ALGEBRAIC SIDE CHANNEL ATTACK (ASCA) ON STREAM CIPHERS

By

Asif Raza Kazmi

A thesis submitted to the Faculty of Information Security Department, Military College of Signals, National University of Sciences and Technology, Pakistan in partial fulfillment of the requirements for the degree of Masters in Information Security

Aug 2017

ABSTRACT

Stream ciphers are known to have less complex hardware design and are generally faster than the block ciphers. Their employment is necessitated in systems where buffering is limited and bits must be processed immediately upon arrival or when data comes in unknown lengths like in wireless communication.

Till the advent of algebraic attacks on stream ciphers, a decade and a half ago, linear feedback shift register (LFSR) based stream ciphers were quite common due to their efficient implementation. However, later years saw a host of new stream ciphers based on nonlinear feedback shift registers (NLFSR), which were highly resistant against traditional algebraic cryptanalysis. At present the success of algebraic attacks against stream ciphers is only limited to lightweight stream ciphers or reduced variants of popular stream ciphers.

Side Channel Analysis (SCA) being an implementation attack, requires access to the hardware implementation of the target stream cipher to capture side channel leakages associated with internal processing. However to find exact secret key or internal state, a handsome quantity of leakage information is needed. In contrast to algebraic attacks, research work on SCA against stream ciphers has been less frequent but more successful.

Combining two cryptanalysis techniques can pay dividends not only in the form of achieving better attack complexities and facilitating exploitation of new vulnerabilities, but also lowering the effect of individual weaknesses of attack techniques. Algebraic attacks and SCA can both be strong candidates for such combination. Rather they are already being applied against block ciphers in unison for quite some time now. However no published work as of this writing can be found on combining algebraic cryptanalysis and SCA against stream ciphers.

In this work we propose to combine algebraic attacks with SCA, on stream ciphers, in a manner that reduces overall attack complexity/ difficulty as compared to isolated application of the two

constituent attacks. This combination attack, termed as *Algebraic Side Channel Attack (ASCA)*, utilize whatever side channel leakage is available from the target stream cipher's implementation in limited time exposure and converts it into algebraic equations. These SCA equations are added into the system of multivariate polynomial equations obtained from algebraic attack technique. Union of these equation sets, can then be resolved through any mathematical method/ tool, to find the unknown variables, i.e., either secret key or internal state of the stream cipher.

Stream ciphers of today, which mostly rely on nonlinearity by design to make them extremely immune to algebraic cryptanalysis, can also fall prey to ASCA. To demonstrate this, we successfully attack Crypto-1, Bivium-B, Trivium and Grain stream ciphers using our proposed technique in 0.158, 11.531, 21.54 and 28.25 seconds respectively.

DEDICATION

To my loving family

ACKNOWLEDGMENT

I am grateful to Allah Almighty, the most beneficent and all compassionate for always blessing me magnanimously. He, being the spring-head of entire knowledge, relieved me with subtle and sometimes clear guidance, when I was stuck up, during the course of this thesis.

I acknowledge the invaluable opportunity provided to me by Pakistan Army in general and my Directorate in particular, to pursue post graduate studies.

I my highly indebted to my supervisor Dr. Mehreen Afzal, firstly for teaching us cryptology from scratch in such a methodological fashion that it felt like *abc*, then for gradually preparing us for research work during coursework and finally for regularly making her available for supervising my thesis, in spite of her tight schedule.

I appreciate faculty of department of information security at my university, for providing us a promising learning environment and teaching other necessary subjects related to information security, aptly. Especially, I would like to express my gratitude to Dr Muhammad Faisal Amjad for nurturing understanding of research methodology among us and Ms Narmeen Shafqat for teaching us some valuable hands-on skills in the realm of ethical hacking.

I am also thankful to all researchers of Cryptology whom work I have gone through and reused, as without their supporting work, I could not have completed my research so comfortably.

TABLE OF CONTENTS

LIST OF FIGURES

LSIT OF TABLES

Chapter 1

INTRODUCTION

1.1 Introduction

This chapters deals with the basic terminologies and concepts which are necessary to understand further work presented in the thesis. A brief overview of stream ciphers, their design, common types and major categories of attacks against them are concisely discussed. Being overall introductory chapter of thesis, it also defines the scope of problem at hand, our intended objectives while undergoing this research and how the content of this thesis is organised into various chapters.

1.2 Stream Ciphers

Stream ciphers generate a keystream of a reasonably long period which XORs with plaintext bit by bit to generate ciphertext. Same plaintext if encrypted again would transform into different ciphertext generally, due to the randomness of keystream. However, keystream is repeated after a period of time. Stream ciphers are known to be faster and less complex in implementation as compared to block ciphers.

1.3 Design of Stream Ciphers

Stream ciphers have two major parts i.e. state update function and output function. During operation the state of the stream cipher is changing with time. The output function generates the keystream bits whose values are dependent upon the prevailing internal state. The keystream bits

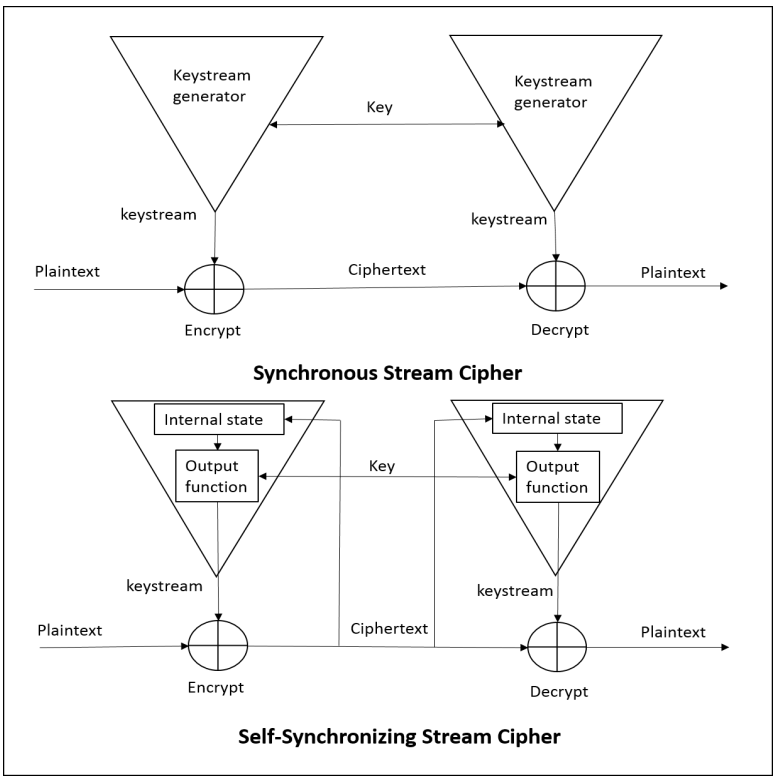

Figure 1.1: Types of Stream Ciphers [1]

are XORed with the plaintext to produce ciphertext which is transmitted and same keystream bits are then XORed with the ciphertext at the receiving end to produce plaintext. Whole process of encryption and decryption occurs bit by bit serially. The initial state of the cipher can be generated by a key setup process through a secret key. Same secret key can be used with different initialization vector (IV) and the process is called resynchronization.

A *synchronous* stream cipher generates the keystream independent of plaintext and ciphertext because it needs synchronization between sender and receiver. Whereas a *self-synchronizing* or *asynchronous* stream cipher generates the keystream which is dependent on key and a few previous ciphertext bits. Figure 1.1 illustrates the difference between synchronous and self-synchronizing stream ciphers.

The security of stream cipher based encryption lies in the randomness of the keystream. Claude Shannon termed one-time pad (OTP) as perfectly secure because its key was as long as the mes-

sage and it never repeated [3]. Real world stream ciphers do not have infinitely long periods but they should have sufficiently long period. A proven method for having sufficiently long period is to design stream ciphers based on linear feedback shift registers (LFSR) with update function based on primitive polynomial. The problem associated with linearity has to be tackled by some means such as irregular clocking. Better and popular approach is to design stream ciphers based on nonlinear feedback shift registers (NLFSR).

1.4 Cryptanalysis of Stream Ciphers

Design of any cipher takes care of the prevalent cryptanalysis techniques. Similarly new attack techniques are also introduced while keeping in view the existing ones. Therefore it is necessary to have a cursory glance on the cryptanalysis techniques already introduced against stream ciphers, before getting deeper into our topic.

Few important attacks of generic nature on the stream ciphers are brute force attack, divide and conquer attack, time-memory-data trade-off attack, resynchronization attack and distinguishing attack etc. *Brute force attack* is also known as exhaustive key search. It can be an attack of most fundamental nature against any cipher, where the key is searched through applying all possible values in the keyspace. Therefore, while designing a cipher, key length is kept as per the dictates of prevalent computing power and security of the system. *Divide and conquer attack* is a well known technique to resolve complex problem and is equally applicable to ciphers. Shannon concept of adding confusion and diffusion [3], can act as a counter measure against this attack strategy. A stream cipher may well be prone to divide and conquer attack, when a portion of its state is updated independently of other components [4], e.g stream cipher based on T-function [5]. In *Time-memory-data trade-off attack* [6], for numerous different states, keystream chunks are calculated beforehand and placed in a repository. Fragments of actual keystream are compared

with the stored chunks for similarity to reach to the state of the cipher. *Resynchronization attacks* are directed at the setup of stream cipher through key and IV. Key/IV setup based on linear function has proved to be highly vulnerable. To guard against resynchronization attack, there should be sufficient confusion and diffusion incorporated into the key/IV setup [4]. In a *distinguishing attack* against stream cipher, its keystream is proved to be biased rather than being a random sequence. Bias in the keystream can be exploited to recover the plaintext.

LFSR based stream ciphers are very popular because of their simple and efficient design. These stream ciphers have been subjected to numerous kind of attacks. *Berlkamp-Massey algorithm* [7], can find the linear complexity of a sequence. This can be effectively used to reach at the state of LFSR based stream cipher through sufficient number of captured keystream bits. *Correlation attacks* exploit the correlation between the LFSR bits and the bits of keystream [8]. These attacks were further improved and renamed as *fast correlation attacks* in [9]. A lot of research work was carried out on these attacks and many refinements were published in later years. Filter and combiner functions with least possible correlation between input and output bits can be used to safeguard against efficacy of fast correlation attacks [4]. *Algebraic attacks* are employed to transform stream ciphers into a system of multivariate equations, which is mathematically resolved to find out the internal state or secret key. Algebraic attack, being constituent of ASCA, is further explained in chapter 3.

Side Channel Analysis (SCA) is yet another potent threat against stream ciphers. These attacks are directed against the implementation of a cipher in hardware, regardless of its algorithmic strength/ weakness. Even ciphers considered immune to other cryptanalysis attacks in theory can fall prey to side channel attacks when implemented due to side channel leakage. Side channel leakage information from the hardware implementation can be of various forms such as power consumption

or timing variations in performing different operations 3.2. SCA being part of ASCA is explained in more detail in chapter 3. Other than passive SCA which does not make any contact with the implementation device, active SCA, also known as fault injection attack, forces a malfunction into the operation of the device through over-clocking, subjecting it to high temperature etc., while capturing side channel leakage.

1.5 Background

Algebraic cryptanalysis against stream ciphers has lost its vitality. Though it is highly successful against LFSR based stream ciphers, but modern stream ciphers which are based on nonlinear update functions are no more troubled by algebraic attacks. Due to built-in nonlinearity, the solving of multivariate quadratic polynomial (MQ) problem associated with such stream ciphers is computationally infeasible. SCA against stream ciphers, though not very common, but is still viable. Side channel leakage from the implementation of ciphers can be related with their internal processing to reveal useful information. Any amount of captured side channel information, may not be good enough for a full fledged SCA, can be converted into algebraic equations and augment already obtained system of equations through algebraic attack. This would reduce overall complexity of MQ problem.

1.6 Problem Statement

While keeping the computational infeasibility of algebraic attacks against stream ciphers based on nonlinear update functions and high probability of capturing some (if not much) useful side channel leakage from cipher's implementation, the idea of adding partial side channel leakage information into multivariate equations, acquired from algebraic transformation of target stream cipher, seems quite promising. This leads to the problem statement of our research work as quoted

below:-

*Can **complexity of MQ problem** generated from **Algebraic Cryptanalysis** against **Stream Ciphers**, be reduced by adding only **partial Side Channel Leakage** information?*

1.7 Objectives

The research work presented in thesis has been carried out with following objectives in mind:-

- Reviewing ASCA on block ciphers.
- Proposing novel idea of ASCA on stream ciphers and devising a generic attack methodology.
- Presenting concept of ASCA on stream ciphers by the help of its application on a simple hypothetical cipher.
- Applying ASCA on a comparatively simple but real world stream cipher as proof of concept. We chose Crypto-1 stream cipher.
- Demonstrating the efficiency of ASCA by applying it on Trivium and Grain stream ciphers.
- Preparation of software simulation for target stream ciphers as mentioned above for experimentation/ application of ASCA.
- Exploring available automated tools and picking relevant ones for obtaining results efficiently. Grain-of-salt for transforming stream ciphers into CNF clauses and CryptoMiniSAT 5.0 for satisfiability/ assignment of Boolean satisfiability (SAT) problem, were used for this work.

1.8 Organisation

For ease of readership and better comprehension, this thesis is organised in logical segmentations as under.

Chapter 1 in the first half touches very basics of stream cipher design and cryptanalytic techniques employed against them and then in other half, prepared the reader for upcoming work by defining scope of research, listing intended objectives and giving organization of the thesis.

Chapter 2 reviews research work on ASCA against block ciphers which is pertinent to understand application of ASCA against stream ciphers. Initial research paper which introduced ASCA on block ciphers is reviewed. A strong form of SCA, called template attacks, is also elucidated while referring to the introductory research work on these attacks. Moreover examples of ASCA on small S-boxes are explained step by step.

Our proposed concept of ASCA on stream ciphers is explained in chapter 3. A generic attack methodology is presented to mount ASCA on any stream cipher. The same methodology is applied against a hypothetical stream cipher. Then in a bid to test our attack methodology, a simple real world cipher based on a nonlinear combiner function, Crypto-1, is subjected to ASCA as proof of concept.

In chapter 4 two popular stream ciphers Trivium and Grain are chosen for mounting ASCA against them. Both of these ciphers are selected for the reason that they have never been successfully attacked employing pure algebraic attack technique. Therefore work/ experimentation presented in this chapter acts as a litmus test for our proposed attack.

Lastly in chapter 5 a concise analysis of the whole work is presented statistically along with concluding remarks and future work possibilities.

1.9 Conclusion

This chapter presented a brief introduction to the thesis as a whole. Firstly a short background theory to the research work is discussed as a preface to the main work. Secondly the problem statement and objectives are highlighted which form the basis of this research. Lastly, a bird-eye view of complete work divided into various chapters is appended.

Chapter 2

RELATED WORK

2.1 Introduction

The work undertaken in this thesis can be related to research work on ASCA on block ciphers. Therefore in this chapter the concept of combining algebraic and side channel attacks on block ciphers is reviewed with special emphasis on first attack of its kind on PRESENT cipher. Moreover this chapter also presents a brief review of template-like SCA, which is a strong form of SCA, useful in mounting ASCA. ASCA can be understood as an advance form of algebraic attack which also uses partial side channel information from the target cipher's implementation, thereby reducing the overall complexity of solving. Towards the end this chapter also includes step-by-step examples of application of ASCA on small S-boxes.

2.2 Diminishing Success of Algebraic Cryptanalysis

Algebraic attacks first transform the target cipher into a system of multivariate equations in an offline phase. Then in an online phase this multivariate quadratic equation (MQ) problem is linearised and resolved through some mathematical technique, such as relinearisation [10], extended linearisation [11], sparse extended linearisation [12], Groebner bases [13] or SAT solving, to find the unknown key.

Mathematically, any system of linear equations can be solved correctly if number of equations are more than unknowns. Such system of equations is called overdefined. Coefficient matrices of a

system of linear equations may have many zero elements. Matrices obtained after removing zero elements are called sparse matrices. Complexities of solving sparse systems is further reduced as compared to the starting set of equations. Though ciphers can be transformed into set of overdefined and sparse system of equations but still due to huge number of equations and unknowns, it is extremely difficult to resolve MQ problem or SAT problem through pure algebraic cryptanalysis technique. As a result success of algebraic attacks is mostly limited to either lightweight ciphers or reduced variants of popular ciphers.

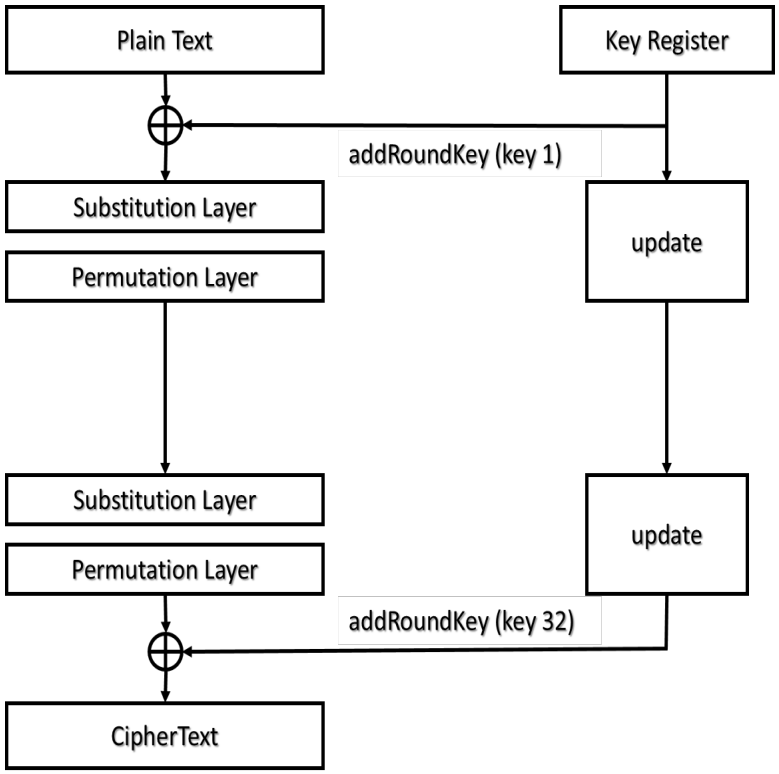

Figure 2.1: PRESENT Block Cipher [2]

2.3 Advent of ASCA on Block Ciphers

Algebraic Side Channel Attack (ASCA) against block ciphers was introduced in 2009 by Renauld and Standaert [14]. They proposed to incorporate partial side channel leakage information from a cipher's implementation into the system of equations obtained through algebraic attack. Traditional differential power analysis aims to recover all target bits, whereas SCA part in ASCA aims

at simpler targets, like knowledge of hamming weights or hamming distances at specific points of cipher's implementation, in few (often only one) trace measurements.

To demonstrate their novel idea, authors applied ASCA against PRESENT block cipher. PRESENT cipher (fig 2.1) [2], is based on substitution-permutation network, comprising 31 rounds, with 64-bit block size and 80/ 128-bit key. In each round of PRESENT, 64-bit block is passed through 16 4x4 Sboxes in parallel. Firstly, authors subjected PRESENT cipher to a traditional algebraic attack, where it was transformed into a system of 40,000 algebraic equations comprising 7,000 variables and 50,000 monomials. Then this algebraic problem was converted

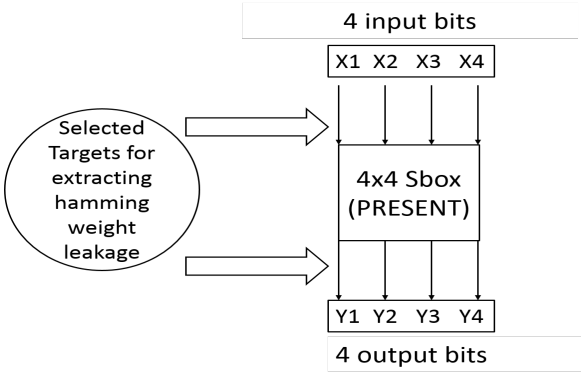

Figure 2.2: Side Channel Leakage Points - PRESENT

into a SAT problem, by representing all algebraic equations in conjunctive normal form (CNF). Secondly, authors got hold of a PRESENT implementation in a PIC 16F877 8-bit micro-controller which leaks power traces with strong correlation to hamming weights of data being manipulated. They employed Baysian template attack technique to obtain hamming weight of data commuting on bus, in a single or few trace measurements, as explained by Chari et al. in [15]. Template attacks work on an assumption that the adversary possesses a replica of a device on which target cipher is implemented and adversary can derive noise characterization of the said cipher precisely. By modelling noise, all available information can be fully extracted in a single measurement sample. Template attacks are further explained in section 2.4 ahead.

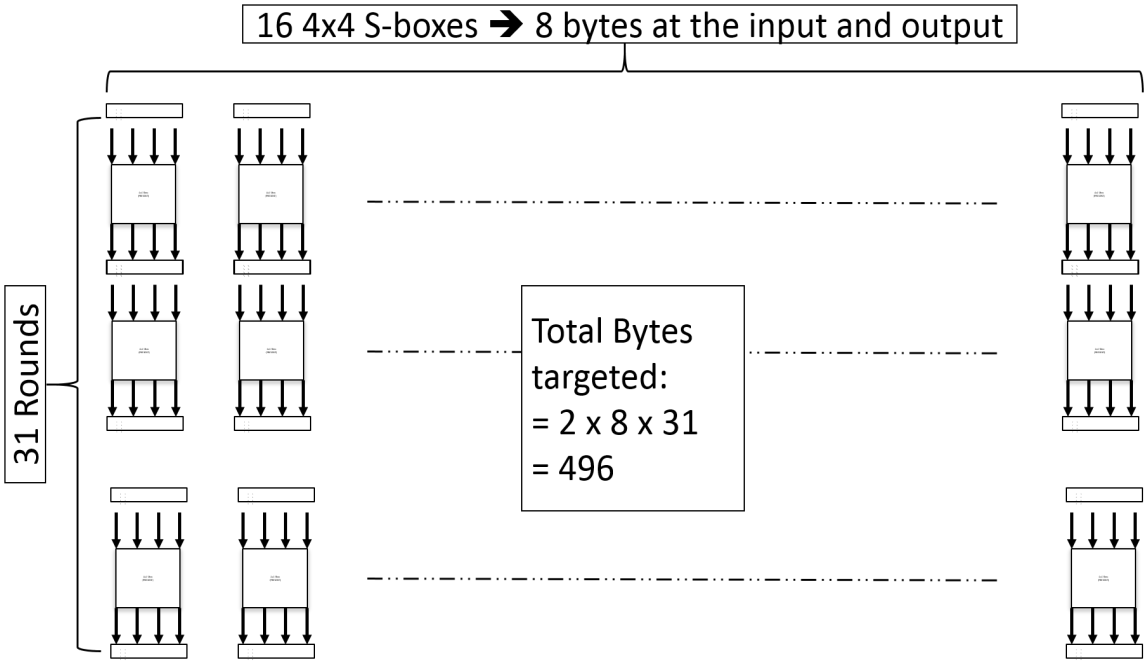

Figure 2.3: Hamming Weights of 496 bytes acquired through SCA - PRESENT

Authors selected the input and output bits of all the S-boxes for their partial SCA on PRESENT (fig 2.2). For a complete 31 round PRESENT, with 16 S-boxes in each round, hamming weights of a total of 496 bytes were acquired (fig 2.3). As per authors, all these hamming weights were recovered in a single encryption step with probability 0.993^2. Thirdly, the additional information obtained through partial side channel attack was added into the SAT problem as CNF clauses. Consequently, authors were able to solve the SAT problem with 100% success rate, in about 2.5 seconds for a full PRESENT cipher.

2.4 Template Attacks

Template attacks were introduced by Chari et.al. in their iconic research paper in 2002 [15]. These attacks were coined as strongest form of SCA because of their success without the need for continued access of the cipher's implementation. There are two main peculiarities of these SCA attacks:-

- Instead of trying to eliminate noise in the captured power traces, it models noise to extract useful information.
- Only one or few captured power traces, from actual implementation device, are enough.
- Adversary possesses a device identical to the actual implementation device.

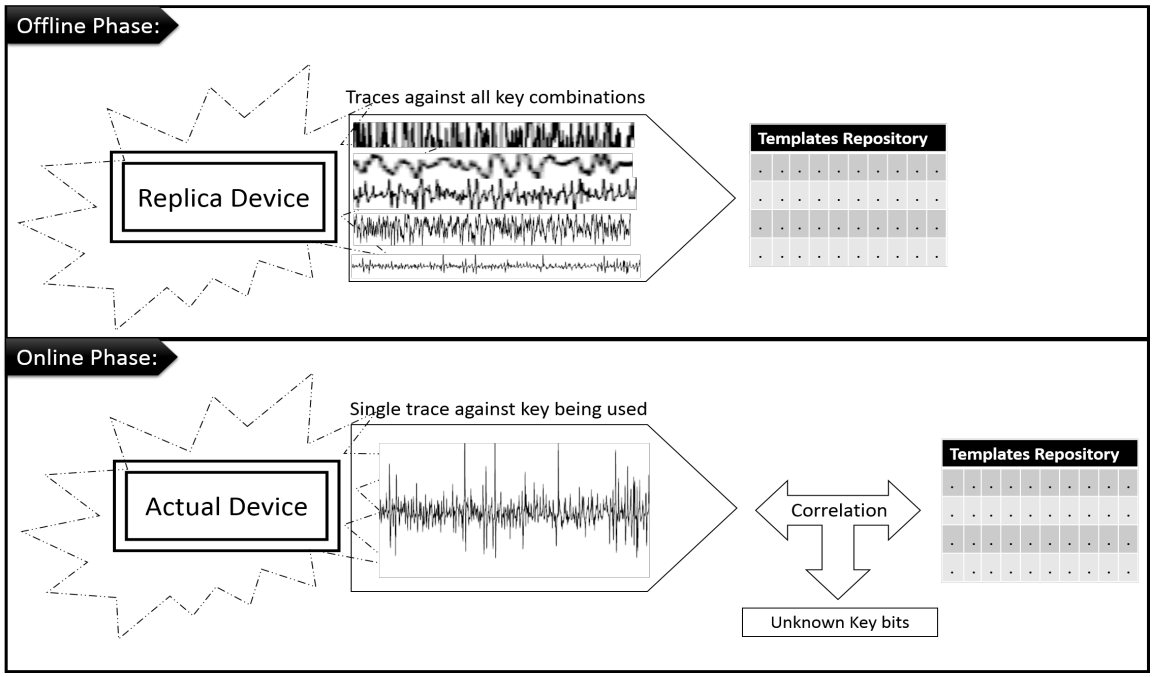

Figure 2.4: Template Attacks

To start with the attack (fig 2.4), adversary acquires a programmable replica of the actual device hosting the target cipher. Though this assumption may seem impractical but it must be noted that use of standardized microcontrollers is quite common and its not difficult to find out which particular microcontroller is being made use of. The replica experimental device is used not only for capturing power traces of all key combinations, but also the associated noise characterization of the leaked signals precisely. This captured information against each key combination, called template, is recorded during the experimentation or offline phase.

Next the adversary needs access to the actual device for capture of a single or few power traces. This can be called online phase of template attacks. This actual trace is compared with the

recorded samples or templates to find a match. In this way the key is found out. Practically, a large key may be attacked through an *extend and prune* strategy to go step by step towards the complete key.

2.5 ASCA on Block Ciphers - Trivial Examples

In this section we explain the concept of ASCA on block ciphers with trivial examples. A 2x2 S-box and then a 3x3 S-box is subjected to ASCA in detail. S-box, being nonlinear part of block ciphers, is highly resistant to lone algebraic attacks because it transforms into high degree equations. Therefore application of ASCA on simple S-boxes can prove its efficacy.

Input		**Output**	
x_0	x_1	y_0	y_1
0	0	0	1
0	1	1	0
1	0	1	1
1	1	0	0

Table 2.1: Truth Table - 2x2 S-box in Example-1

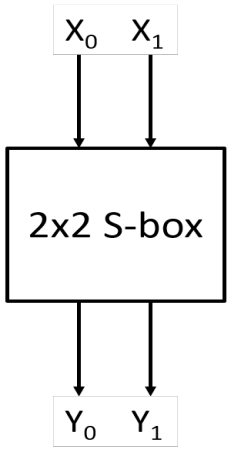

Figure 2.5: A 2x2 S-box

2.5.1 ASCA on 2x2 S-box

A 2x2 S-box is displayed in fig 2.5 and its truth table is shown in table 2.1. Step by step explanation of ASCA on this S-box is given below:-

- 2-degree monomials are obtained from input and output variables of S-box.
- A matrix ordered $M * 2^n$ is generated as shown in table 2.2 where M is equal to number of monomials and n is equal to number of input bits of S-box.

Table 2.2: $M * 2^n$ Matrix for 2x2 S-Box in Example-1

Monomial	**IO-1**	**IO-2**	**IO-3**	**IO-4**
1	1	1	1	1
x_0	0	0	1	1
x_1	0	1	0	1
y_0	0	1	1	0
y_1	1	0	1	0
$x_0.y_0$	0	0	1	0
$x_0.y_1$	0	0	1	0
$x_1.y_0$	0	1	0	0
$x_1.y_1$	0	0	0	0

- Above matrix is transformed into a row-echlon matrix. Number of zero rows give the number of equations for this particular S-box. Following equations are obtained resultantly:-

  $1 + x_0 + y_1 + x_0y_0 + x_1y_0 = 0$

  $x_0 + x_0y_0 + x_1 + x_1y_0 = 0$

  $y_0 + x_0y_0 + x_1y_0 = 0$

  $x_0y_0 + x_0y_1 = 0$

  $x_1y_1 = 0$

- More equations are added through partial side channel information captured from implementation of the cipher, as demonstrated in ensuing points.

- Suppose attacker captures power leakage at the input and output of this S-box and is able to find corresponding hamming weights as $HW(input) = 1$ and $HW(output) = 2$ as shown in fig 2.6.

- Hamming weight information obtained as above is also converted into equations:

  - For $HW(input) = 1 : x_0x_1 = 0, x_0 \oplus x_1 = 1$
  - For $HW(output) = 2 : y_0 = y_1 = 1$

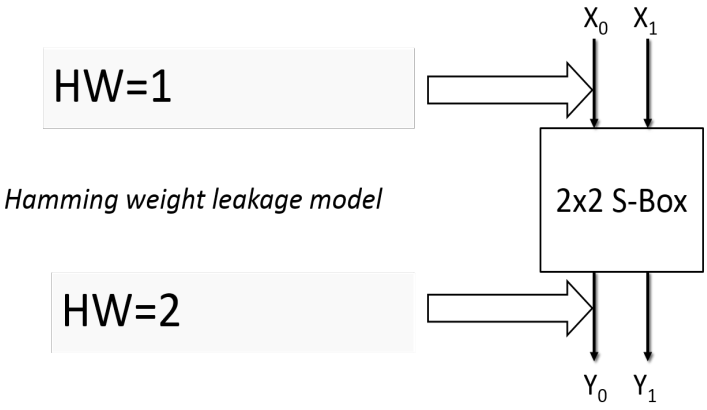

Figure 2.6: Capturing Hamming Weight Leakage - Example-1

- When all equations from algebraic attack and SCA are added together, they can be trivially solved to give values to unknown: $x_0 = 0, x_1 = 1, x_2 = 0, y_0 = 0, y_1 = 0, y_2 = 0$

Input			Output		
x_0	x_1	x_2	y_0	y_1	y_2
0	0	0	1	0	1
0	0	1	0	1	1
0	1	0	0	0	0
0	1	1	1	0	0
1	0	0	1	1	1
1	0	1	0	1	0
1	1	0	1	1	0
1	1	1	0	0	1

Table 2.3: Truth Table - 3x3 S-box in Example-2

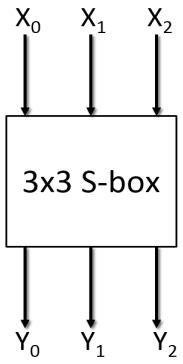

Figure 2.7: A 3x3 S-box

2.5.2 ASCA on 3x3 S-box

A 3x3 S-box is displayed in fig 2.7 and its truth table is shown in table 2.3. For mounting an ASCA, S-box is transformed into algebraic equations through traditional algebraic attack technique and then some more algebraic equations are obtained from side channel leakage information. The steps are briefly explained below:-

- 2-degree monomials are obtained from input and output variables of S-box.

- A matrix ordered $M * 2^n$ is generated as shown in table 2.4 where M is equal to number of monomials and n is equal to number of input bits of S-box.

Table 2.4: $M * 2^n$ Matrix for 3x3 S-Box in Example-2

Monomial	**IO-1**	**IO-2**	**IO-3**	**IO-4**	**IO-5**	**IO-6**	**IO-7**	**IO-8**
1	1	1	1	1	1	1	1	1
x_0	0	0	0	0	1	1	1	1
x_1	0	0	1	1	0	0	1	1
x_2	0	1	0	1	0	1	0	1
y_0	1	0	0	1	1	0	1	0
y_1	0	1	0	0	1	1	1	0
y_2	1	1	0	0	1	0	0	1
$x_0.y_0$	0	0	0	0	1	0	1	0
$x_0.y_1$	0	0	0	0	1	1	1	0
$x_0.y_2$	0	0	0	0	1	0	0	1
$x_1.y_0$	0	0	0	1	0	0	1	0
$x_1.y_1$	0	0	0	0	0	0	1	0
$x_1.y_2$	0	0	0	0	0	0	0	1
$x_2.y_0$	0	0	0	1	0	0	0	0
$x_2.y_1$	0	1	0	0	0	1	0	0
$x_2.y_2$	0	1	0	0	0	0	0	1

- Above matrix is transformed into a row-echlon matrix. Number of zero rows give the number of equations for this particular S-box. Following equations are obtained resultantly:-

$x_2y_2 + y_1 + x_0 = 0$

$x_2y_1 + x_0y_0 + y_1 = 0$

$x_2y_0 + x_1y_1 + x_1y_0 = 0$

$x_1y_2 + x_0y_1 + x_0 = 0$

$x_1y_1 + x_0y_2 + x_0y_1 + x_0y_0 + x_0 = 0$

$x_1y_0 + y_2 + y_1 + y_0 + x_0 = 0$

$x_0y_2 + y_2 + y_0 + x_2 + x_0 = 0$

$x_0y_0 + y_2 + x_1 + x_0 + 1 = 0$

- For an isolated algebraic attack, above mentioned equations are to be solved by any appropriate mathematical technique to find unknown variables. For real world ciphers this is an enormous task. Here for ASCA, more equations are added into this through partial side channel information captured from implementation of the cipher, as demonstrated in ensuing points.

- Suppose attacker captures power leakage at the input and output of this S-box and is able to find corresponding hamming weights as $HW(input) = 1$ and $HW(output) = 0$ as shown in fig 2.8.

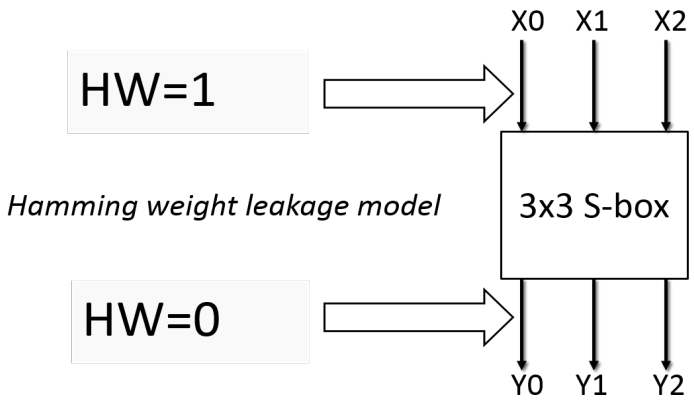

Figure 2.8: Capturing Hamming Weight Leakage - Example-2

- Hamming weight information obtained as above is also converted into equations:
  - For $HW(input) = 1 : x_0x_1 = x_0x_2 = x_1x_3 = 0$ and $x_0 \oplus x_1 \oplus x_2 = 1$
  - For $HW(output) = 0 : y_0 = y_1 = y_2 = 0$

- When all equations from algebraic attack and SCA are added together, they can be trivially solved to give values to unknown: $x_0 = 1, x_1 = 0, y_0 = 1, y_1 = 1$

2.6 Present Status of ASCA on Block Ciphers

From 2009 onwards, ASCA against block ciphers drew significant interest of researchers [16], [17], [18], [19], [20], [21], [22], [23]. ASCA revamped traditional algebraic cryptanalysis on block ciphers, whose successes were confined to toy ciphers and round-reduced variants of popular ciphers.

From 2009 onwards, ASCA against block ciphers drew significant interest of researchers, especially in the direction of its error-tolerance. In [16], authors applied ASCA on AES and highlighted that most of the notions that exist for PRESENT cipher can be noted against unprotected implementation of AES on 8-bit micro-controller. They recovered AES key after observing a single encryption operation through ASCA. In [17], authors emphasized that errors in side channel information to be incorporated in ASCA, makes the SAT problem unsatisfiable, therefore they recommended to use pseudo Boolean optimizers (PBOPT) instead of SAT solvers for ASCA in presence of errors. Optimizers also take into account some additional logical constraints while solving. Another error tolerant technique to deal with inaccurate side channel measurements in ASCA, known as multiple deductions-based ASCA (MDASCA) was introduced in [18]. In [19], authors presented a fresh notion of algebraic immunity for designing ASCA resistant S-boxes. Authors in [20] presented an improved ASCA on AES. In [21], in addition to comparing optimizers and solvers in terms of robustness and speed, authors proposed to look for leakage models other than hamming weight for applying ASCA. In yet another ASCA on AES, using optimizer, authors in [22] established error rate threshold that can be tolerated. They claimed to recover AES key in 10 hours in presence of 20% error rate from 100 measurements on the average. In [24] authors attacked AES key in a template attack combined with ASCA in exceedingly restricted access to implementation device. Further work on error tolerant ASCA in [25] used constraint

programming compiler called BEE (Ben-Gurion University Equi-propagation Encoder.) Authors in [26], while attacking AES, added side channel information along with measurement noise into equation set obtained through algebraic attack and solved it as pseudo Boolean optimization instance.

2.7 Conclusion

In this chapter a succinct view of ASCA on block cipher is given. Introductory paper on ASCA by Renauld et. al [14] has been briefly reviewed. A strong form of SCA, template attacks, has also been reviewed, as the concept of template attacks is central to the ASCA methodology. Moreover to build a thorough understanding of how ASCA works, trivial examples of its application onto 2x2 and 3x3 S-boxes have been explained.

Chapter 3

PROPOSING ASCA ON STREAM CIPHERS

3.1 Introduction

We have observed in the previous chapter that algebraic and side channel attacks are successfully being applied on block ciphers in combination since 2009. On stream ciphers, both these attacks are being applied in isolation as yet. Algebraic attacks against stream ciphers with nonlinear update are not computationally feasible. The complexity of algebraic attacks can be considerably reduced by adding partial information from SCA as the system of equations is made further over-defined. In this chapter, we explain our proposition that algebraic and side channel attacks can be mounted on stream ciphers in unison. A generic attack methodology for the proposed attack is discussed and then applied against a hypothetical stream cipher with nonlinear combiner function. As a proof of concept ASCA is applied on Crypto-1 Stream Cipher in the end.

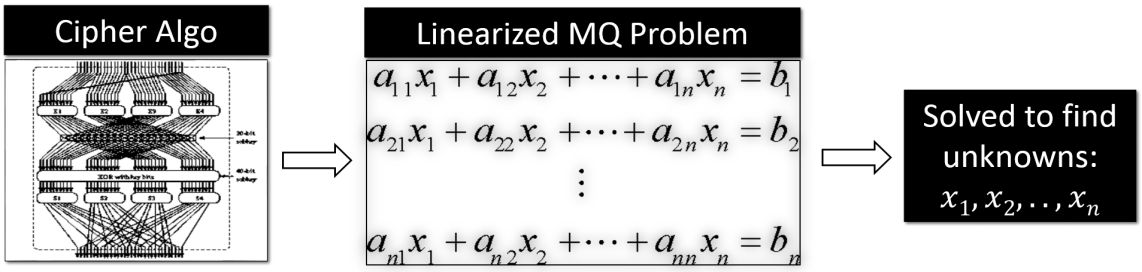

Figure 3.1: Algebraic Attack

3.2 Algebraic Attacks on Stream Ciphers

LFSR based stream ciphers can also be conveniently transformed into set of algebraic equations relating the state bits and the keystream bits as shown in fig 3.1. This set of over-defined equa-

tions can be solved through methods like Linearization, relinearization [10], extended linearization [11], sparse extended linearization [12], Groebner bases [13] and Boolean satisfiability (SAT) solving etc. Algebraic attacks against LFSR based stream ciphers received a lot of attention after their successful application against Toyocrypt [27] and LILI 128 [28].

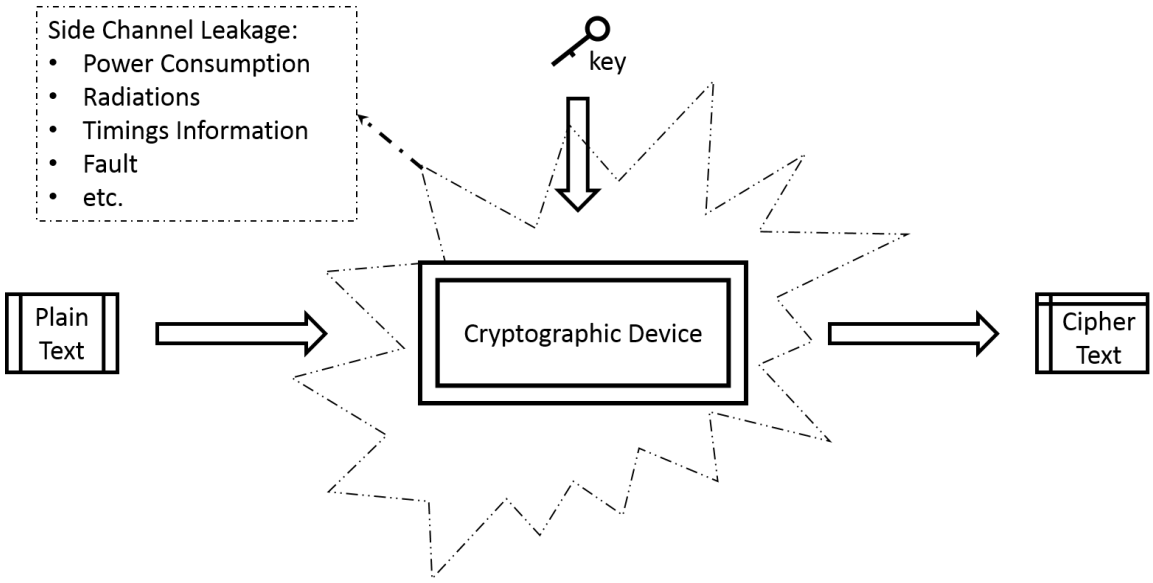

Figure 3.2: Side Channel Leakage from Cryptographic Device during Operation

3.3 SCA on Stream Ciphers

SCA is an implementation attack, which extracts information about secret key or internal states of stream ciphers from side channel leakage (fig 3.2). For capturing any type of side channel leakage, knowledge of the algorithm is not required at all, however for better consumption of that information it can pay dividends. In the ensuing paragraphs we have briefly discussed various types/forms of passive side channel attacks with reference to stream ciphers.

3.3.1 Timing Analysis

Timing analysis is kind of passive SCA where relation of timing variations in completing various operations is established with the data being processed. It was introduced in [29] by Kocher in

1996.

3.3.2 Power Analysis

Cryptographic devices consume static (data-independent) and dynamic (data dependent) power as under [30]:

$P_{avg} = P_{leakage} + P_{switching} + P_{short-circuit}$

where $P_{leakage}$ is static part and $P_{switching} + P_{short-circuit}$ is dynamic part.

For power analysis, only dynamic part, being related to the data processing, is of interest to the adversary. The static part remains constant and carries no information.

Before moving on to the types of power analysis, let us have a bird-eye view of power models.

3.3.2.1 Power Models

Power models help us describe and understand the relation of processed data with the power consumption. Hamming weight and hamming distance models are two common models which are also important with regards to deeper understanding of this thesis.

- Hamming weight model: In this model a relationship between hamming weight of the processed data is established with the power consumption.

- Hamming distance model: This models tends to relate power consumption with the number of transitions i.e. $0 \rightarrow 1$ and $1 \rightarrow 0$. Shift registers are generally implemented through flip-flops. When state of a flip-flop remains same, power consumed is only static. On the other hand when state of flip-flop is changed the power consumption increases.

3.3.2.2 Simple Power Analysis (SPA)

In SPA, only few (or just one) power traces are captured to extract useful information. Knowledge of algorithm is normally utilized for making SPA a success.

3.3.2.3 Differential Power Analysis (DPA)

DPA makes use of a large number of power traces to reach at the key. Normally DPA is attempted under a known plaintext or known ciphertext scenario. Large number of traces make it possible to extract information under noisy conditions as well.

3.3.3 Template Attacks

Template-like attack is a very strong variant of SCA, which has been amply covered in chapter 2 while reviewing iconic paper of Chari et. al. [15].

3.4 Motivation

While taking a lead from successful work on ASCA against block ciphers, we got motivation for proposing ASCA against stream ciphers due to following reasons:

- Firstly, that lone algebraic attacks are not much successful against popular stream ciphers of today and
- Secondly that there exists sufficient vulnerability of stream ciphers to SCA, which can be efficiently utilized for ASCA.

So the items above logically lead us to the proposal of combining algebraic cryptanalysis with SCA to bring a stronger form of attack onto stream ciphers as shown in fig 3.3, known as ASCA,

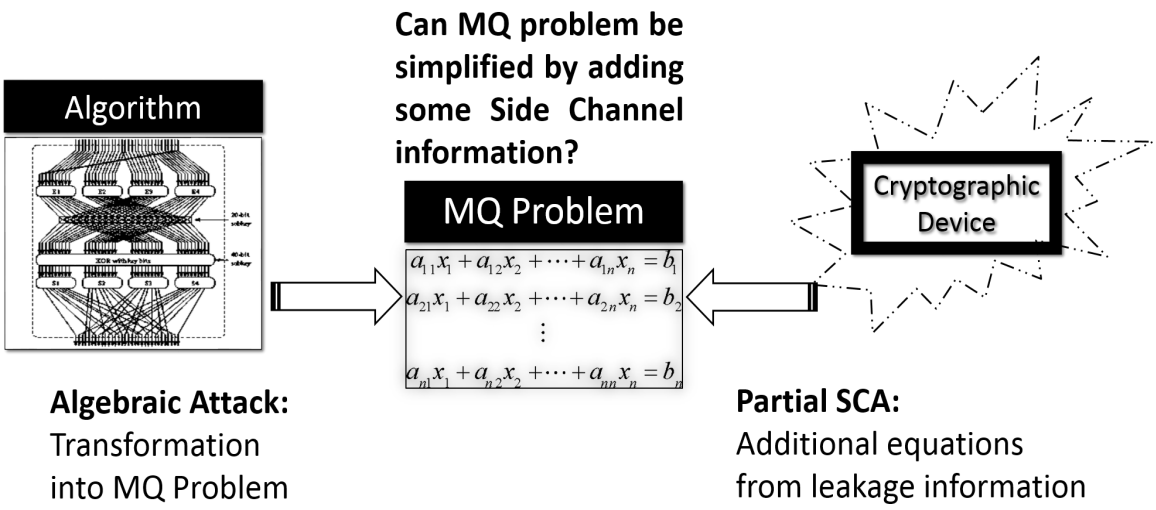

Figure 3.3: Problem Statement Manifest

which has been a proven success against block ciphers.

Further explanation of the a/m points is appended in ensuing paragraphs.

3.4.1 Limited Success of Algebraic Attacks against Stream Ciphers

Traditional algebraic attacks against popular stream ciphers are computationally as complex as those against block ciphers. Algebraic attacks on stream ciphers, introduced by Courtois and Meier in 2003 [31], attempt to solve a system of multivariate polynomial equations, obtained from relation of internal state bits and few output stream bits. This leads to efficient retrieval of internal state bits or secret key subsequently, in case of low degree equations. Courtois also proved that there existed low degree annihilator functions of Boolean function $f()$ and $1 + f()$, thereby reducing overall complexity of algebraic attack. Therefore, lot of research was made in algebraic attacks against stream ciphers in that era [32], [33], [34], [28], [12]. However, later, nonlinearity in the construction of stream ciphers by employing NLFSRs or nonlinear combiner functions became a common feature thereby greatly increasing the degree of generated equations and decreasing the computational feasibility of algebraic attacks against them. Therefore stream

Table 3.1: Susceptibility of eSTREAM Ciphers towards SCA

Cipher	Exploitable Timing Vulnerability	Exploitable Conditional Branching	Exploitable HW Leakage of Data	Exploitable DPA Vulnerability	DPA Attack Complexity
CryptMT	No	No	Yes	Yes	Medium
Dragon	Yes	No	Yes	Yes	Low
HC	May be	No	Yes	Yes	Low
LEX	Yes	May be	Yes	Yes	Low
NLS	Yes	No	Yes	Yes	High
Rabbit	No	No	Yes	Yes	Medium
Salsa20	No	No	Yes	Yes	Low
SOSEMANUK	May be	No	Yes	Yes	Low
Decim	May be	Yes	Yes	Yes	High
F-FCSR	No	No	Yes	Yes	Medium
Grain	No	No	Yes	Yes	Medium
MICKEY	Yes	Yes	Yes	Yes	Medium
Moustique	No	No	Yes	Yes	Medium
Trivium	No	No	Yes	Yes	Medium

ciphers like Sfinks [35] and WG [36] based on LFSRs were dropped in 2nd and 3rd phase of Project eSTREAM [37] respectively. NFLSR based stream ciphers proved to be highly resistant to algebraic attacks because the degree of underlying algebraic equations increased with each clocking of registers, for instance, after about 80 equations of Gain-1 stream cipher obtained through algebraic attack, the degree of equations rose to as high as 160 [38]. Moreover clock controlled ciphers such as A5/1 also possessed impressive resistance against algebraic cryptanalysis [39]. Successful attacks against modern stream ciphers, therefore resort to guessing few bits and in some cases specific guessed bits lead to better results [40], [41].

3.4.2 Susceptibility of Stream Ciphers towards SCA

Side channel attacks, though, have not been as common as algebraic attacks against stream ciphers [42], [43], yet have been more successful. In [42] and [44] authors have mounted a successful DPA against both Trivium and Grain. In [15], authors demonstrated the concept of

template-attack by successfully applying it against RC4 stream cipher. In [45], Benedikt Gierlichs et al. have comprehensively evaluated the susceptibility of eSTREAM ciphers towards SCA while taking into account conventional implementation techniques, leakage models and previous attacks. Their results have been displayed in a summary in table 3.1 . Partial SCA, as part of ASCA, is easier than a full fledged side channel attack as it targets any (maximum possible) leakage information in few (minimum possible) rounds. Whereas a wholesome side channel attack aims to recover all unknowns exactly by itself. Obviously, for ASCA to be successful, leakage information must be exploited judiciously.

3.5 Proposed Attack Methodology

Sequence of steps of ASCA on stream ciphers can be summarized with the help of a flow chart as shown in fig 3.4. A brief introduction to the underlying concept can also be studied in [46].

3.5.1 Algebraic Attack

Algebraic attack generally has offline and online phases of execution. In offline phase, target stream cipher is transformed into a system of multivariate equations, whereas in online phase, adversary needs sufficient output stream bits, depending upon the unknown variables, to solve algebraic equations. Normally stream ciphers have an initialization phase during which no output bit is generated. Mostly algebraic attacks target complete internal state bits and then reach at the secret key by back tracking. However secret key before initialization can also be targeted. In the former case, number of unknowns would be more but equations would be of less degree whereas in the latter case equations would be of higher degree with less unknowns.

For solving multivariate equations many tools employing SAT solving, Groebner bases etc. can be found. For this research, we rely on SAT solving. For SAT solving, the MQ problem has to

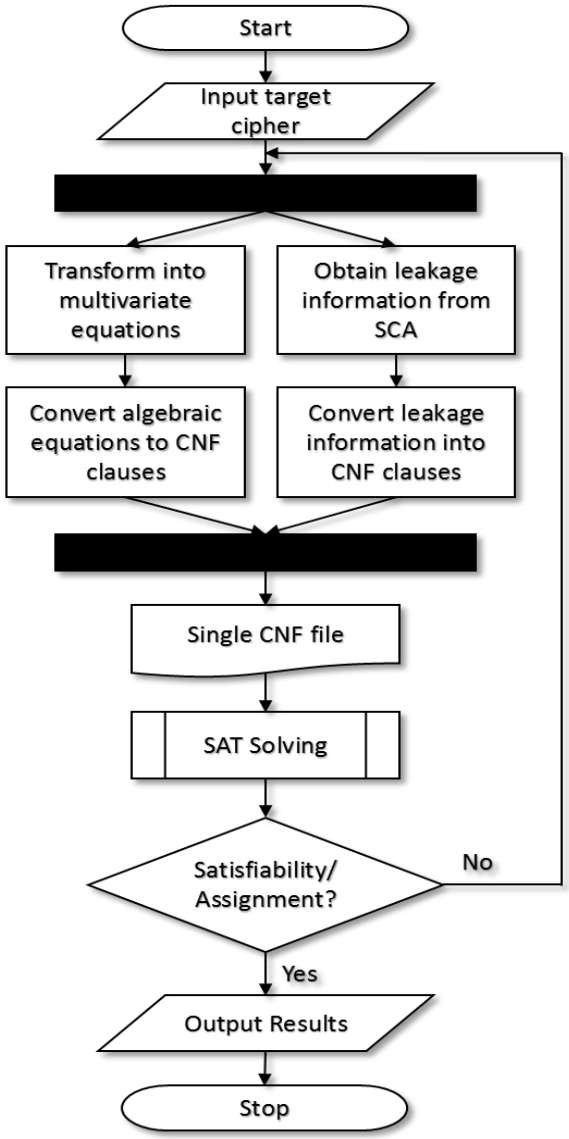

Figure 3.4: Proposed Attack Flow Chart

be converted to a SAT problem. SAT problem consists of clauses of conjunctive normal form (CNF) instead of algebraic equations. We make use of an open source software tool grain-of-salt developed by Mate Soos [47], which transforms target stream ciphers directly into CNF clauses. A basic tutorial on grain-of-salt is given at appendix A. System of multivariate algebraic equations can also be converted manually to SAT problem [48]. Later for finding the unknown variables, CNF clauses are input to an advance SAT solver, known as CryptoMiniSAT 5.0 in our experimentation. Installation and basic usage guide on CryptoMiniSAT 5.0 can be found at

appendix B.

3.5.2 Partial SCA

SCA attacks implementation of ciphers by exploiting leakage of power, electromagnetic radiations, execution time, photonic emissions etc. Timing analysis [29] is possible if relation between execution time of algorithm and its internal states exists. Cipher might be susceptible to timing attack if its algorithm has some conditional branch instructions or table look ups. In case of power analysis, hamming weight or hamming distance information is captured from the leaked power traces. Algorithmic noise due to hardware implementation can also be very helpful to attacker. Simple Power Analysis (SPA) [49] takes advantage of relation between instant power consumption and internal states in one or few power traces. DPA [49] exploits the difference in power consumption due to variation in the data being processed with same key. Though practically it may not be possible to restart the running of stream cipher, researchers have used scenarios where frequent resynchronization is needed [50]. In template attacks [15], adversary acquires exact replica of the encryption device which executes the stream cipher. This is a reasonable assumption as standard micro-controllers are used quite often for this purpose. In first phase, using this identical device, separate templates of typical signal are captured and recorded including associated noise against all possible key values. From actual device, a single leakage trace is thus required to match with recorded template to reach exact key. Instead of performing this classification process on entire key space extend and prune strategy is recommended to be used iteratively. Issues related to template attacks have been discussed in detail in [51]. Observing leakage pattern of the cipher's implementation, selecting leakage points where extraction of maximum useful information is possible in minimum requirement of access to the cipher's implementation and time are also important part of SCA.

For ASCA, side channel information obtained in SCA is transformed into algebraic equations. In this process some unknowns might even be resolved because of leakage information. Conversion of side channel information into usable clauses/ equations is further discussed while demonstrating ASCA on various stream ciphers in upcoming sections and in next chapter.

In this work, a theoretical approach has been adopted, where appropriate leakage points from software implementation of the target stream cipher have been selected and exact side channel leakage information has been extracted. SAT solving is used for finding the unknown variables, therefore, CNF clauses have been formed against partial side channel leakage information.

3.5.3 Solving Combined Information

In ASCA, the multivariate equations obtained from both the phases are combined together as one system of equations, which is then solved to obtain the internal state bits or secret key bits. In this work, the CNF clauses from both phases are combined together to form one CNF file for the input of SAT solver. CryptoMiniSat-5.0 has been employed for satisfiability and assignment to unknown variables.

There may be a case where in spite of adding side channel information, the overall system of equations/ clauses is not solved in reasonable time (a threshold of 3600 seconds was kept for this work.) In such a scenario, more information from SCA may be extracted or even algebraic attack part may also be revisited. Incorrect side channel information would make the system of equations/ clauses unresolvable.

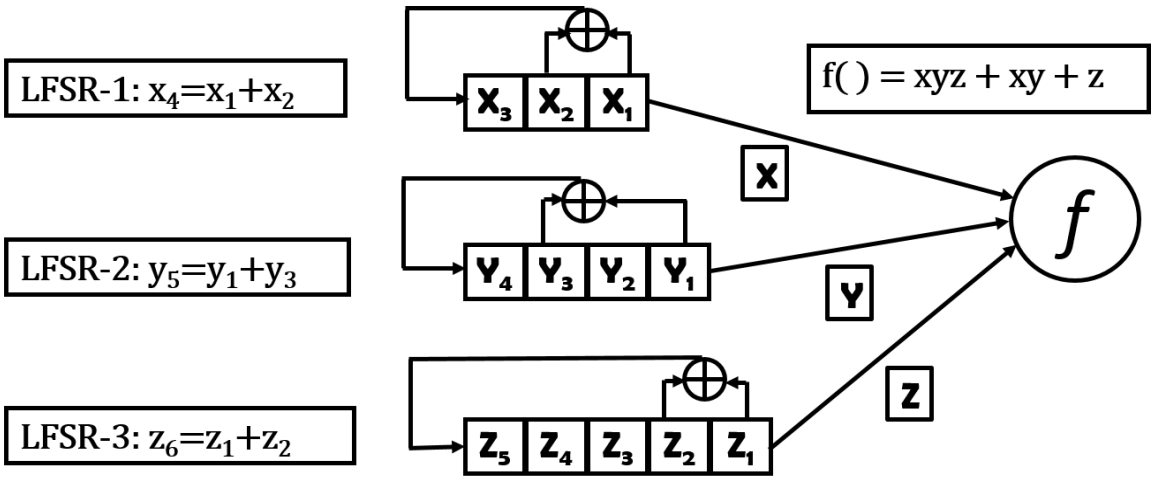

Figure 3.5: A Hypothetical Stream Cipher

3.6 ASCA on Stream Ciphers - A Trivial Example

Here we explain the application of our proposed attack on a simple hypothetical stream cipher (fig 3.5) based on linear feedback shift registers (LFSRs) and a nonlinear combiner function, to demonstrate the methodology. With the knowledge of first few bits of the output stream, adversary formulates few multivariate equations by traditional algebraic attack. Then we assume that adversary is able to get exact hamming weights at the input of combiner function $f()$. Lastly adversary adds the information from both phases to resolve the MQ problem, thereby reaching at internal state bits.

3.6.1 Algebraic Attack Phase

Adversary acquires seven bits of output stream as 1100111. The degree of combiner function $f()$ is 3. Following 2-degree annihilator functions of $f()$ are calculated:

- $g_1() = 1 + z + x + xz.$
- $g_2() = 1 + y + z + yz.$

- $g_3() = x + y + xz + yz$.

- There is no annihilator of $1 + f()$.

Next, following equations can be calculated at first seven clocks, involving unknown state bits and known output bits, from $g()$'s and $f()$:

- Equations against $g_1()$:
  - At t=0: $1 + z_1 + x_1 + x_1 z_1 = 0$
  - At t=1: $1 + z_2 + x_2 + x_2 z_2 = 0$
  - At t=4: $1 + z_5 + x_5 + x_5 z_5 = 0$
  - At t=5: $1 + z_6 + x_6 + x_6 z_6 = 0$
  - At t=6: $1 + z_7 + x_7 + x_7 z_7 = 0$
- Equations against $g_2()$:
  - At t=0: $1 + y_1 + z_1 + y_1 z_1 = 0$
  - At t=1: $1 + y_2 + z_2 + y_2 z_2 = 0$
  - At t=4: $1 + y_5 + z_5 + y_5 z_5 = 0$
  - At t=5: $1 + y_6 + z_6 + y_6 z_6 = 0$
  - At t=6: $1 + y_7 + z_7 + y_7 z_7 = 0$
- Equations against $g_3()$:
  - At t=0: $x_1 + y_1 + x_1 z_1 + y_1 z_1 = 0$
  - At t=1: $x_2 + y_2 + x_2 z_2 + y_2 z_2 = 0$

  - At t=4: $x_5 + y_5 + x_5z_5 + y_5z_5 = 0$
  - At t=5: $x_6 + y_6 + x_6z_6 + y_6z_6 = 0$
  - At t=6: $x_7 + y_7 + x_7z_7 + y_7z_7 = 0$

- Equations against $f()$:
  - At t=0: $x_1y_1z_1 + x_1y_1 + z_1 = 1$
  - At t=1: $x_2y_2z_2 + x_2y_2 + z_2 = 1$
  - At t=2: $x_3y_3z_3 + x_3y_3 + z_3 = 0$
  - At t=3: $x_4y_4z_4 + x_4y_4 + z_4 = 0$
  - At t=4: $x_5y_5z_5 + x_5y_5 + z_5 = 1$
  - At t=5: $x_6y_6z_6 + x_6y_6 + z_6 = 1$
  - At t=6: $x_7y_7z_7 + x_7y_7 + z_7 = 1$

3.6.2 SCA Phase

Through side channel attack adversary finds out hamming weights (HW) at the input of function $f()$ as follows:

- At $t = 0$, $HW = 3$: $x_1 = y_1 = z_1 = 1$
- At $t = 1$, $HW = 2$: $x_2y_2z_2 = 0$
- At $t = 2$, $HW = 1$: $x_3y_3 = y_3z_3 = x_3z_3 = x_3y_3z_3 = 0$
- At $t = 3$, $HW = 1$: $x_4y_4 = y_4z_4 = x_4z_4 = x_4y_4z_4 = 0$
- At $t = 4$, $HW = 1$: $x_5y_5 = y_5z_5 = x_5z_5 = x_5y_5z_5 = 0$

- At $t = 5$, $HW = 2$: $x_6 y_6 z_6 = 0$

- At $t = 6$, $HW = 3$: $x_7 = y_7 = z_7 = x_7 y_7 = x_7 z_7 = y_7 z_7 = x_7 y_7 z_7 = 1$

3.6.3 Combining Equations from both Phases and Solving

- Combining equations at t=0 from both phases: $x_1 = y_1 = z_1 = 1$

- Combining equations at t=1 from both phases: we are unable to find any value.

- Combining equations at t=2 from both phases: $z_3 = 0$

- Combining equations at t=3 from both phases: $z_4 = 0$

- Combining equations at t=4 from both phases: $1 + z_5 + x_5 = 0, 1 + y_5 + z_5 = 0, x_5 + y_5 = 0$

- Either $x_5 = y_5 = 1, z_5 = 0$ Or $x_5 = y_5 = 0, z_5 = 1$ But as $HW = 1$: $x_5 = y_5 = 0$ and $z_5 = 1$ is correct.

- Combining equations at t=5 from both phases: we are unable to find any value.

- Combining equations at t=6 from both phases: $x_7 = y_7 = z_7 = 1$

- From LFSR-1 equation $x_4 = x_1 + x_2$ and already calculated Xs($x_1 = 1, x_5 = 0, x_7 = 1$): $x_2 = x_3 = 0$

- From LFSR-2 equation $y_5 = y_1 + y_3$ and already calculated Ys($y_1 = 1, y_5 = 0, y_7 = 1$): $y_3 = 1$

- From LFSR-3 equation $z_6 = z_1 + z_2$ and already calculated Zs ($z_1 = 1, z_3 = 0, z_4 = 0, z_5 = 1, z_7 = 1$): $z_2 = 1$

Initial states of LFSR-1&3 are completely known however for LFSR-2, y_2 and y_4 are unknown yet. As adversary knows the hamming weights at first seven clocks, so y_2 and y_4 is calculated as follows:

- At $t = 1, x_2 = 0, y_2 = ?, z_2 = 1$ and $HW = 2$ therefore $y_2 = 1$
- At $t = 3, x_4 = x_1 + x_2 = 1, z_4 = 0$ and $HW = 1$ therefore $y_4 = 0$

Therefore, all the internal state bits are now known. Same methodology as in this example attack can be applied on real stream ciphers based as well, except the difference that the solving will have to be carried out with some mathematical tool.

3.7 ASCA on Crypto-1 - Proof of Concept

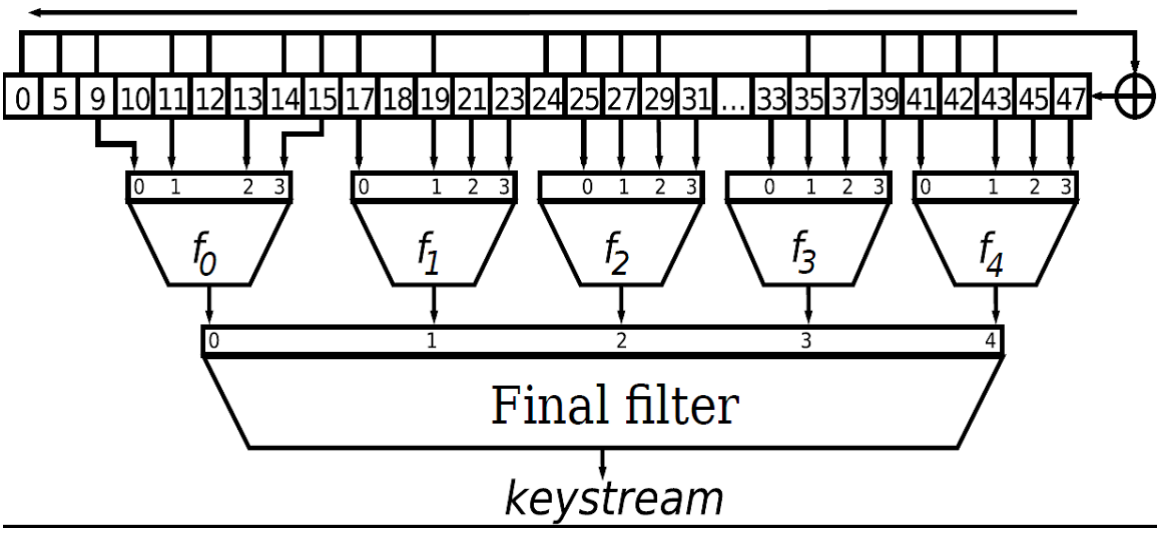

Figure 3.6: Structure of Crypto-1 Stream Cipher

3.7.1 Description of Crypto-1 Stream Cipher

Crypto-1 is a lightweight stream cipher which has been widely used ticketing and wireless access control systems [52]. It has a simple design and comprises nonlinear functions, therefore it has

been selected as first target as proof of concept. Crypto-1 stream cipher as shown in fig 3.6 [53], consists of a single 48-bit LFSR which is initialized with a 48-bit key. Then five sets of 4-bits each from LFSR are fed into five 3-degree nonlinear functions ($f_0, f_1, .., f_4$). If the input bits to these function are denoted by a, b, c, d then the functions are:

$f_0 = f_3 = a \oplus b \oplus b.a \oplus c.a \oplus c.b \oplus d.a \oplus d.c \oplus d.c.a \oplus d.c.b$

$f_1 = f_2 = f_4 = b.a \oplus c \oplus c.a \oplus c.b \oplus c.b.a \oplus d.a \oplus d.b \oplus d.c.a \oplus d.c.b$

The output of these five functions is fed to a 4-degree nonlinear final filter function which generates the key stream:

$f_0 \oplus f_2.f_0 \oplus f_3.f_0 \oplus f_3.f_1.f_0 \oplus f_3.f2.f_1 \oplus f_4 \oplus f_4.f_0 \oplus f_4.f_1.f_0 \oplus f_4.f_2.f_1.f_0 \oplus f_4.f_3 \oplus$

$f_4.f_3.f_0 \oplus f_4.f_3.f_1 \oplus f_4.f_3.f_2.f_1$

Though Crypto-1 has already been broken through algebraic attack in 200 seconds on a PC in [53], but it has been chosen here being an easy first target for demonstrating as to how partial side channel information based on hamming weight leakage model can improve the solving time in ASCA as compared to algebraic attacks.

3.7.2 Algebraic Attack - Offline Phase

In the offline phase, algebraic attack was launched on the cipher making use of 50 random samples of 50-bit output stream using grain-of-salt tool. On a Linux VM furnished with 2 processors and 2.8 GB memory, on the average, it took 509.017 seconds and 26.807 MBs of memory to solve SAT problem comprising 24636 CNF clauses and correctly assign values to the unknown variable by CryptoMiniSat 5.0.

3.7.3 SCA - Online Phase

For online phase, a template-like SCA is simulated as in [14], extracting hamming weight leakage at the inputs of filter functions f_0, f_1, f_2, f_3, f_4 and final filter function. Algebraic equations can be obtained from hamming weight information following procedure given in [19], however, here, this information is directly converted to CNF clauses as needed by the SAT solver. For example, for four inputs of function f_0 at time t, if hamming weight $HW(t) = k$, then product of any $k+1$ or more bits will always be zero. Here value of k can be $0, 1, 2, 3, 4$ (i.e. $|k| = 5$). Possible CNF clauses for various values of k, where variables $1, 2, 3$ and 4 are used for input bits to the function f_0, are displayed in table 3.2.

Table 3.2: Converting HW(t) into CNF Clauses

HW(t)	Implication	CNF Clauses
k=0	all bits are zero	$-1\ 0,\ -2\ 0,\ -3\ 0,\ -4\ 0$
k=1	OR of any two or more bits is one	$-1\ -2\ 0, -1\ -3\ 0, -1\ -4\ 0, -2\ -3\ 0,$ $-2\ -4\ 0, -3\ -4\ 0, -1\ -2\ -3\ 0, -1\ -2\ -4\ 0,$ $-1\ -3\ -4\ 0, -2\ -3\ -4\ 0, -1\ -2\ -3\ -4\ 0, 1\ 2\ 3\ 4\ 0$
k=2	OR of any three or more bits is one	$1\ 2\ 3\ 0, 1\ 2\ 4\ 0, 1\ 3\ 4\ 0, 2\ 3\ 4\ 0, 1\ 2\ 3\ 4\ 0,$ $-1\ -2\ -3\ 0, -1\ -2\ -4\ 0, -1\ -3\ -4\ 0,$ $-2\ -3\ -4\ 0, -1\ -2\ -3\ -4\ 0$
k=3	OR of any four or more bits is one	$1\ 2\ 0, 1\ 3\ 0, 1\ 4\ 0, 2\ 3\ 0, 2\ 4\ 0, 3\ 4\ 0,$ $1\ 2\ 3\ 0, 1\ 2\ 4\ 0, 1\ 3\ 4\ 0, 2\ 3\ 4\ 0,$ $1\ 2\ 3\ 4\ 0, -1\ -2\ -3\ -4\ 0$
k=4	all bits are one	$1\ 0, 2\ 0, 3\ 0, 4\ 0,$

3.7.4 Solving Combined Information and Results

Over a thousand CNF clauses thus acquired from SCA are added into those obtained from algebraic attack and combined CNF file is input to CryptoMiniSAT 5.0 for satisfiability/ assignment. Experimental results of ASCA against same 50 random samples of 50-bit output stream on same machine as used for algebraic attack, took 0.158 seconds and 6.204 MBs of memory, on the av-

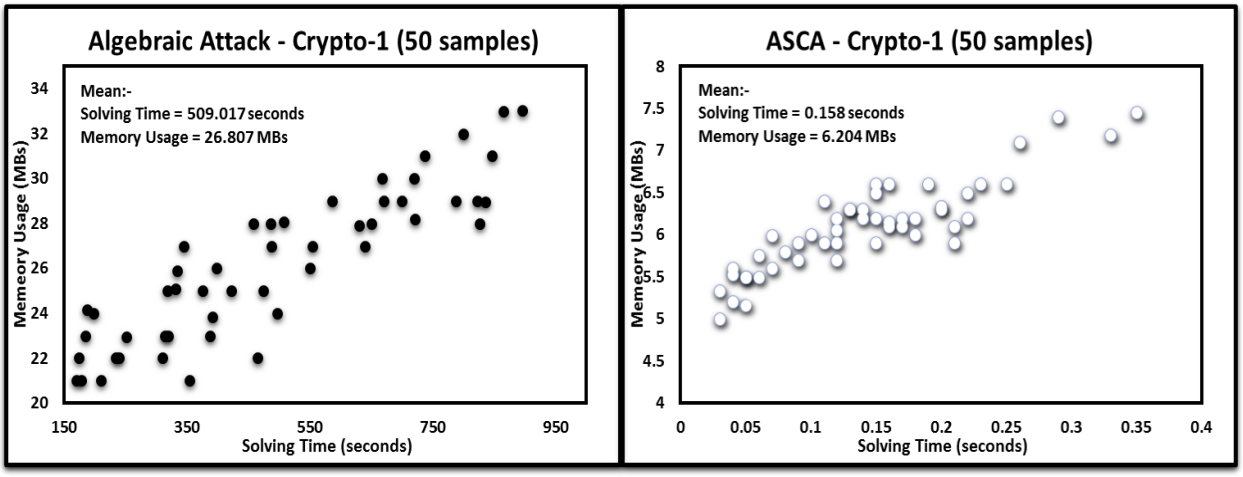

Figure 3.7: A Comparison of Algebraic Attack and ASCA on Crypto-1 Stream Cipher

erage, to complete the attack successfully. A comparison drawn between ASCA and algebraic attack on Crypto-1 stream cipher in the light of above solving, in fig 3.7 clearly indicates former's efficiency.

3.8 Conclusion

This chapter discusses that by augmenting algebraic attacks on stream ciphers with partial side channel leakage information from the hardware implementation of the cipher, resolution of MQ problem can be simplified. An attack methodology in this regard has been presented coupled with a trivial example-attack on a hypothetical stream cipher. In the example, equations obtained from pure algebraic technique have all internal states of LFSRs as unknowns. The additional information coming from SCA is also converted to algebraic equations. When these additional equations are added it becomes easier to resolve the system.

At the culmination of the chapter, newly proposed ASCA methodology is tested against a real world stream cipher, Crypto-1, as a proof of concept. In this attack, the system of multivariate equations is converted to CNF clauses and resolved as a SAT problem with the help of Crypto-

MiniSAT 5.0, an advanced SAT solver. Better efficiency of ASCA as compared to lone algebraic attacks has been proven. Though even lone algebraic attack against Crypto-1 in the past was a success, but ASCA has been able to break the cipher in much less time as demonstrated. Main advantage with ASCA is the additional side channel leakage information, which makes the system of equations furthermore over-defined.

Chapter 4

ASCA ON TRIVIUM AND GRAIN

4.1 Introduction

Concept of ASCA against stream ciphers has been amply explained in the previous chapters and it has been applied against a real world stream cipher Crypto-1 in chapter 3. However Crypto-1 stream cipher has already been successfully attacked through lone algebraic attack. In this chapter, effort is made to apply ASCA against such stream ciphers which are intractable when attacked using pure algebraic cryptanalysis.

4.2 ASCA on Bivium-B and Trivium

First stream cipher chosen for testing ASCA efficacy is Trivium. No published work on successful algebraic attack against full Trivium cipher. It would be interesting to see how Trivium can be attacked with ASCA technique. Bivium-B is a reduced variant of Trivium, which is also subjected to ASCA along side Trivium, though Bivium-B is not intractable when subjected to algebraic attack.

4.2.1 Description of Trivium Stream Cipher

Trivium stream cipher [54], has an internal state of 288 bits comprising three nonlinear registers of 93, 84 and 111 bits as illustrated in fig 4.1. For initialization the cipher is clocked $4 * 288$ times, after 80 bit secret key is loaded into first register, 80 bit IV is loaded into second register and rest all states are loaded with zeros except ones in last three bits of third register. Output

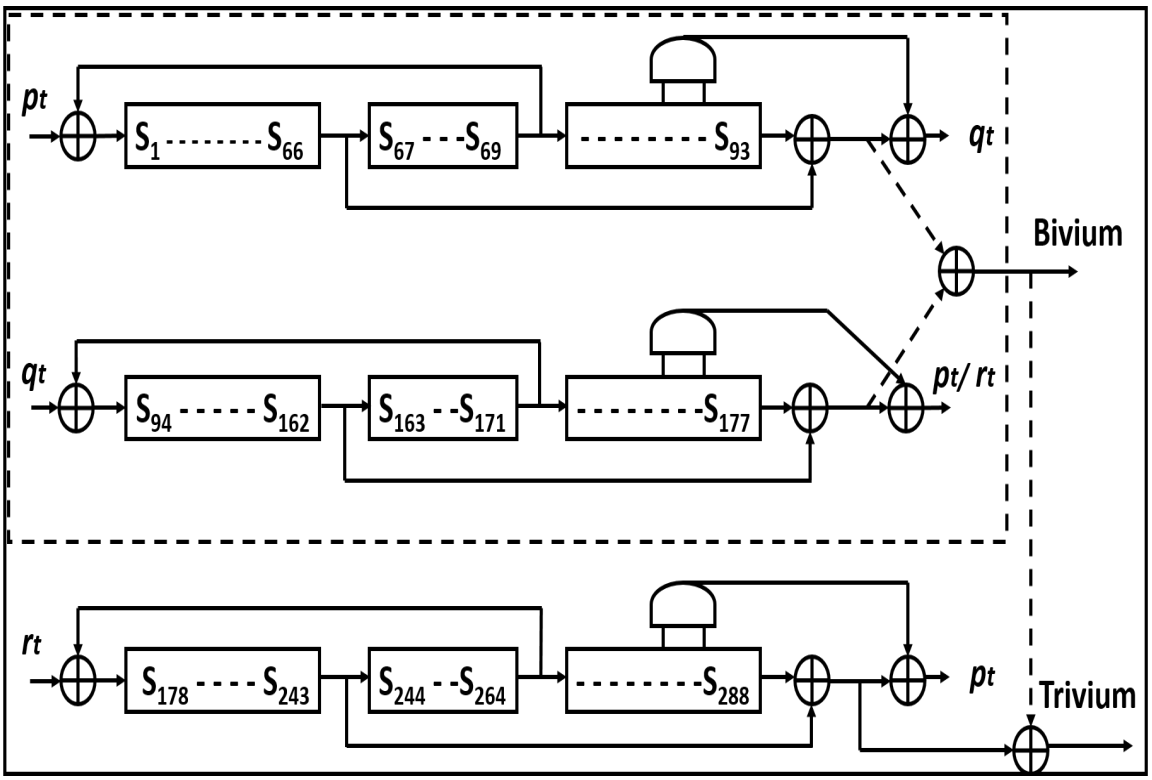

Figure 4.1: Structure of Bivium-B and Trivium Stream Ciphers

stream is produced from a linear output function which combines six internal state bits. Trivium pseudocode is given as algorithm 1.

4.2.2 Best Known Algebraic Attack on Trivium

Algebraic attacks on Trivium in [55], [48], [56], [57], [58], [59] and [60] target complete internal state bits instead of key. The best overall time and data complexity was found to be $2^{42.5}$ and 2^{12} respectively on an average computer in [55] that targeted a round-reduced variant of Trivium of 625 rounds.

4.2.3 Description of Bivium-B Stream Cipher

Bivium-B stream cipher, a reduced variant of Trivium (fig 4.1), has an internal state of 177 bits comprising two nonlinear registers of 93 and 84 bits which are initialized with 80 bit secret key

Algorithm 1: Trivium Pseudocode

for $i \leftarrow 1$ **to** N **do**
    $t_1 \leftarrow s_{66} \oplus s_{93}$;
    $t_2 \leftarrow s_{162} \oplus s_{177}$;
    $t_3 \leftarrow s_{243} \oplus s_{288}$;
    $z_i \leftarrow t_1 \oplus t_2 \oplus t_3$;
    $t_1 \leftarrow t_1 \oplus s_{91}.s_{92} \oplus s_{171}$;
    $t_2 \leftarrow t_2 \oplus s_{175}.s_{176} \oplus s_{264}$;
    $t_3 \leftarrow t_3 \oplus s_{286}.s_{287} \oplus s_{69}$;
    $(s_1, s_2, ..., s_{93}) \leftarrow (t_3, s_1, ..., s_{92})$;
    $(s_{94}, s_{95}, ..., s_{177}) \leftarrow (t_1, s_{94}, ..., s_{176})$;
    $(s_{178}, s_{179}, ..., s_{288}) \leftarrow (t_2, s_{178}, ..., s_{287})$;
end

and 80 bit IV respectively with zeros at remaining states. The cipher is clocked $4 * 177$ times before producing the output stream from a linear output function which combines 4 bits out of 177 internal states [56]. Pseudocode of Bivium-B is given as algorithm 2.

Algorithm 2: Bivium-B Pseudocode

for $i \leftarrow 1$ **to** N **do**
    $t_1 \leftarrow s_{66} \oplus s_{93}$;
    $t_2 \leftarrow s_{162} \oplus s_{177}$;
    $z_i \leftarrow t_1 \oplus t_2$;
    $t_1 \leftarrow t_1 \oplus s_{91}.s_{92} \oplus s_{171}$;
    $t_2 \leftarrow t_2 \oplus s_{175}.s_{176} \oplus s_{69}$;
    $(s_1, s_2, ..., s_{93}) \leftarrow (t_2, s_1, ..., s_{92})$;
    $(s_{94}, s_{95}, ..., s_{177}) \leftarrow (t_1, s_{94}, ..., s_{176})$;
end

4.2.4 Best Known Algebraic Attack on Bivium-B

In [47] Soos M. have claimed to break Bivium-B in an approximated $2^{36.5}$ seconds using a Xeon E5345@2.33GHz computer with the help of grain-of-salt tool to generate CNF clauses from pure algebraic attack and solving them using CryptoMiniSat [61]. This is the most efficient algebraic attack against Bivium-B as per our knowledge.

4.2.5 Attack Modalities and Results

Our ASCA on Bivium-B/ Trivium targets 80 bit secret key instead of complete state of cipher after initialization, while capturing 80 output bits. The success of ASCA on Bivium-B/ Trivium stream cipher or for that matter any other cipher is largely dependent upon availability of leakage point in the hardware implementation of the cipher so that maximum possible side channel information could be extracted in minimum exposure. By looking at the design of both ciphers, there seems no worthwhile benefit of a side channel extraction similar to that employed against Crypto-1 cipher as above. One may attempt to capture hamming weight of registers at each round in a template-like attack and convert them into algebraic equations or CNF clauses but due to bigger size of registers, entropy of hamming weights would be very high [62]. Therefore equations based on hamming weights of registers would not reduce the overall complexity of ASCA to a greater extent. In [45] authors accessed susceptibility of Trivium towards SCA and concluded that its power consumption can be easily described using hamming distance model. They also highlighted that there is almost no possibility of recovering internal states after initialization phase, as the key is later spread into all registers, so useful side channel information can only be extracted in the initialization phase. Due to same inherent structure, this is also applicable to Bivium-B. Moreover authors suggested that a DPA can be mounted while focusing on the input s_{94} of second register. The contents of this flip-flop after each round are given by following equation from pseudocode (line 8 for Bivium-B and line 10 for Trivium as above):

$s_{94}(i+1) = s_{66}(i) \oplus s_{91}(i).s_{92}(i) \oplus s_{93}(i) \oplus s_{171}(i)$

As we choose to subject Bivium-B and Trivium to a known IV attack, therefore only unknown on the right hand side of above mentioned equation is $s_{66}(i)$ which is equal to 66^{th} bit of secret key in round 1. In a DPA, two hypothetical power consumptions of second register using both values (0

and 1) for s_{94}, can help in determining its correct value with correlation coefficient. Consequently, an additional equation or few CNF clauses from SCA are obtained which can augment the system of equations or CNF clauses obtained through classic algebraic attack.

For next round s_{94} will assume new value but rest all bits of second register are known so one can continue with DPA in a similar fashion and put in correct values for s_{94} in subsequent equations of next rounds. How many values of s_{94} or how many round-equations are needed? We aim at involving all 80 secret key bits in these equations and for that proceeding till round 66 is enough, for both Bivium-B and Trivium. Resultantly equations based on following expressions are acquired, either equal to 1 or 0 depending upon the value of s_{94} as per DPA outcome:

Round 1: $k_{66} \oplus 0 * 0 \oplus 0 \oplus iv_{78}$

Round 2: $k_{65} \oplus 0 * 0 \oplus 0 \oplus iv_{77}$

.

.

.

Round 11: $k_{56} \oplus 0 * 0 \oplus 0 \oplus iv_{68}$

Round 12: $k_{55} \oplus k_{80} * 0 \oplus 0 \oplus iv_{67}$

Round 13: $k_{54} \oplus k_{79} * k_{80} \oplus 0 \oplus iv_{66}$

Round 14: $k_{53} \oplus k_{78} * k_{79} \oplus k_{80} \oplus iv_{65}$

Round 15: $k_{52} \oplus k_{77} * k_{78} \oplus k_{79} \oplus iv_{64}$

.

.

.

Round 64: $k_3 \oplus k_{28} * k_{29} \oplus k_{30} \oplus iv_{15}$

Round 65: $k_2 \oplus k_{27} * k_{28} \oplus k_{29} \oplus iv_{14}$

Round 66: $k_1 \oplus k_{26} * k_{27} \oplus k_{28} \oplus iv_{13}$

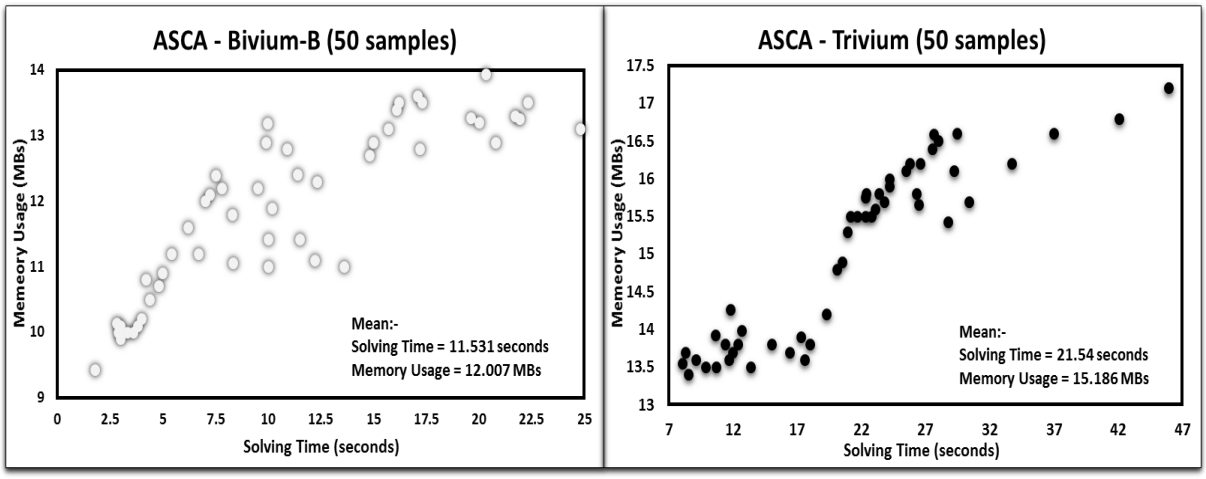

Figure 4.2: ASCA Results against Bivium-B and Trivium Stream Ciphers

Though for SCA part only equations from first 66 rounds were utilized, but for the algebraic attack part there is a need to include complete initialization phase ($4 * 177$ rounds in case of Bivium-B and $4 * 288$ rounds in case of Trivium), other than 80 additional rounds for 80 output bits. This is done in order to employ variables introduced for initial state bits in the new equations expressing relation of output bits with the state bits. Including initialization phase in the algebraic attack has a disadvantage of dealing with high degree equations but at the same time has an advantage of targeting less unknown variables of only secret key bits as compared to that of complete state bits. In ASCA, since leakage information is obtained at initialization phase, therefore this has to be included in algebraic attack phase as well.

Lastly the CNF clauses obtained so far from both algebraic and SCA phases are combined together as input it to CryptoMiniSat 5.0 for satisfiability/ assignment.

Table 4.1: ASCA Results against Bivium-B and Trivium

Bivium-B		**Trivium**	
Solving Time Range (sec)	**# of Samples**	**Solving Time Range (sec)**	**# of Samples**
$t < 5$	12	$t < 10$	5
$5 \leq t < 10$	13	$10 \leq t < 20$	15
$10 \leq t < 15$	10	$20 \leq t < 30$	25
$15 \leq t < 20$	8	$30 \leq t < 40$	3
$20 \leq t < 25$	7	$40 \leq t < 50$	2

The results of ASCA against both Bivium-B and Trivium, on a Linux machine, with Intel Core i3 CPU M350 @2.27GHz and 6 GB memory, are summarized in fig 4.2 and table 4.1. Targeting 80 bit secret key, on 50 random samples each making use of 80 output stream bits, on the average, it took 11.531 and 21.54 seconds along with 12.007 and 15.186 MBs of memory to complete the attack successfully on Bivium-B and Trivium respectfully.

Trivium stream cipher has been regarded as highly resistant to algebraic attacks due to the computational infeasibility to solve its associated MQ problem. However, here, Trivium's multivariate equations converted into SAT problem were solved in slightly over 21 seconds. Thanks to partial side channel leakage information extracted from a correlation DPA. The said side channel information was also converted into CNF clauses so as to simplify the overall SAT problem, thereby making the attack a success.

4.3 ASCA on Grain

Another popular stream cipher Grain is chosen for mounting ASCA in this section. Grain is considered to be extremely resistant to algebraic attacks. Full round Grain has never been successfully attacked through algebraic technique. On the other hand, research work highlighting

high algebraic immunity of Grain against algebraic attacks exists in literature. Grain's nonlinear update function results into increasing degree of equations with each clock, when transformed into algebraic equations. A successful ASCA is demonstrated against Grain in ensuing paragraphs.

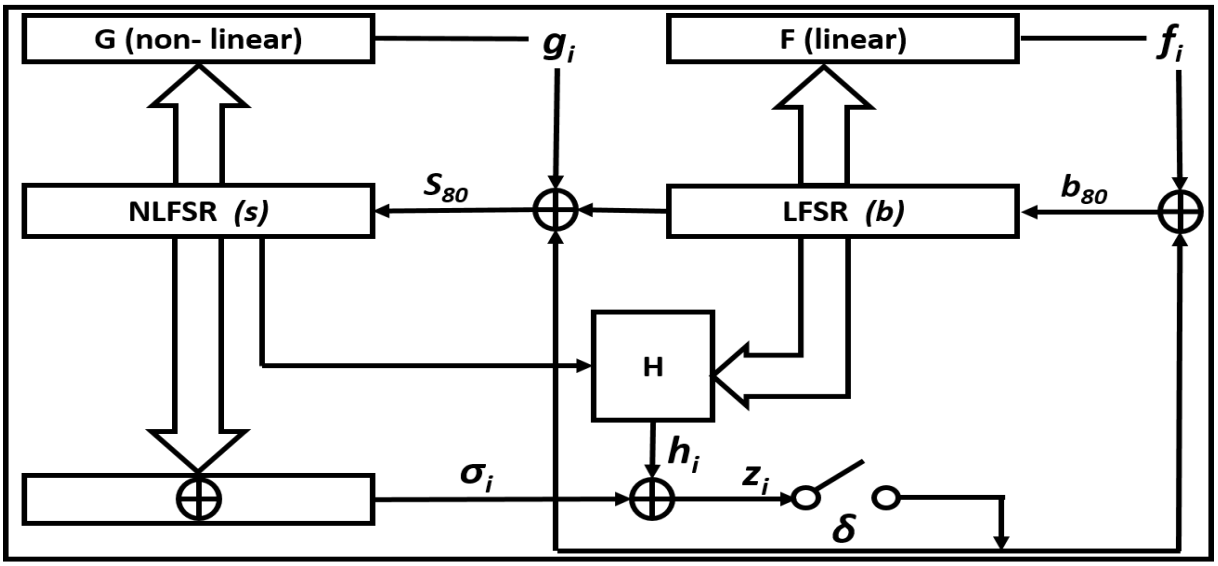

Figure 4.3: Structure of Grain Stream Cipher

4.3.1 Description of Grain Stream Cipher

Grain v1 cipher [63], illustrated in fig 4.3, consists of an 80-bit NLFSR (denoted as s with state bits $s_0, s_1, ..., s_{79}$) and an 80-bit LFSR (denoted as b with state bits $b_0, b_1, ..., b_{79}$) which are loaded with the 80-bit secret key and 64-bit IV plus remaining 1's, respectively. Output h_i of a 3-degree nonlinear function H, which gets 5 input bits from both registers, is XORed with a linear function σ_i to generate output stream z_i. During initialization phase, which lasts for 160 rounds, $\delta = 1$ so that no output stream is generated and z_i is XORed with the feedback functions of both registers. Following equations govern the working of the cipher:

- Feedback functions of s and b:

  $s_{80} = b_0 \oplus g_i \oplus z_i.\delta$

$b_{80} = f_i \oplus z_i.\delta$

- 6-degree Nonlinear function g_i:

  $s_{62} \oplus s_{60} \oplus s_{52} \oplus s_{45} \oplus s_{37} \oplus s_{33} \oplus s_{28} \oplus s_{21} \oplus s_{14} \oplus s_9 \oplus s_0 \oplus s_{63}s_{60} \oplus s_{37}s_{33}$

  $\oplus s_{15}s_9 \oplus s_{60}s_{52}s_{45} \oplus s_{33}s_{28}s_{21} \oplus s_{63}s_{45}s_{28}s_9 \oplus s_{60}s_{52}s_{37}s_{33} \oplus s_{63}s_{60}s_{21}s_{15} \oplus$

  $s_{63}s_{60}s_{52}s_{45}s_{37} \oplus s_{33}s_{28}s_{21}s_{15}s_9 \oplus s_{52}s_{45}s_{37}s_{33}s_{28}s_{21}$

- Linear function f_i:

  $b_{62} \oplus b_{51} \oplus b_{38} \oplus b_{23} \oplus b_{13} \oplus b_0$

- Linear function σ_i:

  $s_1 \oplus s_2 \oplus s_4 \oplus s_{10} \oplus s_{31} \oplus s_{43} \oplus s_{56}$

- 3-degree Nonlinear function h_i:

  $b_{25} \oplus s_{63} \oplus b_3b_{64} \oplus b_{46}b_{64} \oplus b_{64}s_{63} \oplus b_3b_{25}b_{46} \oplus b_3b_{46}b_{64} \oplus b_3b_{46}s_{63} \oplus b_{25}b_{46}s_{63}$

  $\oplus b_{46}b_{64}s_{63}$

- Output stream z_i:

  $\sigma_i \oplus h_i$

4.3.2 Best Known Algebraic Attack on Grain v1

There haven't been much work/ success on algebraic cryptanalysis of Grain v1 in the past. Though Grain v1 has been subjected to other attacks like in [64], [65], [66] but only one instance of pure algebraic attack against Grain v1 is published in [41], claiming to solve equations of Grain v1 in $2^{80.70}$ seconds, using Groebner bases technique.

4.3.3 Unsuccessful Attempt

Similar to attacks on other ciphers in previous sections, Grain v1 is converted into CNF clauses by the help of grain-of-salt tool [47], using 80 output stream bits. Initialization phase consisting of 160 rounds is included in the attack, as we intended targeting the secret key directly instead of targeting complete 160 state bits.

In [45] authors mentioned that Grain v1 is susceptible to SPA in case of bit-serialized implementations, by measuring hamming distances of key bits subsequently after resetting to a defined state at key setup. Whereas in [42], Fischer et al. mounted a successful DPA with chosen IV while attacking the key setup process thereby learning the key bits iteratively. Authors in [45] also mentioned that a DPA could possibly be mounted specifically before or after the output function based on hamming distance model.

In this work, in a known IV scenario, new incoming bit of LFSR in each round was targeted, under the fact that all 80 bits of the LFSR b are known initially at round 0. A correlation DPA can measure the value of new bit in each subsequent round by the help of all known bits of LFSR in the previous round as follows:

- From the equations given above it can be observed that in round 1, b_{80} would equal:

  $b_{62} \oplus b_{51} \oplus b_{38} \oplus b_{23} \oplus b_{13} \oplus b_0 \oplus s_1 \oplus s_2 \oplus s_4 \oplus s_{10} \oplus s_{31} \oplus s_{43} \oplus s_{56} \oplus b_{25} \oplus s_{63}$

  $\oplus\, b_3 b_{64} \oplus b_{46} b_{64} \oplus b_{64} s_{63} \oplus b_3 b_{25} b_{46} \oplus b_3 b_{46} b_{64} \oplus b_3 b_{46} s_{63} \oplus b_{25} b_{46} s_{63} \oplus b_{46} b_{64} s_{63}$

- Here all bits coming from LFSR with suffix b are known from previous round, therefore value of b_{80} becomes:

  $s_1 \oplus s_2 \oplus s_4 \oplus s_{10} \oplus s_{31} \oplus s_{43} \oplus s_{56} \oplus s_{63} \oplus b_{64} s_{63} \oplus b_3 b_{46} s_{63} \oplus b_{25} b_{46} s_{63} \oplus$

  $b_{46} b_{64} s_{63} \oplus k_1$

- Further b_{80} can be simplified as:

  $s_1 \oplus s_2 \oplus s_4 \oplus s_{10} \oplus s_{31} \oplus s_{43} \oplus s_{56} \oplus s_{63}.k_2 \oplus k_1$

- If the value of b_{80} is measured as b through correlation DPA, then one of the two possible equations can be obtained for each round during the initialization phase as under:

  - For $k_2 = 1$: $s_1 \oplus s_2 \oplus s_4 \oplus s_{10} \oplus s_{31} \oplus s_{43} \oplus s_{56} \oplus s_{63} = b \oplus k_1$
  - For $k_2 = 0$: $s_1 \oplus s_2 \oplus s_4 \oplus s_{10} \oplus s_{31} \oplus s_{43} \oplus s_{56} = b \oplus k_1$

- After the initialization phase, the value of b_{80} is only determined from bits of LFSR as under:

  $b_{62} \oplus b_{51} \oplus b_{38} \oplus b_{23} \oplus b_{13} \oplus b_0$

  Therefore side channel equations targeting b_{80} would only be useful till the end of initialization phase.

160 equations thus obtained through SCA were added into CNF clauses already obtained. However none of the 50 samples resulted into a satisfiable solution within our time limit (3600 seconds.)

4.3.4 Successful Attack and Results

In the second attempt we only modified the SCA part to capture the leakage at the input of function h_i and achieved success. This susceptibility of Grain v1 was also pointed out by authors in [45], though no practical SCA on Grain v1 exploiting this leakage has been published so far. We propose a template-like SCA [15], at the input of h_i using hamming weight leakage model. Similar technique was applied in [14] by authors while applying ASCA against PRESENT block cipher, when they captured hamming weight leakage at the input and output of its S-boxes through template attacks.

Input bits at function h_i during various rounds are:

Round 1: $b_3, b_{25}, b_{46}, b_{64}, s_{63}$

Round 2: $b_4, b_{26}, b_{47}, b_{65}, s_{64}$

Round 3: $b_5, b_{27}, b_{48}, b_{66}, s_{65}$

... and so on

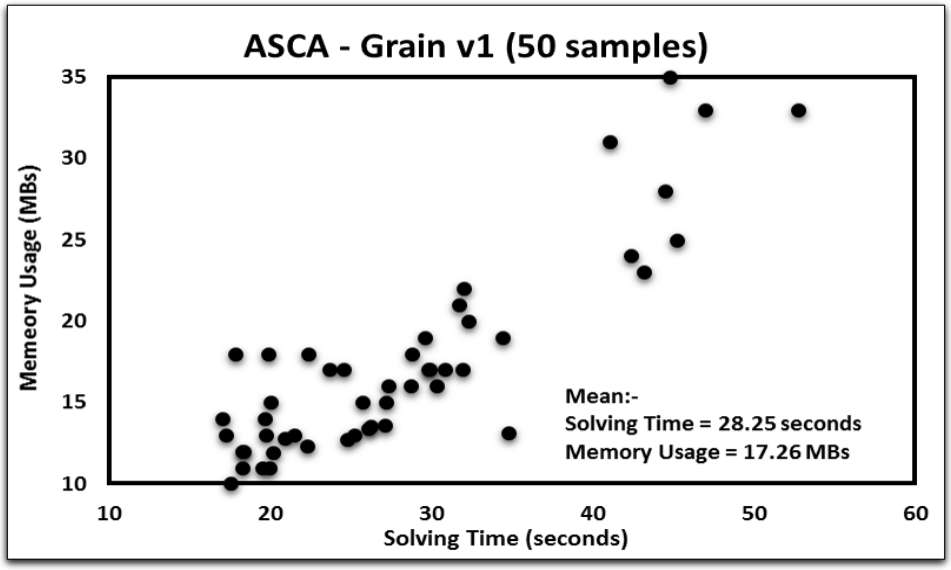

Figure 4.4: ASCA Results against Grain v1 Stream Cipher

Table 4.2: ASCA Results against Grain v1

Solving Time Range (sec)	**# of Samples**	**Memory Usage (MBs)**	**# of Samples**
$10 \leq t < 20$	12	$10 \leq m < 15$	21
$20 \leq t < 30$	21	$15 \leq m < 20$	18
$30 \leq t < 40$	9	$20 \leq m < 25$	5
$40 \leq t < 50$	7	$25 \leq m < 30$	2
$50 \leq t < 60$	1	$30 \leq m < 35$	4

Hamming weight information of these bits can either be converted to algebraic equations as explained [19] or directly to CNF cluases as explained in Section 4. Here it was directly converted into CNF clauses and added to already obtained CNF clauses from algebraic attack. This time CryptoMiniSAT 5.0 was able give satisfiability/ assignment, on the average, in 28.25 seconds

using 17.26 MBs of memory, for 50 samples, on same machine as used for Trivium above. A summary of results is displayed in fig 4.4 and table 4.2.

Grain family of ciphers are an excellent implementation of stream ciphers based on nonlinear update. In spite of high degree multivariate polynomial equations leading to towering resistance against algebraic cryptanalysis, it was successfully broken by ASCA as shown above. In the first attempt which could not prove successful, partial side channel information through a simulated correlation DPA was not good enough to simplify the solving. Therefore, in second attempt which proved successful, a different side channel leakage information based on hamming weight leakage model through template-like attacks did the trick. This demonstration also highlighted that in case of unsuccessful solving, attacker might choose to revisit any of the constituent attack as explained earlier in fig. 3.4 in chapter 3.

4.4 Conclusion

This chapter proved that ASCA can be highly effective as compared to lone algebraic attack. This was demonstrated by targeting Trivium and Grain stream ciphers which have never been successfully negotiated by traditional algebraic cryptanalysis. Another important aspect of ASCA is that of repeating SCA part with changed leakage points, in case of unsuccessful or computationally infeasible solving, as experienced while attacking Grain.

Chapter 5

CONCLUSION AND FUTURE WORK

5.1 Introduction

In this chapter, the proposed idea of combining algebraic and side channel attacks on stream ciphers is summed up. Mainly the application of ASCA against Crypto-1, Bivium-B, Trivium and Grain v1 stream ciphers, which has been explained in earlier chapters 3 and 4 is concisely analysed. Moreover, few counter measures against the potent threat of ASCA are listed before concluding and mentioning future work possibilities.

5.2 Analysis of Attack Results

For implementation of ASCA, algebraic structure of equations as well as leakage information of a cipher plays an important role. SCA and algebraic attacks against Trivium and Grain can be found in literature. We have given an efficient combination of both thus greatly reducing the attack time complexity for both ciphers. Table 5.1 and 5.2 summarize results of our experiments on Crypto-1, Bivium-B, Trivium and Grain v1 with a comparison with full-blown SCA and lone algebraic attacks respectively.

Table 5.1 highlights that only limited side channel information was extracted for attacked stream ciphers. Traditional SCA targets complete key bits, demanding continued access to the implementation device to acquire more number of power traces. However partial SCA as part of ASCA, needs limited access to the hardware device, only for acquiring single or few traces, as it has an

Table 5.1: Comparison of Full SCA and Partial SCA

Cipher	**Full SCA Reference**	**Partial SCA (this work)**		
		Leakage points	**Leakage model / Attack Type**	**CNF clauses**
Crypto-1	Nil	Input of filter functions	Hamming weight, Template attack, Single trace	1234 (average)
Bivium-B	Nil	1st NLFSR	Known IV, Correlation DPA targeting 1st NLFSR	66 XOR clauses
Trivium	Correlation DPA, Hamming distance model, 550 traces [44]; CPA, Resynchronization phase, chosen IV, 256 traces [67]; Chosen IV, DPA, Resynchronization phase, theoretical [42]	1st NLFSR	Known IV, Correlation DPA targeting 1st NLFSR	66 XOR clauses
Grain v1	Correlation DPA, Hamming distance model, 2600 traces [44]; Chosen IV, DPA, 256 traces per IV [42]	LFSR	Known IV, Correlation DPA targeting LFSR	160 XOR clauses
		Input of function h_i	Hamming weight, Template attack, Single trace	240 XOR clauses

easier target, i.e., few relations translating into algebraic equations or CNF clauses. These supporting equations or clauses make the system of equations/ clauses obtained through algebraic attack, further overdefined and thus simplified.

Table 5.2: Comparison of Lone Algebraic Attacks and ASCA

Cipher	**Best Known Algebraic Attack (seconds)**	**ASCA (this work)**			
		Known Output bits	**Init. Phase Included?**	**Combined CNF: # of Var, Average # of (Algebraic+SCA clauses)**	**Solving Time (seconds)**
Crypto-1	200 [52]	50	No	1541, 24636 + 1234	0.158
Bivium-B	$2^{36.5}$ [47]	80	Yes	3006, 5282 + 66 XOR	11.531
Trivium	$2^{42.5}$, 625 round-reduced version [55]	80	Yes	6975, 12640 + 66 XOR	21.54
Grain v1	$2^{80.7}$ [41]	80	Yes	4549, 16490 + 160 XOR	>3600
		80	Yes	4549, 16490 + 240 XOR	28.25

Table 5.2 elucidates the reduction in complexity of solving, when only few CNF clauses (or algebraic equations) are added into those obtained from lone algebraic attack. The information coming from partial SCA reinforces the algebraic attack to bring down overall attack (ASCA) complexity many times.

5.3 Countermeasures against ASCA

While looking to protect the stream ciphers against ASCA, one has to go deeper into the understanding of constituent attack techniques. Algebraic cryptanalysis and SCA both have been under constant attention of researchers independently as well. The extent to which a stream cipher is resistant to both these attacks individually can give an idea of its resistance against ASCA. Pre-

sumably, resistance to ASCA should be close to the lesser of the resistance to the constituent attacks. In the light of our work presented in this thesis and related research work on ASCA against block ciphers, some counter measures which can make the stream ciphers more resistant against this attack technique, are summed up in ensuing paragraphs.

5.3.1 Countermeasures against SCA

To guard against SCA, the relation between data processing and associated leakage should be randomized or the leakage should be suppressed altogether. Overall such countermeasures can be divided into two types, i.e. algorithm dependent and algorithm independent countermeasures [43].

5.3.1.1 Hardware Oriented Countermeasures

Algorithm independent countermeasures, as the name suggests, are incorporated in the hardware implementation of the cipher:-

- Reducing signal to noise ratio (SNR) by the help of specially designed logic styles suppressing side channel leakage [68]. This would force the attacker to get more number of samples, thereby making the SCA difficult. However use of leakage resistant logic styles leads to almost double sized circuitry and power consumption.

- SNR can also be reduced by generating random noise by adding noisy components [69]. This technique is especially useful in case of lightweight ciphers.

- In ASCA, the partial side channel information can come from anywhere within the cipher implementation. For example, we observed in chapter 3 that in case of Crypto-1 stream cipher, hamming weight leakages were captured at the input of all filter and combiner functions. However, the hamming weight information can be rendered almost useless, if the

size of data against which hamming weight is known, is too big. For instance, if we know hamming weight of a 48 bit register at a particular clock cycle as 35, the algebraic equations or CNF clauses obtained from this hamming weight, as explained in table 3.2, would be too many. Therefore employment of large sized data buses, bigger feed back shift registers, combiner functions with more number of input/ output would make the task of attacker difficult.

5.3.1.2 Software Oriented Countermeasures

These are algorithm dependent countermeasures which are essentially incorporated into the software code of the cipher. Random values are inserted into the code to hide the intermediate values. This is also called masking or blinding. On the other hand, it may be noted that ASCA exploits leakage of any cycle in the implementation [70]. So additional cycles due to insertion of random values to implement masking might decrease the resistance against algebraic cryptanalysis.

5.3.1.3 Additional Countermeasures against Template Attacks

In case of template-like SCA [15], where a single power trace obtained from actual device is good enough for the attacker, all the countermeasures aimed at reducing the number of samples are not viable. However countermeasures to incorporate randomization in computation can mitigate the attack somehow. Template attacks rely on the assumption of possession of a replica device by the adversary. This assumption itself is a weakness of template-like SCA, as admitted by authors proposing the technique [15]. Therefore making use of uncommon micro-controllers might also limit the threat of template attacks.

5.3.2 Countermeasures against Algebraic Cryptanalysis

To ascertain as to how much a cipher is resistant against algebraic cryptanalysis, concept of algebraic immunity comes handy. Stream ciphers based on high degree Boolean functions result into complex MQ problem when subjected to algebraic attack. It must be noted that annihilator functions of lower degree against a high degree Boolean function can also be found by the attacker. The algebraic immunity of a stream cipher is equal to the degree of lowest annihilator function. Higher the value of algebraic immunity, higher would be the resistance of that stream cipher against algebraic attack. Courtois et. al. in [28] gave the concept of algebraic immunity and annihilator functions as under:-

> The algebraic immunity $AI(f)$ of a Boolean function $f()$ with n variables is the lowest degree of any non null function $g()$, called annihilator of $f()$. Annihilator $g()$ of $f()$ can be defined as a function where $f() * g() = 0$ or $(1 + f) * g() = 0$.

For block ciphers a rule of thumb with regards to algebraic attack complexity can be found at [12]:-

$$W.F \cong \Gamma^{\omega}[(Block\ Size).(Number\ of\ Rounds)^2]^{\omega\lceil t/r \rceil} \tag{5.1}$$

Where

$\Gamma = (t/s)^{\lceil t/r \rceil}$ = Real s-box contribution in algebraic cryptanalysis complexity

t = # of monomials in equations of S-box

r = # of equations obtained from s-box

s = # of inputs of s-box

ω = Complexity of Gaussian Elimination = 2.37

In [19], authors redefined the concept of algebraic immunity under the threat of ASCA against block ciphers. They incorporated the extent of side channel leakage from s-boxes into the value of algebraic immunity.

5.4 Future Work

This research work proposes to apply ASCA against stream ciphers for the very first time. Research work on ASCA against block ciphers is under way since 2009. ASCA on block ciphers has become a reasonably mature form of attack and is considered extremely potent threat to cryptosystems. The refinements in ASCA against block ciphers such as its enhanced error tolerance and efficient solving are equally valid for ASCA against stream ciphers. Therefore further research and experimentation in these two areas with regards to stream ciphers is an obvious lead from our work.

Every new attack gives rise to newer safeguards. ASCA against stream ciphers and even against block ciphers, keeping in view its success probability, also invites cryptographers to incorporate countermeasures against it. Therefore, work on safeguarding against ASCA against both block and stream ciphers is yet another open area for research.

5.5 Summing-up of Work

The thesis proposes ASCA against stream ciphers. ASCA is being applied against block ciphers for the last 8 years. Combination of algebraic and side channel attacks in ASCA has the effect of overcoming weaknesses of both attacks. Pure algebraic attacks are computationally infeasible if the cipher's transformation to algebraic equations leads to a complex MQ problem. Pure SCA requires continued access to the cipher and high amount of leakage. On the other hand, ASCA banks on best of both attacks. Algebraic attack phase in ASCA does not try to resolve the key

by solving on its own, till the time it is augmented by some more equations/ clauses coming from partial SCA. SCA phase does not target complete key, rather it captures maximum possible leakage in minimum traces, converts it into equations/ clauses and provides them to algebraic counterpart. Based on this concept, a generic attack methodology was presented in chapter 3.

To demonstrate the efficacy of ASCA on stream ciphers, it was applied against Crypto-1, Bivium-B, Trivium and Grain v1 stream ciphers in chapters 3 and 4 respectively. It can been seen that in application of ASCA, the information coming from SCA can be quite small and can be obtained in few samples. This is unlike traditional SCAs where complete key is targeted with little or no knowledge of algorithm. Central to success of ASCA is selection/ availability of leakage points in the implementation of stream cipher and then judicious utilization of side channel leakage information into MQ problem or SAT problem developed from algebraic attack. This necessitates thorough knowledge of algorithm of target cipher.

The results of experimentation performed in the process of this thesis, were summed up in tables 5.1 and 5.2. The results statistically presented the proof to our proposition that only partial side channel leakage information from cipher's implementation can prove sufficient to efficiently resolve the MQ problem generated through traditional algebraic attack technique. Lastly, few possible countermeasures to curtail the application of ASCA were discussed.

5.6 Conclusion

This chapter sums up the results of ASCA on various stream ciphers as demonstrated during the course of thesis. It also elucidates some known countermeasures against both the constituent attacks of ASCA. Future research directions are also discussed before finally summing up the thesis.

Appendix A

GRAIN OF SALT

A.1 Description

Grain of Salt (GoS) is an automated tool developed by Mate Soos [47] which can transform shift register based stream cipher into conjunctive normal form (CNF) clauses. For algebraic attacks, a stream cipher has to be transformed into algebraic equations and then this system of multivariate equations is solved through any mathematical method. For standard SAT solver-based attacks, these algebraic equations are to be converted into CNF clauses, as the input to SAT solvers is in the form of CNF clauses. Converting algebraic equations manually into CNF clauses is a very tedious task. GoS can do this conversion automatically.

A.2 Input to GoS

For GoS to get into action and generate CNF clauses for a stream cipher, the cipher's description has to be fed to it in the form of a standardized format. The format includes separate configuration files for overall stream cipher's design/ construction, filter functions and combiner/ output functions. Therefore, description files for the target stream cipher has to be prepared first and then fed to GoS. GoS then generates CNF file as output.

A.2.1 Main Configuration File

The main configuration file for the target stream cipher contains information such as number and size of shift registers, whether registers are linear or nonlinear during initialization and normal

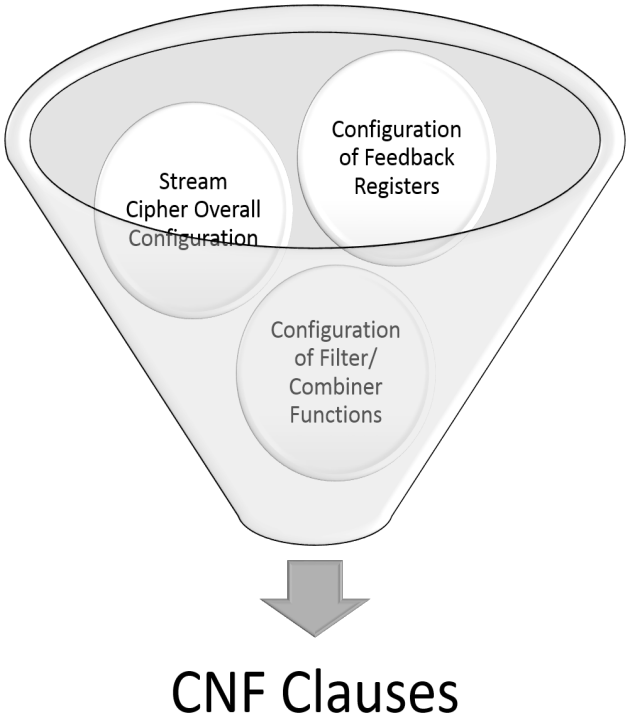

Figure A.1: Grain of Salt (GoS) - Working

functioning, number of filter functions, number of rounds in the initialization phase, which state bits are loaded with IV and which state bits are loaded with 1's.

A.2.2 Feedback Register(s) Configuration File(s)

Separate configuration files for all the feedback registers are to be prepared containing the algebraic expression of feedback functions during initialization phase (if any) and normal working of the cipher.

A.2.3 Filter(s) and Combiner/ Output Functions

Similarly configuration files for all filter functions and combiner/ output functions are to be prepared. These files contain the expression for the respective functions.

A.3 Installation Guide

Source code of GoS is available for download at `https://github.com/msoos/grainofsalt` under GNU General Public Licence. To compile and install GoS following steps are needed:-

- Install *Linux* operating system.
- Install *Cmake*. Cmake can be freely downloaded at `https://cmake.org/download/` and it can be installed by following instructions at `https://cmake.org/install/`.
- Download *Bignum* library from `https://gmplib.org/` and install it with development headers as per instructions provided.
- Download *Boost* library from `http://www.boost.org/` and install it with development headers as per instructions provided.
- Download *zlib* library from `http://zlib.net/` and install it with development headers as per instructions provided.
- After above mentioned installations, issue following commands in sequence:-
  - cd build
  - cmake ../
  - make

A.4 Usage

GoS has a variety of features to analyse stream ciphers. Details of important features can be found in [47] and their usage commands along with various options can be listed through the tool itself by issuing *./grainofsalt –help* as shown in table A.1

Table A.1: Grain of Salt (GoS) - Commands

Options	**Description**
-h [–help]	produce help message
–crypto arg	cryptographic function to simulate. Must have a directory that contains all functions where the binary is executed
–outputs arg	set number of output bits produced
–karnaugh arg	set number of monomials after which karnaugh-table is not used (for pure XOR-s it is never used). Default is 0, i.e. by default karnaugh is not used
–xorcut arg	set maximum length after which the XOR is cut into at least 2 pieces. Default is 7
–noextmonomials	if set, extended monomials will not be used
–init arg	possible values: yes/no. Controls whether the initialisation phase of the ciphers are activated
–base-shift arg	Controls the base shifting. Can only be used to control reference state variables
–deterBits arg	Set thie many of the variables that have been determined to be 'best bits' to use when generating the cipher
–genDeterBits arg	Generate the given number of deterministic bits through a greedy randomised algorithm. NOTE: if the file has already been generated, it will be removed!
–probBits arg	Set this many reference state variables randomly
–xorclauses	Use XOR clauses as per CryptoMiniSat (xors will not be cut)
–seed arg	Seed can be given to generate different CNFs with different runs of the executable. Default is 0
–debug	If set, all CNF-s will be Satisfiable, since the help bits given will all be correct
–num arg	The number of problem instances to generate. Default is 1
–stats	Print statistics to directory 'stats'
–verbose	Print verbose messages. E.g. functions used, bits set,etc.
–cnfDir arg	Put generated CNF files into this directory. By default, it is 'satfiles'
–linearize	If set, linearizeable shift registers will be linearized, i.e. its feedback function will be calculated from the same set of reference state bits. Default is not to do this.
–permutateVars	If set, variables will be permutated in the generated CNF
–permutateClauses	If set, clauses will be permutated in teh generated CNF(s)
–nopropagate	If set, facts will not be propagated at the ANF level. Expect slower solving.

For example following command can be used to generate 50 CNF files including XOR clauses,

against target stream cipher, which is defined through description files placed in a directory named *grain*, with a pre-requisite that 80 output stream bits are known and initialization phase of the cipher is included:-

```
./grainofsalt --crypto grain --outputs 80 --init yes --num 50
--xorclauses
```

- **–crypto grain** option means that the target stream cipher's description will be picked up from directory *grain*.
- **–outputs 80** option means that 80 bits of output stream are known.
- **–init yes** means that initialization phase of the target cipher will be included in the algebraic attack. Therefore corresponding CNF files will include clauses from initialization phase.
- **–num 50** option means that 50 CNF files against as many different input sets will be generated.
- **–xorclauses** implies that XOR clauses will be included in the CNF files. This is useful only if the SAT solver being used can handle XOR clauses.

Appendix B

CRYPTOMINISAT 5.0

B.1 Description

CryptoMiniSAT 5.0 is a SAT solver, developed by Mate Soos [61]. Like other SAT solvers, it takes CNF clauses or file in DIMACS[1] format and returns either satisfiability/ assignment to variables or unsatisfiability. As an exception, CryptoMiniSAT 5.0 can also take XOR clauses as input.

B.2 Installation Guide

Source code of CryptoMiniSAT 5.0 can be downloaded from `https://github.com/msoos/cryptominisat` under MIT licence. The code needs to be compiled on Linux operating system as per the accompanying instructions. *Cmake* and *boost library* should be installed before compilation. For building/ installing following sequence of action should be followed:-

- Extract the downloaded zip file containing source code.
- Move into the extracted directory.
- Issue command *cmake*.
- Issue command *make*.
- Issue command *make install* as super user.

[1] http://www.satcompetition.org/2009/format-benchmarks2009.html

- Issue command *ldconfig* as super user.

B.3 Usage

Typical usage of CryptoMiniSAT 5.0 is depicted in fig B.1. The input CNF file contains 3 variables '1', '2', '3' and 3 clauses. When this CNF file is fed to CryptoMiniSAT 5.0 with following command:-

```
cryptominisat5  filename.cnf
```

The solver processes the CNF clauses and in this case returns that the expression is satisfiable with assignment to variables.

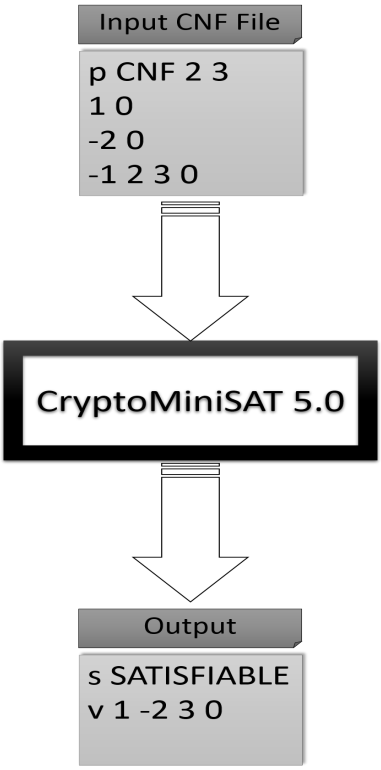

Figure B.1: CryptoMiniSAT - Working

BIBLIOGRAPHY

[1] B. Schneier, *Applied cryptography: protocols, algorithms, and source code in C* (john wiley & sons, 2007).

[2] A. Bogdanov, L. R. Knudsen, G. Leander, C. Paar, A. Poschmann, M. J. Robshaw, Y. Seurin, and C. Vikkelsoe, "PRESENT: An ultra-lightweight block cipher," In *International Workshop on Cryptographic Hardware and Embedded Systems*, pp. 450–466 (2007).

[3] C. E. Shannon, "Communication theory of secrecy systems," Bell Labs Technical Journal **28,** 656–715 (1949).

[4] H. Wu, "Cryptanalysis and design of stream ciphers," A PhD thesis of Katholieke Universiteit Leuven, Belgium (2008).

[5] A. Klimov and A. Shamir, "Cryptographic applications of T-functions," In *International Workshop on Selected Areas in Cryptography*, pp. 248–261 (2003).

[6] J. D. Golić, "Cryptanalysis of alleged A5 stream cipher," In *International Conference on the Theory and Applications of Cryptographic Techniques*, pp. 239–255 (1997).

[7] E. Berlekamp, "Algorithmic Coding Theory,", 1968.

[8] T. Siegenthaler, "Correlation-immunity of nonlinear combining functions for cryptographic applications (Corresp.)," IEEE Transactions on Information theory **30,** 776–780 (1984).

[9] W. Meier and O. Staffelbach, "Fast correlation attacks on stream ciphers," In *Workshop on the Theory and Application of of Cryptographic Techniques*, pp. 301–314 (1988).

[10] A. Kipnis and A. Shamir, "Cryptanalysis of the HFE public key cryptosystem by relinearization," In *Annual International Cryptology Conference*, pp. 19–30 (1999).

[11] N. Courtois, A. Klimov, J. Patarin, and A. Shamir, "Efficient algorithms for solving overdefined systems of multivariate polynomial equations," In *International Conference on the Theory and Applications of Cryptographic Techniques*, pp. 392–407 (2000).

[12] N. T. Courtois and J. Pieprzyk, "Cryptanalysis of block ciphers with overdefined systems of equations," In *International Conference on the Theory and Application of Cryptology and Information Security*, pp. 267–287 (2002).

[13] B. Buchberger and F. Winkler, *Gröbner bases and applications* (Cambridge University Press, 1998), Vol. 251.

[14] M. Renauld and F.-X. Standaert, "Algebraic side-channel attacks," In *International Conference on Information Security and Cryptology*, pp. 393–410 (2009).

[15] S. Chari, J. R. Rao, and P. Rohatgi, "Template attacks," In *International Workshop on Cryptographic Hardware and Embedded Systems*, pp. 13–28 (2002).

[16] M. Renauld, F.-X. Standaert, and N. Veyrat-Charvillon, "Algebraic side-channel attacks on the AES: Why time also matters in DPA," in *Cryptographic Hardware and Embedded Systems-CHES 2009* (Springer, 2009), pp. 97–111.

[17] Y. Oren, M. Kirschbaum, T. Popp, and A. Wool, "Algebraic side-channel analysis in the presence of errors," In *International Workshop on Cryptographic Hardware and Embedded Systems*, pp. 428–442 (2010).

[18] X. Zhao, F. Zhang, S. Guo, T. Wang, Z. Shi, H. Liu, and K. Ji, "MDASCA: an enhanced algebraic side-channel attack for error tolerance and new leakage model exploitation," In *International Workshop on Constructive Side-Channel Analysis and Secure Design*, pp. 231–248 (2012).

[19] C. Carlet, J.-C. Faugere, C. Goyet, and G. Renault, "Analysis of the algebraic side channel attack," Journal of Cryptographic Engineering **2,** 45–62 (2012).

[20] M. S. E. Mohamed, S. Bulygin, M. Zohner, A. Heuser, M. Walter, and J. Buchmann, "Improved algebraic side-channel attack on AES," In *Hardware-Oriented Security and Trust (HOST), 2012 IEEE International Symposium on*, pp. 146–151 (2012).

[21] Y. Oren, M. Renauld, F.-X. Standaert, and A. Wool, "Algebraic side-channel attacks beyond the hamming weight leakage model," In *International Workshop on Cryptographic Hardware and Embedded Systems*, pp. 140–154 (2012).

[22] Y. Oren and A. Wool, "Tolerant Algebraic Side-Channel Analysis of AES.," IACR Cryptology ePrint Archive **2012,** 92 (2012).

[23] X. Zhao, T. Wang, S. Guo, F. Zhang, Z. Shi, H. Liu, and K. Wu, "SAT based error tolerant algebraic side-channel attacks," In *2011 Conference on Cryptographic Algorithms and Cryptographic Chips, CASC*, (2011).

[24] Y. Oren, O. Weisse, and A. Wool, "Practical template-algebraic side channel attacks with extremely low data complexity," In *Proceedings of the 2nd International Workshop on Hardware and Architectural Support for Security and Privacy*, p. 7 (2013).

[25] L. Song, L. Hu, S. Sun, Z. Zhang, D. Shi, and R. Hao, "Error-Tolerant Algebraic Side-Channel Attacks Using BEE," In *International Conference on Information and Communications Security*, pp. 1–15 (2014).

[26] Y. Oren and A. Wool, "Side-channel cryptographic attacks using pseudo-boolean optimization," Constraints **21,** 616–645 (2016).

[27] N. T. Courtois, "Higher order correlation attacks, XL algorithm and cryptanalysis of Toyocrypt," In *International Conference on Information Security and Cryptology*, pp. 182–199 (2002).

[28] N. T. Courtois, "Fast algebraic attacks on stream ciphers with linear feedback," In *Annual International Cryptology Conference*, pp. 176–194 (2003).

[29] P. C. Kocher, "Timing attacks on implementations of Diffie-Hellman, RSA, DSS, and other systems," In *Annual International Cryptology Conference*, pp. 104–113 (1996).

[30] A. P. Chandrakasan and R. W. Brodersen, "Minimizing Power Consumption in CMOS Circuits, 1995,".

[31] N. Courtios and W. Meier, "Algebraic attacks on stream ciphers with linear feedback," Advances in Cryptolo-gy-EUROCRYPT pp. 346–359 (2003).

[32] F. Armknecht, "Improving fast algebraic attacks," In *International Workshop on Fast Software Encryption*, pp. 65–82 (2004).

[33] F. Armknecht and M. Krause, "Algebraic attacks on combiners with memory," In *Annual International Cryptology Conference*, pp. 162–175 (2003).

[34] N. T. Courtois, "Algebraic attacks on combiners with memory and several outputs," In *International Conference on Information Security and Cryptology*, pp. 3–20 (2004).

[35] A. Braeken, J. Lano, N. Mentens, B. Preneel, and I. Verbauwhede, "SFINKS: A synchronous stream cipher for restricted hardware environments," In *SKEW-Symmetric Key Encryption Workshop*, **55,** 72 (2005).

[36] Y. Nawaz and G. Gong, "The WG stream cipher," ECRYPT Stream Cipher Project Report 2005 33 (2005).

[37] "The eSTREAM Project,", http://www.ecrypt.eu.org/stream/project.html.

[38] M. Afzal and A. Masood, "Resistance of stream ciphers to algebraic recovery of internal secret states," In *Convergence and Hybrid Information Technology, 2008. ICCIT'08. Third International Conference on*, **2,** 625–630 (2008).

[39] S. Al Hinai, L. M. Batten, and B. Colbert, "Mutually clock-controlled feedback shift registers provide resistance to algebraic attacks," In *International Conference on Information Security and Cryptology*, pp. 201–215 (2007).

[40] P. Datta, D. Roy, and S. Mukhopadhyay, "A probabilistic algebraic attack on the grain family of stream ciphers," In *International Conference on Network and System Security*, pp. 558–565 (2014).

[41] M. Afzal and A. Masood, "Algebraic cryptanalysis of a nlfsr based stream cipher," In *Information and Communication Technologies: From Theory to Applications, 2008. ICTTA 2008. 3rd International Conference on*, pp. 1–6 (2008).

[42] W. Fischer, B. M. Gammel, O. Kniffler, and J. Velten, "Differential power analysis of stream ciphers," In *Cryptographers Track at the RSA Conference*, pp. 257–270 (2007).

[43] C. Rechberger and E. Oswald, "Stream ciphers and side-channel analysis," In *In ECRYPT Workshop, SASC-The State of the Art of Stream Ciphers*, pp. 320–326 (2004).

[44] D. Strobel, I. C. Paar, and M. Kasper, "Side channel analysis attacks on stream ciphers," Masterarbeit Ruhr-Universität Bochum, Lehrstuhl Embedded Security (2009).

[45] B. Gierlichs *et al.*, "Susceptibility of eSTREAM candidates towards side channel analysis," Proceedings of SASC pp. 123–150 (2008).

[46] A. R. Kazmi, M. Afzal, M. F. Amjad, and A. Rashdi, "Combining Algebraic and Side Channel Attacks on Stream Ciphers," In *International Conference on Communication Technologies*, p. in press (2017).

[47] M. Soos, "Grain of saltan automated way to test stream ciphers through SAT solvers," In *Tools*, **10,** 131–144 (2010).

[48] C. McDonald, C. Charnes, and J. Pieprzyk, "An algebraic analysis of Trivium ciphers based on the boolean satisfiability problem," In *Proceedings of the 4th International Workshop on Boolean Functions: Cryptography and Applications*, pp. 173–184 (2008).

[49] P. Kocher, J. Jaffe, and B. Jun, "Differential power analysis," In *Annual International Cryptology Conference*, pp. 388–397 (1999).

[50] J. Lano, N. Mentens, B. Preneel, and I. Verbauwhede, "Power analysis of synchronous stream ciphers with resynchronization mechanism," In *ECRYPT Workshop, SASC–The State of the Art of Stream Ciphers*, pp. 327–333 (2004).

[51] C. Rechberger, *Side channel analysis of stream ciphers* (na, 2004).

[52] N. Courtois, K. Nohl, and S. O'Neil, "Algebraic Attacks on the Crypto-1 Stream Cipher in MiFare Classic and Oyster Cards.," IACR Cryptology ePrint Archive **2008,** 166 (2008).

[53] K. Nohl, "Cryptanalysis of crypto-1," Computer Science Department University of Virginia, White Paper (2008).

[54] C. D. Canniere and B. Preneel, "Trivium specifications," citeseer. ist. psu. edu/734144. html (2005).

[55] F.-M. Quedenfeld and C. Wolf, "Advanced Algebraic Attack on Trivium," In *International Conference on Mathematical Aspects of Computer and Information Sciences*, pp. 268–282 (2015).

[56] H. Raddum, "Cryptanalytic results on Trivium," eSTREAM, ECRYPT Stream Cipher Project, Report **39,** 2006 (2006).

[57] T. E. Schilling and H. Raddum, "Analysis of trivium using compressed right hand side equations," In *International Conference on Information Security and Cryptology*, pp. 18–32 (2011).

[58] I. Simonetti, J.-C. Faugere, and L. Perret, "Algebraic attack against trivium," In *First International Conference on Symbolic Computation and Cryptography, SCC*, **8,** 95–102 (2008).

[59] S.-G. Teo, K. K.-H. Wong, H. Bartlett, L. Simpson, and E. Dawson, "Algebraic analysis of Trivium-like ciphers (Poster)," In *Proceedings of the Twelfth Australasian Information Security Conference-Volume 149*, pp. 77–81 (2014).

[60] M. Afzal and A. Masood, "Experimental results on algebraic analysis of trivium and tweaked trivium," in *Global E-Security* (Springer, 2008), pp. 93–101.

[61] M. Soos, "CryptoMiniSat–a SAT solver for cryptographic problems," URL http://www. msoos. org/cryptominisat4 (2009).

[62] T. M. Cover and J. A. Thomas, *Elements of information theory* (John Wiley & Sons, 2012).

[63] M. Hell, T. Johansson, A. Maximov, and W. Meier, "The Grain family of stream ciphers," in *New Stream Cipher Designs* (Springer, 2008), pp. 179–190.

[64] S. Banik, S. Maitra, and S. Sarkar, "A differential fault attack on the grain family of stream ciphers," In *International Workshop on Cryptographic Hardware and Embedded Systems*, pp. 122–139 (2012).

[65] S. Karmakar and D. R. Chowdhury, "Fault analysis of Grain-128 by targeting NFSR," In *International Conference on Cryptology in Africa*, pp. 298–315 (2011).

[66] H. Zhang and X. Wang, "Cryptanalysis of Stream Cipher Grain Family.," IACR Cryptology ePrint Archive **2009,** 109 (2009).

[67] Y. Jia, Y. Hu, F. Wang, and H. Wang, "Correlation power analysis of Trivium," Security and Communication Networks **5,** 479–484 (2012).

[68] K. Tiri, M. Akmal, and I. Verbauwhede, "A dynamic and differential CMOS logic with signal independent power consumption to withstand differential power analysis on smart cards," In *Solid-State Circuits Conference, 2002. ESSCIRC 2002. Proceedings of the 28th European*, pp. 403–406 (2002).

[69] C. Clavier, J.-S. Coron, and N. Dabbous, "Differential power analysis in the presence of hardware countermeasures," In *Cryptographic Hardware and Embedded SystemsCHES 2000*, pp. 13–48 (2000).

[70] M. Renauld and F.-X. Standaert, "Combining algebraic and side-channel cryptanalysis against block ciphers," In *30th Symposium on Information Theory in the Benelux*, pp. 97–104 (2009).