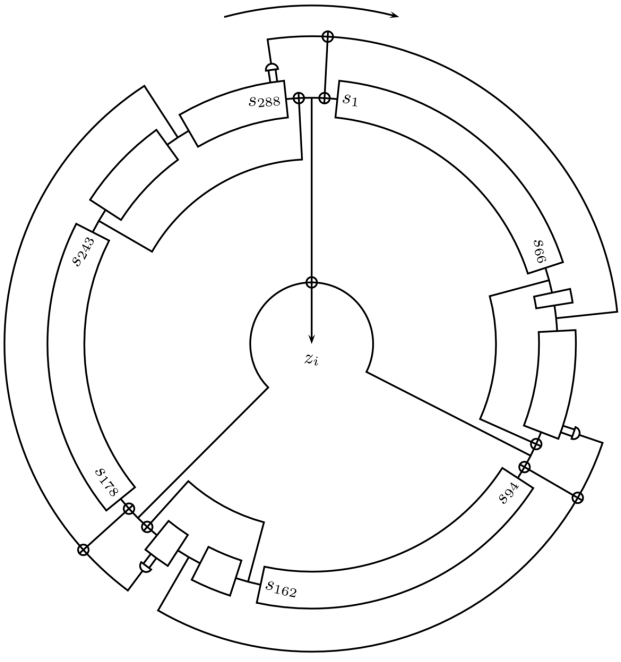
**Figure 1** CPA attack structure (see online version for colours)



### *1.2 TRIVIUM stream cipher*

TRIVIUM (De Canniere and Preneel, 2006) is one of the hardware synchronous stream ciphers presented in the eSTREAM project. The word 'TRIVIUM' is set to indicate that this stream cipher is very trivial or simple in design. Despite that fact, it has a very efficient hardware implementation. It is formed with the use of 11 XOR gates, 3 AND gates, and 3 shift registers. The addition of the AND gates make this a nonlinear stream cipher. An 80-bit secret key and an 80-bit initial vector ($IV$) are fed to the cipher to generate a keystream up to $2^{64}$ bits. Then, the keystream is XORed with the plaintext to generate the ciphertext. One bit of the keystream is generated per clock cycle, and in 64 clock cycles, the keystream is generated. TRIVIUM does not use previous internal states for a new keystream generation, and hence this cipher can be parallelised to carry out 64 operations per clock cycle and to output 64-bit word per clock cycle. Hence, this makes higher non-algorithmic noise and in turn, performing power analysis attacks on this cipher becomes more difficult (Gierlichs et al., 2018). Figure 2 shows the internal structure of the TRIVIUM stream cipher.

**Figure 2** TRIVIUM internal structure




*Source:* De Canniere and Preneel (2006)

TRIVIUM has a 80-bit length secret key and a 80-bit length IV and three nonlinear shift register (NFSRs). These shift registers are of different lengths (93, 84 and 111 bits). These registers are named from $S_1$ to $S_{288}$. First register, second register and third register are denoted as $S_1$ to $S_{93}$, $S_{94}$ to $S_{177}$ and $S_{178}$ to $S_{288}$ accordingly.

TRIVIUM goes through a set up time of $4 \times 288$ iteration steps and then outputs a key stream bit $Z_i$. The setup and output generation is a recursion formula and can be described as follows ($\oplus$ denotes XOR).

$$(a_1, ..., a_{93}) = (0, ..., 0, k_{80}, ..., k_1)$$
$$(b_1, ..., b_{84}) = (0, 0, 0, 0, IV_{80}, ..., IV_1)$$
$$(c_1, ..., c_{111}) = (1, 1, 1, 0, ..., 0)$$
$$a_{i+93} = a_{i+24} \oplus c_i \oplus c_{i+1}c_{i+2} \oplus c_{i+45}$$
$$b_{i+84} = b_{i+6} \oplus a_i \oplus a_{i+1}a_{i+2} \oplus a_{i+27}$$
$$c_{i+111} = c_{i+24} \oplus b_i \oplus b_{i+1}b_{i+2} \oplus b_{i+15}$$
$$z_i = a_i \oplus b_i \oplus c_i \oplus a_{i+27} \oplus b_{i+15} \oplus c_{i+45}$$

Here, $i$ ranges from 1 to $4 \times 288$ in the set up stage and there after for output generation $i$ is greater than $4 \times 288$. The proposed attack is at the re-synchronisation phase of the TRIVIUM cipher. That is where the initial set up is done along with setting up the secret key and the IV. The inventors mention that the TRIVIUM is also efficient as a software implementation (De Canniere and Preneel, 2006).

### *1.3 Our contribution*

Initially, we implement the TRIVIUM stream cipher on a verified testbed. Then, we establish two CPA attack models to recover the keystream of the TRIVIUM stream cipher:

1 byte attack model

2 bit attack model.

The byte attack model could successfully recover the keystream. We also discuss the feasibility of the bit attack model to recover the secret key. We investigate the leakage contribution of operations for performing the XOR operation during the encryption step, and compare the leakage of the XOR operation itself to the memory access of reading the operands and writing the results. As per our knowledge, this is the first CPA attack done on software implementation of the TRIVIUM stream cipher (particularly, on a PIC18F2550).

In Section 2, previous research on power analysis on stream ciphers is discussed. Then, in Section 3, the overall methodology on the two CPA attack models on the TRIVIUM testbed is elaborated. Next, in Section 3, we evaluate the results of the experimental attack models, and finally, in Section 4, we review the impact of power analysis attacks on the TRIVIUM stream cipher.

## 2 Related works

Power analysis attacks analyse the power consumption of a cryptosystem to derive the secret key. These attacks are very active on embedded devices such as smart cards. Gamaarachchi and Ganegoda (2018) have designed a testbed for power analysis attacks on block ciphers such as Speck (Beaulieu et al., 2013). However, in our work, this testbed is used to find vulnerabilities in TRIVIUM stream cipher.

### 2.1 Power analysis attacks on stream ciphers

Stream ciphers are well known for their security mechanisms. Hence, these can be used for video streaming and secure wireless connections. There are two types of stream ciphers namely, software and hardware stream ciphers. The eSTREAM project introduced four software stream ciphers and three hardware stream ciphers. The hardware stream ciphers are Grain v1, MICKEY 2.0 and TRIVIUM. Hardware stream ciphers are more efficient than the software stream ciphers. Generally, they are used for hardware applications and embedded systems with limited storage, power consumption, and gate count. Therefore, researchers continue to focus their work on finding vulnerabilities in hardware stream ciphers.

#### 2.1.1 Power analysis attacks on TRIVIUM

Jia et al. (2012) propose a methodology to implement CPA on TRIVIUM stream cipher. Here, a mathematical model based on the Hamming power model has been designed to carry out the attack. A simulation according to their theoretical model has been executed to derive the secret key. It does not provide any information on hardware simulation of the attack, and thus one could still try this implementation practically on a hardware environment to verify the proposed attack's possibility. Therefore, the main focus is on the practical implementation of this attack on TRIVIUM cipher.

Fischer et al. (2007) practically implemented an attack on TRIVIUM cipher using DPA. When comparing with the CPA, DPA needs more power traces to derive the secret key. Since the CPA method for power analysis is more efficient than the DPA, it is used to check if TRIVIUM is vulnerable. Tena-Sánchez and Acosta (2016) implemented an optimised DPA attack on TRIVIUM using correlation shape distinguishes. Moreover, a simulation based on this approach has been mounted to check the vulnerability of the cipher. Atani et al. (2008) implemented this cipher using two different submicron technologies to check the most resistant technology using the DPA attack. They simulated it using the software-based approach as well. Both of these attacks (Tena-Sánchez and Acosta, 2016; Atani et al., 2008) have not been tested using hardware. However, our research involves hardware testing. Furthermore, CPA is used in our research which causes the attack to be more efficient.

Sim et al. (2020) outlines a DPA attack on TRIVIUM which reduces the keystream guessing space from 80 to 14 bits. One aspect to point out is, Our research focuses on CPA, which uses less number of power traces for the retrieval of the key, and on the other hand we prove that XOR operation between the keystream and the plain text is vulnerable against CPA and if proven TRIVIUM is reversible then by solving equations backward we can derive the secret key.

#### 2.1.2 Power analysis attacks on other stream ciphers

In the work of Sandeep and Rajesh (2010), both CPA and DPA methods are used to retrieve the secret key of the MICKEY-128 2.0 stream cipher. They have shown that MICKEY-128 cipher is vulnerable to both attacks. Liu et al. (2010) also propose a methodology to retrieve the secret key of the MICKEY-128 2.0 cipher using a CPA method. In the work of Chakraborty and Mukhopadhyay (2016), they have practically implemented MICKEY-128 2.0 cipher on a Xilinx Virtex-5 FPGA device. Here they

have followed a different approach than previous attempts. They have used the least squares support vector machine (LS-SVM) classification algorithm to determine the key bit. Zadeh and Heys (2013), mounted a SPA attack on grain. This is a theoretical attack that illustrates the vulnerabilities of grain. Fischer et al. (2007) and Chakraborty et al. (2015) explain DPA methods to retrieve the secret key successfully of the grain stream cipher. This research work shows that other stream cipher implementations are vulnerable to power analysis attacks. However, not much work has been published on attacking TRIVIUM implementation using power analysis. Therefore, our research is done with the aim of addressing this issue.

### *2.2 CPA on block ciphers*

Speck (Beaulieu et al., 2013) is a block cipher which was published by the National Security Agency in 2013. This encryption algorithm uses three major operations namely, add, rotate and XOR. Since these operations have a low memory requirement compared to traditional block ciphers, the Speck algorithm is considered to be a lightweight block cipher. According to the power analysis attack proposed by Gamaarachchi et al. (2017) for Speck, two sets of power measurements are required. The first set of power traces is used to derive the right half of the secret key. Then another set of power traces and the derived right half of the key is used to derive the left half of the secret key. For both of these power measurements, the XOR operations of the algorithm have been selected as the intermediate states. In this attacking process, the CPA method has been used to derive the secret key. Similarly, in our attack which is done on the TRIVIUM stream cipher, a XOR operation is selected as the intermediate state. However, in Gamaarachchi et al. (2017) the XOR operation involves a byte, and in our case, the XOR operation is done bit by bit.

## 3 Methodology

TRIVIUM is a synchronous stream cipher. It generates up to $2^{64}$ bits using an IV and key of 80-bits each. This bit stream is generated bit by bit. Hence, all of the operations in this algorithm are bit manipulations. Performing a power analysis attack using CPA on a cipher which does bit manipulations is relatively tough. Even though TRIVIUM is built for hardware platforms, it is also efficient in software applications. Hence, identifying power analysis attack vulnerabilities in the software implementation of TRIVIUM is essential. In this section, two approaches to attack the TRIVIUM stream cipher is explained. The source code for Subsections 3.2 and 3.3 is available online (Dilshan, 2018). Furthermore, these attacks were executed on release mode, and had no interference from the debug tool.

### *3.1 Testbed*

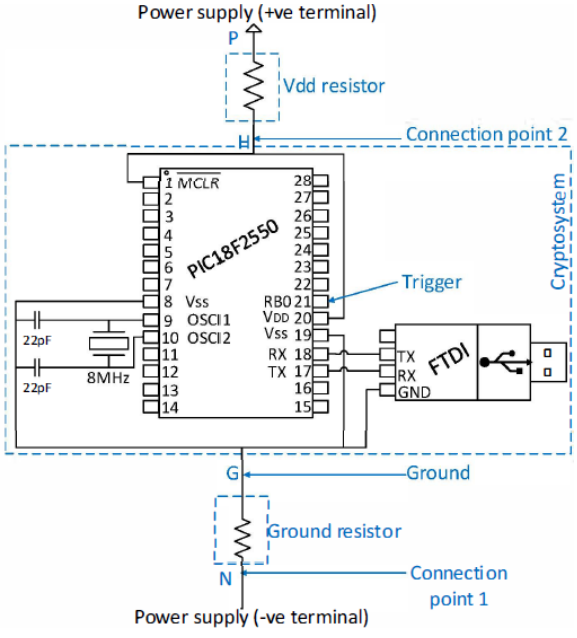
In order to achieve this goal, first, it is required to mimic the TRIVIUM encryption. The testbed developed by Gamaarachchi and Ganegoda (2018) is used in this research. Most of the software resources needed for this testbed are available online (Gamaarachchi, 2015). Previously, this testbed has been used to attack AES and Speck ciphers. After

setting up the testbed, TRIVIUM is implemented in it. Thereafter, power traces are obtained from the testbed, while the encryption process takes place. Next, these power traces are analysed in order to derive the key of the encryption.

The testbed which is set up for the experiment consists of an oscilloscope, PIC micro-controller, FTDI device, and a PCB. The oscilloscope is used to capture power traces while the encryption takes place. A PIC micro-controller is used to implement the cipher. FTDI device is used for the communication between the computer and micro-controller.

The micro-controller is programmed to encrypt plaintext continuously which is given via USB. A PIC18F2550 is used for this purpose. It has 2,048 bytes of memory which is sufficient for the implementation of AES as well as TRIVIUM. The PIC is working on an 8 MHz external clock which is mounted on the PCB. CCS PIC C is used to compile C code and produce the hex file for the micro-controller. This hex file is then burnt to the micro-controller using a PICkit 3. MPLAB IPE is used as the interface when using the PICkit 3. The key is hard-coded to the micro-controller to verify the encryption and ciphertext is generated for different plaintext.

**Figure 3** Circuit diagram for power analysis on TRIVIUM (see online version for colours)




*Source:* Gamaarachchi (2015)

A Tektronix MSO2012B oscilloscope is used to obtain power traces. These power traces are then analysed to derive the secret key of the encryption. Gamaarachchi (2015) presents two power capturing methods namely, *ground resistor* and *VDD resistor* method. These are based on the position of the resistor. Figure 3 demonstrates both methods. In our approach *ground resistor* method was used. Hence, VDD resistor in Figure 3 does not exist in our testbed. The oscilloscope is connected to the PCB using two probes. One probe is used to measure the power level. The other probe is used to

trigger the specific point where the power trace of the encryption should be recorded. The section triggered can be configured using the micro-controller. The oscilloscope is connected to the computer using a USB type B cable. The recorded power traces are saved in the computer using this connection.

### 3.2 Attacking software implementation of TRIVIUM using CPA: bit attack model

TRIVIUM is a hardware oriented stream cipher. Hence, the way a CPA attack is executed is different than in a block cipher. This difference originates from the structure (Figure 4) of the TRIVIUM algorithm. The proposed attack is executed at the resynchronisation phase of the TRIVIUM cipher. That is where the initial set up is done along with setting up the secret key and the IV.

At encryption in TRIVIUM, values of shift registers vary. Those variations cause dynamic power consumption. This leakage of information from the internal flip flops can be modelled by a hypothetical power model. It can be used to correlate with the actual power traces to retrieve secret information of the algorithm. Hence, it is appropriate to use Hamming distance to model the power consumption of the hardware implementation of TRIVIUM. Thus, the hypothetical power consumption of TRIVIUM can be modelled for every clock cycle with the consideration of state change in the 288 state bits.

This CPA attack aims to find intermediate variable values of TRIVIUM by solving non-linear equations sequentially to disclose the secret bits of the key. Hence, the proposed method is to use a CPA attack for the first 80 rounds of the initialisation phase of the TRIVIUM.

An intermediate value is selected from the algorithm (the selection function) which includes a part of the secret key as well as a variable data portion. The plaintext is kept fixed, and an $IV$ is selected as the variable data portion. According to the initialisation algorithm it is seen that, $b_{i+84} = b_{i+6} \oplus a_i \oplus a_{i+1}\dot{a}_{i+2} \oplus a_{i+27}$ qualifies as the selection function as it has portions of both $IV$ and the secret key (K) (at the first stage key is stored in register and $IV$ is stored in register $b$).

An hypothesis function is formed as this: $\sigma_i$ where, $\sigma_i = a_i \oplus a_{i+1}\dot{a}_{i+2} \oplus a_{i+27}$. Total of 80 hypothesis equations are required to reveal all the 80-bits of the secret key. This can be done with one CPA attack, targeting $\sigma_i$ bits ($\sigma_1 - \sigma_{80}$).

Then, with all the $\sigma_i$ equations and the initial register values of registers a,b, and c, 80 non-linear equations which would contain 80 unknown $k_i$ can be derived. The entire key of the algorithm can be now determined through solving the equations with known ($\sigma_i$) values.

Let us focus on the CPA attack which determines the 80 bits of $\sigma_i$. Power consumption has to be measured at the encryption process for chosen data blocks. An arbitrary (256, 500, 1,000, ...) number of IVs can be used for the attack. $IV$s are changed by re-synchronising the device with different $IV$s. Each time Iv is changed, power traces are recorded. Likewise, power traces of 80 cycles in the initialisation of TRIVIUM for each $IV$ used can be obtained.

Obtaining hypothetical power traces is the next step. Intermediate value has two parts, the $IV$ portion and the hypothesis ($\sigma$). Sigma can either be 1 or 0 as it represents a state change of a bit. The particular $IV$ bit in the intermediate state, changes over the first 80 cycles of the initialisation phase. It is directly mapped to the successive bits in the $IV$. Hence, a hypothetical power model addressing the power consumption of the

intermediate value at 80 cycles of the initialisation phase can be developed. Thus, by correlating these hypothetical values with the real power traces, correlation coefficients for sigma bit guesses (0 and 1) can be derived at each iteration up to 80.

Pearson correlation is used to identify the highest correlating points in these two matrices. Moreover, by doing so, the values of all the $\sigma_i$ values are correctly derived. 80-bits of the secret key are derived by sequentially solving equations derived earlier and known sigma values.

### *3.3 Attacking software implementation of TRIVIUM using CPA: keystream attack model*

All the operations do not occur in a bit-wise manner in TRIVIUM even though it is a stream cipher. This gives TRIVIUM some qualities of a block cipher. In other words, the ciphertext is generated byte-by-byte. A plaintext byte is put into an XOR function with a keystream byte to generate a ciphertext byte. Thus, A CPA attack targeting this byte operation can be used to retrieve the keystream of the encryption.

CPA can be implemented to retrieve a selected portion of the keystream. Initially, it is suitable to try and retrieve one byte of the keystream. The selection function is the XOR operation between the plaintext byte and the keystream byte. Here, the plaintext is the variable data portion. The value of the ciphertext byte after the XOR operation between the modified plaintext byte and keystream byte is selected as the intermediate value. It is dependent on the keystream and the plaintext, which should be the case for a known plaintext power analysis attack.

First, the power traces must be captured such that those include the power consumption of the device during the first iteration of keystream generation. The next step is the CPA of power traces to recover keystream byte. The keystream is attacked as one byte at a time attack. For each possible 256 combinations of a keystream byte, hypothetical power consumption values are calculated using the Hamming weight model. Pearson correlation is used to identify the highest correlating points afterward. The value with the highest correlation would be the correct keystream byte.

This attack can be altered to match the size of the keystream that is extracted. The time taken for complete extraction of the keystream depends on the size of the plaintext.

## 4 Results and discussion

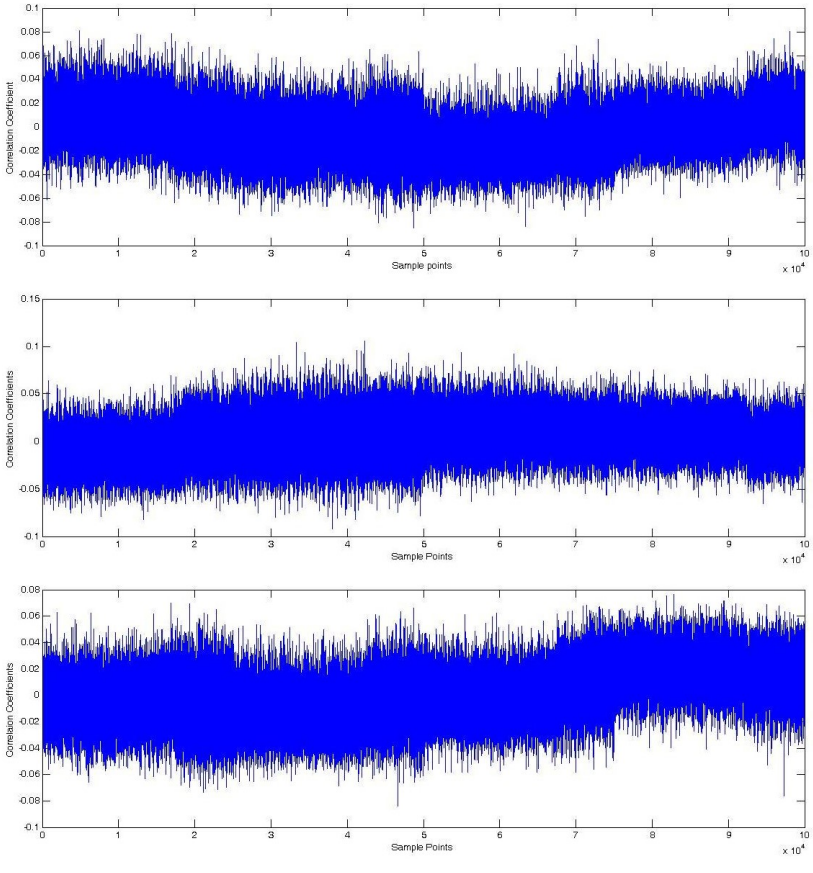
The application of CPA with only having 1 bit as the key-guess bit became sophisticated due to the lack of variance in the hypothetical power model. Thus, it was hard to find patterns in the variation of correlation coefficients. The recorded coefficients were not indicating high values. Hence, CPA attacks were carried out on XOR operations between two bytes as well as two bits.

### *4.1 The behaviour of the coefficients when executed for first 80 cycles in TRIVIUM*

This setup is used to find the correlation coefficients with time. For the first step of setup, the secret key was set to zeroes. In theoretical-based, the first 13 sigma values are correlate to the latter bits of the secret key. Hence, all the sigma values are considered

must give a higher correlation to the 0 guesses. However, results of experiment showed the attack was not successful as the guessed values were wrong. Figure 4 illustrates the variation of correlation coefficients with time for the first three correct sigma bit guesses over 1,700 power traces (sample points – 100,000). The traces do not point out significant peak points or a common pattern. Theoretically, there should be a peak point at the leftmost side of the graph moving to the right as the guessed sigma value increases. However, that pattern is not visible. This may be due to the lack of power traces obtained or the noise created by other operations in the triggered power trace.

**Figure 4** Sample points vs. correlation coefficients (see online version for colours)




## *4.2 The behaviour of the XOR byte operation on the same setup*

Figure 5 represents the variation of correlation values with time of the correct key guess in XOR byte operation (sample points – 100,000). Three significant peaks can be seen here. The first peak is the power variation when an $IV$ is loaded to the micro-controller, and the rightmost peaks are for the read and write operations of the byte to the micro-controller. This behaviour proves that an XOR byte operation can be attacked using CPA.

### *4.3 The behaviour of the XOR bit operation on the same setup*

Figures 6 and 7 demonstrate variation of correlation coefficients with number of traces for correct key guess 01 and 00 respectively. Both graphs show a significant drop in the coefficient values at the beginning and have slight fluctuations towards the end. Both instances were tested for 20,000 power traces to see the variation of correlation coefficients as well as the correctness of the guessed key. Figure 6 proves that bit XOR can be attacked by stating a higher correlation for the correct key guess, whereas Figure 7 states the wrong key. Both graphs look similar if not for the marginal differences between the correlation values. Hence, this result is not enough to prove that a bit XOR operation can be attacked with CPA.

**Figure 5** XOR byte: sample points vs. correlation coefficients (see online version for colours)

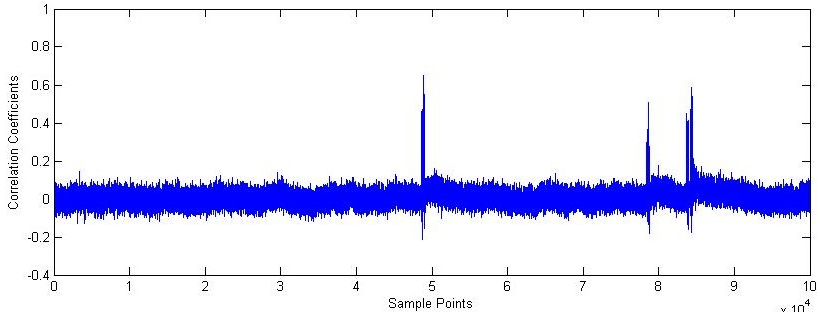



**Figure 6** XOR test: correct key 01 (see online version for colours)

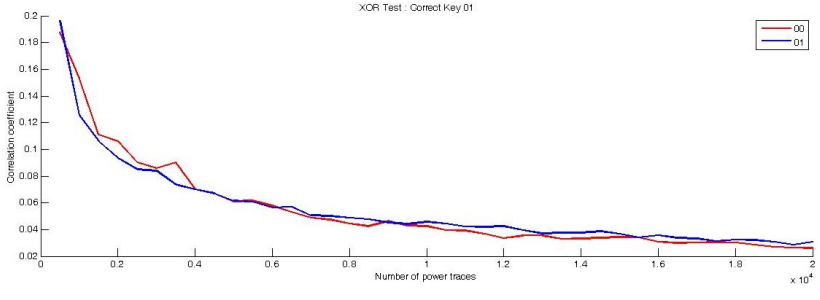
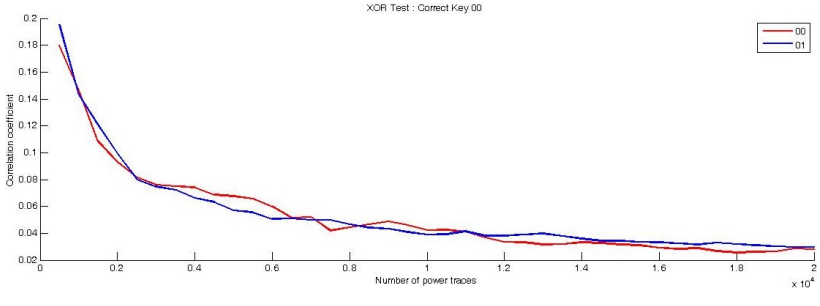



**Figure 7** XOR test: correct key 00 (see online version for colours)



Since byte XOR operation can be attacked using CPA, another way to attack TRIVIUM is to attack the XOR operation between the keystream and the plaintext, and then solve equations backward to derive the secret key, if only TRIVIUM is reversible.

**Figure 8** Correlation coefficients vs. time/sample points (see online version for colours)

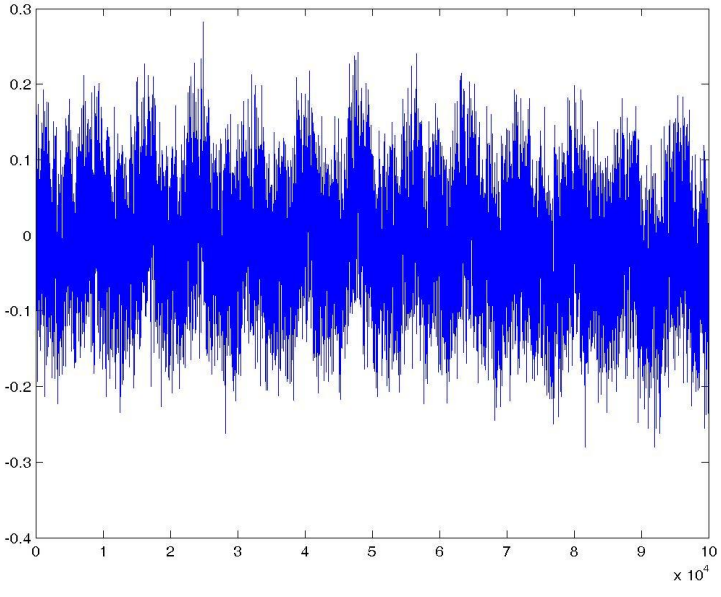
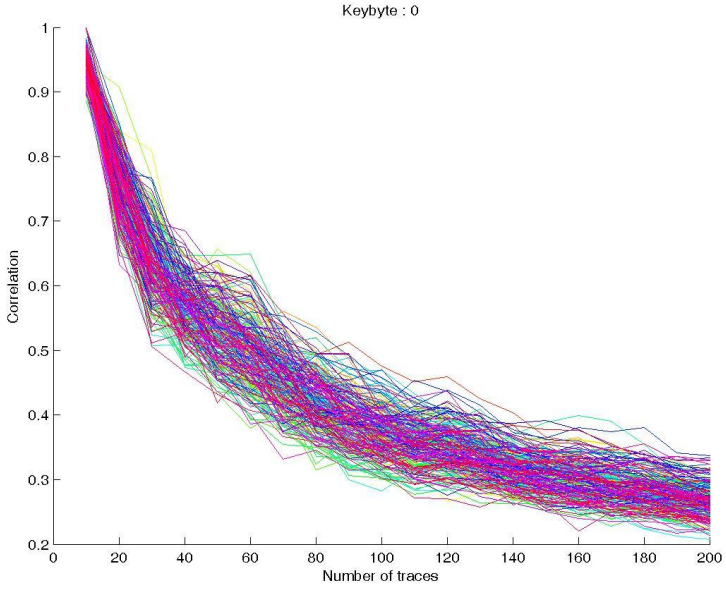



**Figure 9** Number of traces vs. correlation coefficients (see online version for colours)



### *4.4 Key stream attack*

The XOR operation between the keystream and the plaintext generates the ciphertext. Since the ciphertext is public, knowing the keystream would reveal the plaintext. Hence, an attack on the XOR operation between a keystream byte and a plaintext byte was approached. Furthermore, there is research work (Maximov and Biryukov, 2007) done in order to derive the key of TRIVIUM using the keystream.

#### *4.4.1 CPA on XOR of keystream and plaintext byte*

Initiated the attack by taking 200 power traces triggering the XOR operation and executing CPA on the traces. Figure 8 represents the variation of coefficients with time for the expected keystream byte. Due to the amount of noise, no peak point could be found in the plot. The maximum coefficient was less than 0.3 which cannot be considered as a peak point for a successful attack.

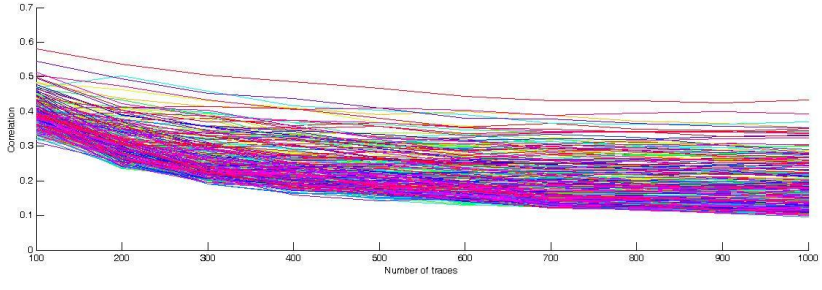
Figure 9 illustrates the variation of different key byte guesses with many power traces (sample points – 100,000). It was unable to identify a curve that would separate itself from the rest and remain at a higher level of the correlation value. Hence, the first attempt on attacking the XOR operation between the plaintext and the keystream was not successful.

By inspecting every stage up to the key extraction, it was suspected that the above-mentioned issue might be a synchronisation error caused by MATLAB and the oscilloscope. To further elaborate, stabilising the triggered power traces and capturing the traces through MATLAB with proper synchronisation must be done. If not it may cause noise in power traces.

#### *4.4.2 Key stream attack isolating the XOR operation*

As it was proven that CPA could be done on an isolated XOR operation, the next step was to isolate the XOR operation in TRIVIUM completely by extracting XOR operand values through global variable assignments and perform CPA. Isolated XOR byte operation did not have synchronisation issues. Therefore, this experiment would confirm that XOR byte operation in TRIVIUM is vulnerable to CPA if it is conducted with proper synchronisation between the stages.

**Figure 10** Number of traces vs. correlation coefficients (see online version for colours)




As expected, the correct keystream byte was retrieved through CPA. Figure 10 demonstrates the variation of coefficients with many traces and the correct guess curve is

separating itself from the rest. Even though this attack was successful, the vulnerability of keystream cannot be fully proven due to the fact the XOR operand values were extracted from the source, and the XOR operation was done in an isolated manner.

### 4.4.3 Key stream attack without any modification in the code

In order to perform this, the proper synchronisation between MATLAB and the oscilloscope had to be achieved. It is quite challenging to find the precise time intervals to wait for the trigger from the oscilloscope and for the oscilloscope to capture a trace.

**Figure 11** Code snippet of XOR in code (see online version for colours)

```
output_high(PIN_B0);
for(nop_count = 0; nop_count<100;nop_count++){
    #asm
    nop
   #endasm
  }
input_backup[(length - mark)] ^=keystream;
for(nop_count = 0; nop_count<100;nop_count++){
    #asm
    nop
   #endasm
  }
output_low (PIN_B0);
for(nop_count = 0; nop_count<100;nop_count++){
    #asm
    nop
   #endasm
  }
```

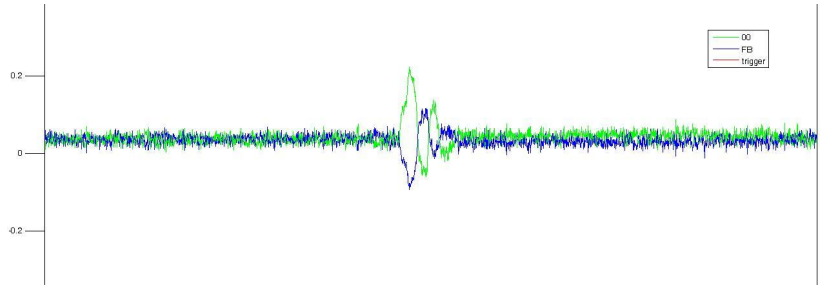
Eventually, it was found that there is a specific operation mode called 'normal mode' in the oscilloscope to capture a trace and hold that until another trace is captured. In other words, there would not be a situation where there is no triggered trace on the oscilloscope output. Hence, the problems caused due to the inability to synchronise operations in the testbed were solved by using the 'normal mode'.

Figure 11 is a snippet of the TRIVIUM source code where the XOR operation happens between the plaintext byte (it is loaded into an array called `input_backup`) and the keystream byte. `output_low` and `output_high` are the commands used to trigger the power consumption within that code block. Apart from that inline assembly NOP instructions are put to reduce the noise and make it easier to identify the peak points when doing CPA.

Given that the synchronisation issue was fixed, and CPA on XOR byte operation was already tested and successful, correct key guesses from CPA was expected. The expected result was 0xFB as the keystream byte, but 0x00 had the highest correlation value. Figure 12 shows correlation coefficients are varied with time/sample points for expected key guess 0xFB (blue graph) and CPA key guess 0x00 (green graph). A couple

of significant peaks can be seen in both expected and actual guesses. It is clear that these peak points are corresponding to the different stages in the XOR operation. The power consumption in an XOR operation may differ in the way that operation is executed. For instance, an XOR operation between array elements will differ with a simple XOR operation between two variables.

**Figure 12** Correlation coefficients vs. time/sample points on expected (0xFB) and guessed (0x00) key guesses (see online version for colours)




Another interesting point to note is that CPA returned the same result 0x00 for other expected keystream bytes as well. This was the same for the increased number of traces as well. The reason for this falsely recognising of the key as 0x00 is related to the power consumption of loading plaintext from memory. In the triggered XOR operation, initially, the plaintext is loaded from memory and then only the XOR operation takes place. When a number is in an XOR operation with 0, the result would be that number itself. Hence, if the plaintext is in an XOR operation with 0, the result would be the plaintext itself. During CPA all the key possibilities are checked. When 0 is used as the key guess in the CPA algorithm, it correlates power consumption at the time where the plaintext is loaded to memory with hypothetical power consumption calculated for the XOR operation with key guess 0. This would result in a high correlation and a peak in the plot. This can be avoided either by trimming out the power consumption relating to the loading of the plaintext or do that in the code itself by not including it in the triggered trace. However, it has to be done systematically. That is discussed in the next section.

### *4.4.4 Experiments on keystream attack*

The main reason behind these experiments is to clearly understand the stages in the XOR operation in relevance to the CPA algorithm. Three experiments were conducted to identify the issue. All the experiments are for 100,000 sample points. They are as follows.

1 Triggering the XOR operation itself along with writing the result to memory, and writing that back from memory to register.

2 Triggering the same setup with only changing the precedence of the XOR operation.

3 Triggering only the part which writes the result of the XOR operation to memory.

### 4.4.5 Experiment 1

Figure 13 breaks down the relevant power consumption portions of the XOR function into three parts. At first, three new variables named $t1$, $t2$, $t3$ are defined. $t1$ contains the first byte of the plaintext, $t2$ contains the value of the XOR operation. The value of $t2$ is written to an array, and that value is written to variable $t3$. Hence, $t3$ contains the final result of the XOR operation as well. It is important to note that the plaintext loading portion is removed from the triggered section. There are three main experimental operations in this code block. These will be called steps 1, 2 and 3 in the rest of this section.

1 $t2 = t1 \oplus \texttt{keystream}$: Reads plaintext byte from $t1$ and XOR it with `keystream` byte and then writes the value to $t2$ (register operations).

2 $\texttt{input\_backup}[\texttt{length} - \texttt{mark}] = t2$: Writes $t2$ value to the specific index in array(write memory operation).

3 $t3 = \texttt{input\_backup}[\texttt{length} - \texttt{mark}]$: Loads from specific index in the array to $t3$ (read memory operation).

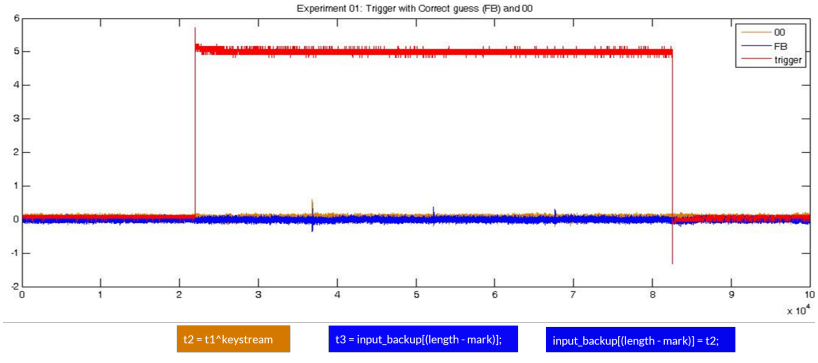
**Figure 13** Experiment 1 code snippet of XOR in code (see online version for colours)

```
output_high(PIN_B0);
for(nop_count = 0; nop_count<50;nop_count++){
   #asm
   nop
  #endasm
}
t2 = t1^keystream;
 for(nop_count = 0; nop_count<50;nop_count++){
   #asm
   nop
  #endasm
}
input_backup[(length - mark)] = t2;
for(nop_count = 0; nop_count<50;nop_count++){
   #asm
   nop
   #endasm
 }
 t3 = input_backup[(length - mark)];
for(nop_count = 0; nop_count<50;nop_count++){
   #asm
   nop
   #endasm
 }
output_low (PIN_B0);
```

Figure 14 illustrates the variation of correlation coefficients with sampling points for both correct guess (0xFB) (blue colour graph) and (0x00) (orange colour graph). The trigger line is also depicted to show which section of the power trace is captured. At the bottom, three boxes are marked with extractions of the experimental portions of

the code. The colour of the box indicated which key guess prevailed in peak points concerning the value of correlation coefficients.

**Figure 14** Trigger with correct guess (0xFB) and (0x00) (see online version for colours)




Steps 2 and 3 resulted in giving the correct key guess 0xFB with CPA. However, in step 1, the key guess was wrong. This result convinced that power consumption in memory read/write operations is higher than that in register operations but, still the reason behind the result of step 1 is not known. In order to identify this, the assembly level code has to be examined to see how it is handled. The impact of the assembly code would become self-explanatory with the result of the next experiment.

### *4.4.6 Experiment 2*

The only difference made in this experiment is swapping the operations in step 1. Other sections remain the same. Figure 15 is the code snippet of the triggered block.

1 $t2 = \texttt{keystream} \oplus t1$: Reads `keystream` byte and XOR it with plaintext byte from $t1$ and then writes the value to $t2$ (register operations).

2 $\texttt{input\_backup}[\texttt{length} - \texttt{mark}] = t2$: Writes $t2$ value to the specific index in array(write memory operation).

3 $t3 = \texttt{input\_backup}[\texttt{length} - \texttt{mark}]$: Loads from specific index in the array to $t3$ (read memory operation).

Figure 16 illustrates the variation of correlation coefficients with sampling points for both correct guess (0xFB) (blue colour graph) and (0x00) (orange colour graph). The trigger line is also depicted to show which section of the power trace is captured. At the bottom, three boxes are marked with extractions of the experimental portions of the code. The colour of the box indicates which key guess prevailed in peak points concerning the value of correlation coefficients.

Steps 1, 2 and 3 resulted in giving the correct key guess 0xFB with CPA. The reason for this behaviour lies in the assembly code extraction depicted in Figure 17.

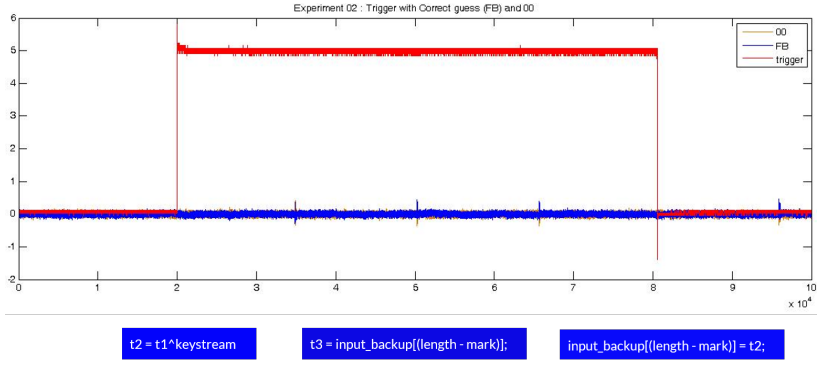
In experiment 1, it reads from the register file and loads the plaintext into the working register initially, and then does the XOR operation. However, after the variables are swapped as given in experiment 2, only the keystream is loaded to the working

register. The cause of this problem is connected to the correlation values related to loading the plaintext vs. loading the keystream. When plaintext is loaded, it causes correlation with 00, which in turn would return the key as 00. This issue is caused by the compiler. If the compiler optimises the code to handle precedence in an XOR function, this scenario would not have occurred.

**Figure 15** Experiment 2 code snippet of XOR in code (see online version for colours)

```
output_high(PIN_B0);
for(nop_count = 0; nop_count<50;nop_count++){
   #asm
   nop
  #endasm
}
t2 = keystream^t1;
 for(nop_count = 0; nop_count<50;nop_count++){
   #asm
   nop
  #endasm
}
input_backup[(length - mark)] = t2;
for(nop_count = 0; nop_count<50;nop_count++){
   #asm
   nop
   #endasm
  }
 t3 = input_backup[(length - mark)];
for(nop_count = 0; nop_count<50;nop_count++){
   #asm
   nop
   #endasm
  }
output_low (PIN_B0);
```

**Figure 16** Trigger with correct guess (0xFB) and (0x00) (see online version for colours)




### *4.4.7 Experiment 3*

This experiment executes only step 3 (code snippet Figure 18). In experiment 2 variables had to be swapped to get the correct guess. Here, the possibility to obtain the correct

key by only triggering the XOR operation value which is loaded/written to memory is checked. It should be noted that memory read/write operations consume more power. Hence, the effect of noise would fade out, and the CPA should work in theory.

As expected, Figure 19 shows that the correct guess can be obtained with CPA. Thus, it is proven that no swapping is needed for the attack and it can be done by only capturing the memory write operation.

**Figure 17** Assembly code extraction of the XOR operation (see online version for colours)

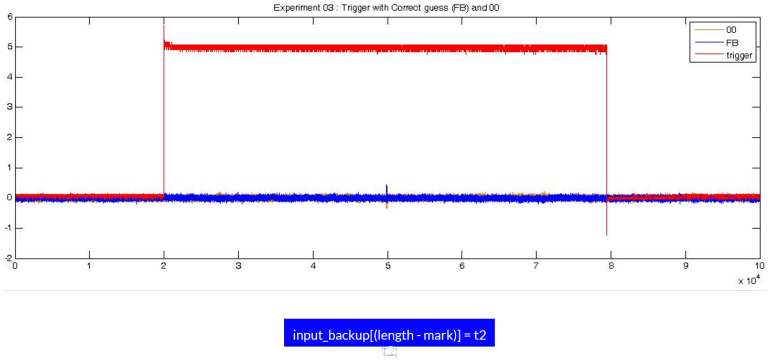
```
09DC:  INCF   x88,F
09DE:  BRA    09D4
.....          t2 = t1^keystream;
09E0:  MOVF   x8A,W
09E2:  XORWF  x87,W
09E4:  MOVWF  x8B
```

t1 - Plaintext
x8A - t1
x87 - Keystream
W - Working Register

**Figure 18** Experiment 3 code snippet (see online version for colours)

```
output_high(PIN_B0);
  for(nop_count = 0; nop_count<50;nop_count++){
     #asm
     nop
    #endasm
  }
  input_backup[(length - mark)] = t2;
  for(nop_count = 0; nop_count<50;nop_count++){
     #asm
     nop
     #endasm
    }
output_low (PIN_B0);
```

**Figure 19** Trigger with correct guess (0xFB) and (0x00) (see online version for colours)



## 5 Conclusions

It is evident that the CPA on TRIVIUM implemented on an 8-bit micro-controller requires a significantly higher number of power traces to derive secret information about the secret key. The main reason for that is the complexity associated with executing a CPA attack on a bitwise XOR operation. The fewer number of guesses used in the Hamming power model acts as a limiting factor concerning noise reduction in the correlation traces. This incident rose the question about the vulnerability of an XOR operation against power analysis. It was seen that XOR operation between two bytes could be attacked but, bit XOR operation is not vulnerable against a large number of power traces. As byte XOR operation can be attacked, another way to attack TRIVIUM is to attack the XOR operation between the keystream and the plaintext, and then solve equations backward to derive the secret key, if only TRIVIUM is reversible.

An isolated XOR byte operation was tested with a CPA attack and it successfully retrieved the output value of the XOR function. The same approach was applied to the ciphertext generation byte XOR operation. Experiments on the attack point revealed the vulnerabilities in the TRIVIUM algorithm as well as the compiler. As these findings led to a successful key stream attack on TRIVIUM, it is proven that TRIVIUM keystream is vulnerable to power analysis attacks. There are a few research articles (Maximov and Biryukov, 2007) which illustrates how knowing part of the keystream could be used to derive the secret key of TRIVIUM. Hence, this vulnerability can be used to derive the secret key as well.

A few more approaches can be taken to improve the project further. The noise in the power measurements can be reduced, and an attack can be tried on the bit-wise XOR operation in TRIVIUM. If this can be carried out successfully, the secret key of TRIVIUM can be obtained. Furthermore, this attack can be tried out on different testbeds which consist of different hardware components.

## Acknowledgements

University Research Grant URG/2018/19/E of University of Peradeniya, Sri Lanka.

## References

Atani, R.E., Mirzakuchaki, S., Atani, S.E., Atani, R.E., Mirzakuchaki, S., Atani, S.E. and Fhnw, W.M. (2008) 'Design and simulation of a DPA resistive circuit for TRIVIUM stream cipher based on SABL logic styles', *2008 15th International Conference on Mixed Design of Integrated Circuits and Systems*, pp.203–207.

Beaulieu, R., Shors, D., Smith, J., Treatman-Clark, S., Weeks, B. and Wingers, L. (2013) *The Simon and Speck Families of Lightweight Block Ciphers*, Cryptology ePrint Archive, Report 2013/404.

Brier, E., Clavier, C. and Olivier, F. (2004) 'Correlation power analysis with a leakage model', in *Cryptographic Hardware and Embedded Systems – CHES 2004*, pp.16–29.

Chakraborty, A. and Mukhopadhyay, D. (2016) 'A practical template attack on MICKEY-128 2.0 using PSO generated IVS and LS-SVM', in *29th International Conference on VLSI Design and 15th International Conference on Embedded Systems, VLSID*, Kolkata, India, 4–8 January, pp.529–534.

Chakraborty, A., Mazumdar, B. and Mukhopadhyay, D. (2015) 'A practical DPA on Grain v1 using LS-SVM', in *Proceedings of the 2015 IEEE International Symposium on Hardware-Oriented Security and Trust, HOST 2015*.

De Canniere, C. and Preneel, B. (2006) *TRIVIUM Specifications*, eSTREAM, ECRYPT Stream Cipher Project.

Dilshan, M. (2018) *Power-Analysis-Attack-on-TRIVIUM*, October [online] https://github.com/MalithaDilshan/Power-Analysis-Attack-on-TRIVIUM (accessed 10 November 2021).

Fischer, W., Gammel, B.M., Kniffler, O. and Velten, J. (2007) 'Differential power analysis of stream ciphers', in *Topics in Cryptology – CT-RSA 2007*, pp.257–270.

Gamaarachchi, H. and Ganegoda, H. (2018) 'Power analysis based side channel attack', *CoRR*, abs/1801.00932.

Gamaarachchi, H., Ganegoda, H. and Ragel, R. (2017) 'Breaking speck cryptosystem using correlation power analysis attack', *Journal of the National Science Foundation of Sri Lanka*, December, Vol. 45, p.393.

Gamaarachchi, H. (2015) *The A to Z of Building a Testbed for Power Analysis Attacks*, December [online] https://github.com/hasindu2008/PowerAnalysis.

Gierlichs, B., Batina, L., Clavier, C., Eisenbarth, T., Gouget, A., Handschuh, H., Kasper, T., Lemke-Rust, K., Mangard, S., Moradi, A. and Oswald, E. (2018) 'Susceptibility of estream candidates towards side channel analysis', May.

Jia, Y., Hu, Y., Wang, F. and Wang, H. (2012) 'Correlation power analysis of TRIVIUM', *Security and Communication Networks*, May, Vol. 5, pp.479–484.

Kocher, P., Jaffe, J. and Jun, B. (1999) 'Differential power analysis', in Wiener, M. (Ed.): *CRYPTO'99*, pp.388–397, Springer, Berlin, Heidelberg.

Liu, J., Gu, D. and Guo, Z. (2010) 'Correlation power analysis against stream cipher MICKEY v2', in *Proceedings – 2010 International Conference on Computational Intelligence and Security, CIS 2010*.

Lo, O., Buchanan, W.J. and Carson, D. (2017) 'Power analysis attacks on the AES-128 s-box using differential power analysis (DPA) and correlation power analysis (CPA)', *Journal of Cyber Security Technology*, Vol. 1, No. 2, pp.88–107.

Maximov, A. and Biryukov, A. (2007) 'Two trivial attacks on TRIVIUM', in Adams, C., Miri, A. and Wiener, M. (Eds.): *Selected Areas in Cryptography*, pp.36–55, Springer, Berlin, Heidelberg.

National Institute of Standards and Technology (1993) *FIPS 46-3: Data Encryption Standard*, March.

Rivest, R.L., Shamir, A. and Adleman, L. (1978) 'A method for obtaining digital signatures and public-key cryptosystems', *Commun. ACM*, February, Vol. 21, No. 2, pp.120–126.

Sandeep, S. and Rajesh, C.B. (2010) 'Differential power analysis on FPGA implementation of MICKEY 128', in *Proceedings – 2010 3rd IEEE International Conference on Computer Science and Information Technology, ICCSIT 2010*.

Sim, S.M., Jap, D. and Bhasin, S. (2020) 'DAPA: differential analysis aided power attack on (non-) linear feedback shift registers (extended version).', *Cryptology ePrint Archive*.

Tena-Sánchez, E. and Acosta, A.J. (2016) 'Optimized DPA attack on TRIVIUM stream cipher using correlation shape distinguishers', in *2015 Conference on Design of Circuits and Integrated Systems, DCIS 2015*.

Zadeh, A.A. and Heys, H.M. (2013) 'Theoretical simple power analysis of the grain stream cipher', in *26th IEEE Canadian Conference on Electrical and Computer Engineering CCECE 2013*, Regina, SK, Canada, 5–8 May, pp.1–5.