Analysis of Block Cipher Constructions against Biclique and Multiset Attacks

By
Mohona Ghosh

Indraprastha Institute of Information Technology, Delhi
(IIIT-Delhi)

Supervisors: Dr. Somitra Sanadhya
Dr. Donghoon Chang

January, 2016

Analysis of Block Cipher Constructions against Biclique and Multiset Attacks

By
Mohona Ghosh

Submitted
in partial fulfillment of the requirements for the degree of
Doctor of Philosophy
in Computer Science & Engineering

to the

Indraprastha Institute of Information Technology, Delhi
January, 2016

Certificate

This is to certify that the thesis titled - **"Analysis of Block Cipher Constructions against Biclique and Multiset Attacks"** being submitted by **Mohona Ghosh** to Indraprastha Institute of Information Technology, Delhi, for the award of the degree of Doctor of Philosophy, is an original research work carried out by her under our supervision. In our opinion, the thesis has reached the standards fulfilling the requirements of the regulations relating to the degree.

The results contained in this thesis have not been submitted in part or full to any other university or institute for the award of any degree/diploma.

Dr. Somitra Sanadhya

Dr. Donghoon Chang

January, 2016
Department of Computer Science
Indraprastha Institute of Information Technology, Delhi
New Delhi, 110020

Acknowledgments

"It takes a big heart to help shape little minds" - Unknown

First and foremost, I express my heartfelt gratitude to my esteemed teacher and guide, Dr. Somitra Sanadhya, my inspiration, for his invaluable guidance, constant support and for the intellectual stimulation given, which I will cherish in my heart always. His constant oasis of ideas exceptionally enrich the learner's thought process. I am extremely fortunate to have him as my advisor.

I also express my sincere gratitude to my esteemed co-advisor, Dr. Donghoon Chang, who has helped me immensely throughout my PhD. life. All of the research I have conducted in completing my thesis wouldn't have been possible without his vision, encouragement and support.

I would like to thank my PhD examiners, Dr. Vincent Rijmen, Dr. Sourav Mukhopadhyay and Dr. Jiageng Chen for their valuable comments and suggestions which helped me improve my dissertation.

I am grateful to Dr. Andrey Bogdanov for the inspiring and fruitful collaboration I had with him. His passion for cryptanalysis and the perseverance of always pushing the results to a higher standard will always inspire me in my future research pursuits.

I take this opportunity to thank all the members of my Crypto Lab. The resources and environment provided by them really helped me in my research. They have made my research life at IIIT-Delhi less tensed.

I would like to thank all my friends from IIIT-Delhi, especially Sweta, Monalisa, Monika, Megha, Madhvi, Madhur, Jyoti and Tarun for their help and cooperation that always kept my spirits high. Their company made my graduate life a memorable one.

I had a great opportunity to closely work with some brilliant undergraduate students: Akshima and Aarushi Goel.

I would also like to forward my gratitude to Tata Consultancy Services (TCS), India for awarding me the prestigious TCS fellowship for my full PhD period.

On a personal front , I owe my heartfelt thanks to my parents, my uncle and my sister for their unconditional support, understanding and encouragement without which this dissertation would not have got completed.

Above all, I am grateful to the Almighty, who showered the opportunity, blessings and moral courage on me to complete this dissertation.

List of Publications

The author names are in the alphabetical order.

1. Andrey Bogdanov, Donghoon Chang, **Mohona Ghosh**, and Somitra Kumar Sanadhya. Bicliques with Minimal Data and Time Complexity for AES. In Jooyoung Lee and Jongsung Kim, editors, *Information Security and Cryptology - ICISC 2014 - 17th International Conference, Seoul, Korea, December 3-5, 2014, Revised Selected Papers, volume 8949 of Lecture Notes in Computer Science, pages 160-174. Springer, 2014.*

2. Megha Agrawal, Donghoon Chang, **Mohona Ghosh**, and Somitra Kumar Sanadhya. Collision attack on 4-branch, type-2 GFN based hash functions using Sliced Biclique Cryptanalysis Technique. In Dongdai Lin, Moti Yung, and Jianying Zhou, editors, *Information Security and Cryptology - 10th International Conference, Inscrypt 2014, Beijing, China, December 13-15, 2014, Revised Selected Papers, volume 8957 of Lecture Notes in Computer Science, pages 343-360. Springer, 2014.*

3. Donghoon Chang, **Mohona Ghosh**, and Somitra Kumar Sanadhya. Biclique Cryptanalysis of full round AES-128 based hashing modes. In Dongdai Lin, Moti Yung, and Xiaofeng Wang, editors, Information Security and Cryptology - 11th International Conference, Inscrypt 2015, Beijing, China, November 1-3, 2015, Revised Selected Papers, volume 9589 of Lecture Notes in Computer Science. Springer, 2015.

4. Akshima, Donghoon Chang, **Mohona Ghosh**, Aarushi Goel, and Somitra Kumar Sanadhya. Single Key Recovery Attacks on 9-round Kalyna-128/256 and Kalyna-256/512. In Soonhak Kwon and Aaram Yun, editors, Information Security and Cryptology - ICISC 2015 - 18th International Conference, Seoul, Korea, November 25-27, 2015, Revised Selected Papers, volume 9558 of Lecture Notes in Computer Science. Springer, 2015.

5. Akshima, Donghoon Chang, **Mohona Ghosh**, Aarushi Goel, and Somitra Kumar Sanadhya. Improved Meet-in-the-Middle Attacks on 7 and 8-Round ARIA-192 and ARIA-256. In Alex Biryukov and Vipul Goyal, editors, *Progress in Cryptology - INDOCRYPT 2015 - 16th International Conference on Cryptology in India, Bangalore, India, December 6-9, 2015, Proceedings, volume 9462 of Lecture Notes in Computer Science, pages 198-217. Springer, 2015.*

Abstract

Cryptographic protocols have been a cornerstone of secure communications among armed forces and diplomatic missions since time immemorial. With easy availability and low cost of computing facilities and Internet, the domain of cryptology has not only expanded to non-government uses but also in fulfilling the common needs of individuals. Block ciphers are the basic building blocks of most of today's deployed cryptography and are one of the most widely used cryptographic primitives. They play a crucial role in providing *confidentiality* of data transmitted over insecure communication channels - one of the fundamental goals of cryptography. Apart from it, block ciphers are also used to build other cryptographic mechanisms such as - Hash functions and Message Authentication Codes. Hence, it is crucial to ensure construction of a secure and robust block cipher design. To achieve so, it is imperative to analyze and evaluate the resistance of block ciphers against a variety of cryptanalytic attacks.

This thesis is devoted to the security analysis of block ciphers and block cipher based hash functions against some of the current state-of-the-art cryptanalytic techniques. We specifically focus on *Biclique Cryptanalysis* and *Multiset Attacks* in this work. We propose a new extension of biclique technique - termed as *Star based Bicliques* and use them to solve the problem of high data complexity usually associated with this technique. Further, we also employ the above cryptanalytic methods to provide the best attacks on few standardized block ciphers. Our cryptanalytic results are as follows:

1. We study biclique based key recovery attacks and find improvements that lower the attack costs compared to the original attack in [39]. These attacks are applied to full round AES-128 (10-rounds), AES-192 (12-rounds) and AES-256 (14-rounds) with interesting observations and results. As part of the results, we propose star-based bicliques which allow us to launch attacks with the minimal data complexity in accordance with the unicity distance. Each attack requires just 2-3 known plaintexts with success probability 1.
2. We utilize the biclique based key recovery attacks to find second-preimages on AES based hashing modes. In our attacks, the initialization vector (IV) is a public constant that cannot be changed by the attacker. Under this setting, with message padding restrictions, the biclique trails constructed for key recovery attack in [39] cannot be utilized here. We construct new biclique trails that satisfy the above restrictions and launch second preimage attacks on all 12 PGV hashing modes based on full round AES-128.
3. We investigate the security of Generalized Feistel Networks (GFNs) in known-key scenario. We apply a variant of biclique technique - termed as sliced biclique cryptanalysis on 4-branch, Type-2 Generalized Feistel Networks (GFNs) based hash functions to generate actual collisions. We further demonstrate the best 8-round collision attack on 4-branch, Type-2 based GFNs when the round function F is instantiated with double SP layers.

4. We analyze the security of Korean Encryption Standard ARIA against meet-in-the-middle attack model. We conduct multiset based key recovery attacks on 7 and 8-round ARIA-192 and ARIA-256 with improved time, memory and data complexities compared to [168]. While the previous attacks on ARIA could only recover some round keys, our attacks show the first recovery of the complete master secret key.
5. We analyze the security of recently announced Ukrainian Encryption Standard Kalyna against meet-in-the-middle attack model. We apply multiset attacks supplemented with further related advancements in this attack technique to recover the secret key from 9-round Kalyna-128/256 and Kalyna-256/512. This improves upon the previous best attack reported in [13] in terms of number of rounds attacked by 2.

In terms of either the attack complexity or the number of attacked rounds, the attacks presented in the thesis are better than any previously published cryptanalytic results for the block ciphers concerned.

Contents

List of Figures

List of Tables

Glossary

AES	:	Advanced Encryption Standard
DES	:	Data Encryption Standard
IV	:	Initialization Vector
CV	:	Chaining Value
MAC	:	Message Authentication Code
NIST	:	National Institute of Standards and Technology
PGV modes	:	Block cipher based hash modes named after Preneel, Govaerts, and Vandewalle
MMO	:	Matyas-Meyer-Oseas mode
MP	:	Miyaguchi Preneel mode
DM	:	Davies Meyer mode
SP	:	Substitution Permutation
FN	:	Feistel Network
GFN	:	Generalized Feistel Network
$\circ$	:	Superposition
$\oplus$	:	XOR operation
$\boxplus$	:	Addition modulo 2^n
$\ll$ ($\gg$)	:	Left (right) shift of a bit string
$\lll$ ($\ggg$)	:	Left (right) rotation of a bit string
$\|\|$	:	String Concatenation
ISO	:	International Organization for Standardization
GSM	:	Global System for Mobile Communications
NSA	:	National Security Agency
LFSR	:	Linear Feedback Shift Register
SBL	:	Single Block Length
DBL	:	Double Block Length
GF	:	Galois Field
MDS	:	Maximum Distance Separable
CC	:	Chosen Ciphertext
CP	:	Chosen Plaintext
MITM	:	Meet-in-the-Middle
SSB	:	Super S-box

Chapter 1

Introduction

The word *cryptography* and the associated word *cryptology* have very similar etymological origins. They are derived from the Greek words *kriptos*, which means "hidden"; *graphos*, which translates to "writing"; and *logos*, which is "word" or "speech". Cryptology is the science and art of secret communications. Since the time of Julius Caesar and even before, people have protected the confidentiality of their communications by cryptography. Historically, cryptology has been used by diplomatic missions and armed forces [161]. However, in the modern era, with easy availability and low cost of computing facilities and Internet, the domain of cryptology has not only expanded to non-government uses but also in fulfilling the common needs of the individuals. Today, cryptology plays a fundamental role - in securing access to Internet banking, secure login to websites, secure e-commerce, protecting the integrity of data online, secure computations on clouds etc. People continue to use cryptography though far more sophisticated than Caesar's, to protect their vital information as it passes through possibly hostile environments. Cryptology typically forms a small but important component under the wide umbrella of security solutions with main focus on the following functionalities [110]:

- Confidentiality - Ensures protection of data and resources from leaking to unauthorized listeners over an insecure communication channel.
- Data Integrity - Ensures that an adversary who has access to the communication channel cannot modify the contents of transmitted data by improper or unauthorized means.
- Authenticity - Ensures that the data received over an insecure channel has been sent by the authorized sender and not by an undesired source.

With advancements in technology in which cryptographic solutions are being deployed, increasing by leaps and bounds, it is of utmost importance to understand and analyze the cryptographic designs and protocols with rigor so that security in terms of confidentiality, data authenticity, access control and privacy is maintained. These

advancements have also led to a renewed interest in the study of cryptographic applications in other fields such as biometrics, ubiquitous computing, mobile networks etc. and forged global cooperation and collaboration between research communities and IT industries world over.

Cryptology broadly comprises of two types of studies - *cryptography* and *cryptanalysis*. While cryptography deals with the design of mechanisms providing certain security goals, cryptanalysis focuses on analyzing these designs with the aim of finding some flaw/weakness in them and violate the security goals. The most common cryptographic techniques fall under two categories - secret-key (symmetric) cryptography and public-key (asymmetric) cryptography. In secret-key cryptography, each pair of communicating parties requires the knowledge of one common key which is kept secret from other unauthorized parties. The sender uses the secret key to encrypt the message (plaintext) whereas the receiver uses the same secret key to decrypt the message (ciphertext). In public-key cryptography, introduced by Diffie and Hellman [65] in 1976, each participating party has a pair of keys, termed as public and private key respectively that are used for secure communication. The sender uses the public key of the receiver to encrypt the message while the receiver uses her private key to decrypt the message. In either case, only the authorized members can recover the valid data from the ciphertext, for the rest, this data appears illegible. Examples of public key cryptosystems are RSA [146], elliptic curve cryptosystems [112] etc. The security of public key cryptography is established on the computational hardness assumption that deducing the private key from public key is extremely difficult. On the other hand, symmetric key cryptography rests its security on the size of the secret key used. The larger the size, the higher is the security.

The basic building blocks of symmetric key cryptography are - *block ciphers*, *stream ciphers* and *message authentication codes (MACs)*. Block ciphers and stream ciphers provide data confidentiality by encrypting plaintext into ciphertext with the help of the secret key. MACs provide data integrity and authentication. The sender computes an *authentication tag* for some message (as a function of message and the secret key) and sends it to the receiver together with the message. The receiver then checks the validity of the message by computing the authentication tag (as a function of the received message and same secret key). If the authentication tag calculated by the receiver coincides with the one obtained from the communication channel, the message is accepted else discarded. *Cryptographic hash functions* are another set of primitives which provide functionality akin to MACs. They take a string of arbitrary length as input and produce a fixed length output called *hash*. Strictly speaking, they do not belong to symmetric key algorithms since they do not accept a key. However, many hash function designs are inspired from block ciphers. As such, many cryptographic techniques which were initially developed for analyzing block ciphers are applicable to hash functions as well and vice versa. Hence, we include hash functions under the belt of symmetric cryptographic primitives. In this report, we concentrate on cryptanalysis

of block ciphers and constructions based on them.

1.1 Motivation

The area of block cipher cryptanalysis, as discussed above, focuses on finding flaws in a block cipher design with the ultimate aim of breaking it and disclaiming the security assurances of its designer. Breaking a cipher doesn't always mean finding a practical way for an attacker to recover the secret key or a plaintext from its given ciphertext. In academic cryptography, breaking a cipher simply means finding a weakness in the cipher that can be exploited with a complexity less than exhaustive search. Sometimes these breaks may require an unrealistic amount of time, data and memory. However, as Bruce Schneier has said - "a break can just be a *certificational weakness*: evidence that the cipher does not perform as advertised" [154].

To achieve these goals, a number of cryptanalytic techniques have been developed and research in cryptanalysis field is growing fast. Spearheaded by differential attacks [28] and linear cryptanalysis [131], many attack techniques such as square attack [57, 79], boomerang attack [173], impossible differential attack [23, 126, 140], related key attack [25, 26, 31], meet-in-the-middle attack [66, 72] etc. have been proposed and presented in literature. These attacks in turn have led to the development and evolution of security criteria for the evaluation of block ciphers, e.g., size of the key required, size of the data block processed, number of rounds in a construction, inclusion of non-linear functions in the design etc. In fact, a block cipher is not considered secure until it has withstood certain threshold cryptanalysis. The trust in its security increases when it shows continued resistance over a stretch of time (usually in years), even against advances in cryptanalysis that were not previously conceived. It is thus of great importance to investigate the security of a block cipher algorithm against a variety of cryptanalytic attacks. Two cryptanalysis techniques - *Biclique Cryptanalysis* and *Multiset Attacks* till present have been known to yield the best cryptanalytic results on Advanced Encryption Standard (AES) and form the specific focus of this thesis.

AES, standardized by the US NIST in October 2000, has been accepted and adopted worldwide thereafter. It remains the most favored cryptographic scheme in both software and hardware applications. Despite the design having been subjected to tremendous scrutiny in the past 15 years, until 2011, it has remained remarkably immune to all cryptanalytic attacks. In Asiacrypt, 2011, *biclique cryptanalysis* for AES was proposed and it was the first technique in single key model that challenged the 128-bit security of full 10-round AES-128 block cipher. The application of this technique showed that for full AES (all 3 AES variants), the key can be recovered with a complexity lesser than brute-force by a factor of 3-5 [39]. Since then, biclique cryptanalysis technique has been adopted to attack many other block ciphers such as ARIA [52], SQUARE [129], TWINE [53], HIGHT [86], PRESENT [4, 92] etc. The advantage of this technique

is that it is generic, i.e., it can be applied to most of the block ciphers as long as a few conditions are met. However, the attack technique has some inherent limitations - high time and data complexity. Soon after the initial excitement, crypto purists started comparing it with brute force as the standard brute force is very likely to be both cheaper and faster in reality. Hence, the question of possibilities in reduction of attack complexities started surfacing. Moreover, although the biclique technique has been used to analyze many block ciphers, no formal work except [99] was available in the literature which analyzed the security of block cipher based hash functions against the biclique attack. In [39], Bogdanov et al. showed the conversion of biclique based key recovery attack on AES to preimage attack on AES based compression function. However, translation of this preimage attack on compression function to hash function is not trivial. This is because biclique attacks consist of finding some biclique trails which are then used to recover the secret key. In all of the existing biclique attacks, the biclique trails allow modifications to the key as well as the message input to the block cipher. However, in the hash function settings, we are usually interested in the known-key scenario, where the key input to the block cipher is known publically. Typical examples include Miyaguchi-Preneel mode and Matyas-Meyer-Oseas mode [34,141]. In these modes, a fixed value (i.e., the IV) that is not under the attacker's control and cannot be changed by her is fed as the key input to the block cipher. Moreover, in the subsequent iterations of these modes, [1] it is difficult for an attacker to control the value of the chaining variable (produced as the output of each iteration) as well. Hence, such settings warrant construction of new biclique trails that satisfy the fixed IV (and known CV) restriction. These are some of the problems which we attempt to solve in this thesis.

Leaving aside the biclique attacks, the next best class of attack on AES in the secret key model is the *Multiset Attack*. With the recent advancements in this technique in the past 2-3 years [61, 71, 122, 147], the underlying attacks can be considered the most efficient attacks on AES so far (in terms of lowest computational complexity). Though this line of attacks has been extensively studied for AES research [57, 60, 61, 71, 79, 122, 147], their application to other block ciphers has not been investigated much yet. One of the possible reasons could be that many of these attacks exploit some specific design properties such as - weak key schedule of AES. In AES, recovery of a subkey allows recovery of all the other subkeys as well as the master key. However, this property does not hold true for other block ciphers. A case in point is the Korean Encryption Standard *ARIA* [114] and the Ukrainian Encryption Standard *Kalyna* [138]. The designs of both of these block ciphers are inspired from AES, however they both have stronger key schedule algorithms. After 2010, no formal security analysis of ARIA exists in literature. Moreover, none of the existing attacks on ARIA have been able to recover the actual secret key. Kalyna has recently been announced as Ukrainian Encryption Standard in June, 2015. It supports 3 block and key sizes: 128-bit, 256-

[1] in iterated hash functions

bit and 512-bit. According to the notations followed in [138], Kalyna-128 includes all 3 variants of Kalyna with block size of 128-bits and key size of 128, 256 and 512-bits. Similarly, Kalyna variants with block size of 256-bit and all 3 key sizes are commonly termed as Kalyna-256. The designers of Kalyna claim Kalyna-128 to be resistant against all types of cryptanalysis techniques when the number of rounds is $\geq$ 6 [1]. For Kalyna-256, the minimum number of rounds for which it is resistant to all types of cryptanalysis techniques is claimed to be 7. However, no justification of these claims has been provided. Coupled with the fact, that Kalyna has not yet received significant attention from the cryptanalysis community, motivated us to analyze its security against multiset attacks. In our works, we investigate the effectiveness of multiset attacks against both ARIA and Kalyna and improve the current best known cryptanalytic results on both of them.

1.2 Thesis Organization

In this thesis, we present improvements in some cryptanalytic attack algorithms and results on cryptanalysis of a few block ciphers and cryptographic hash functions constructed from block ciphers. Our main results are presented in Chapters 3, 4, 5, 6 and 7. Each chapter provides literature review, followed by cryptanalysis results and conclusions. The thesis outline is as follows:

- In Chapter 2, we introduce block ciphers and hash functions. We review a number of currently known cryptanalytic methods for block ciphers that will facilitate the understanding of the work discussed in subsequent chapters.

- In Chapter 3, we re-evaluate the security bound of full round AES against biclique attack. We try to solve some of the inherent limitations of biclique attacks such as high data complexity. Through a computer-assisted search we try to find optimal attacks that produce the lowest computational and data complexities for biclique key recovery attack on all 3 variants of AES.

- In Chapter 4, we deal with AES-based hash functions and investigate their resistance against biclique attacks. First, we give algorithms to search best biclique trails that do not modify the IV input to the hash function. Then, using these biclique trails we launch second preimage attacks on all 12 PGV modes.

- In Chapter 5, we analyze Type-2 Generalized Feistel Networks (GFNs) in known key scenario and derive actual collisions for hash functions constructed from 4-branch, type-2 GFNs. We further demonstrate the best 8-round collision attack on this construction when the round function F is instantiated with double substitution-permutation (double SP) layers.

- In Chapter 6, we present multiset attack, a variant of meet-in-the-middle attack and give key recovery attacks on reduced round ARIA-192 and ARIA-256. We

present new 4-round distinguishers and use them to launch attacks on 7 and 8-round ARIA-192 and ARIA-256 with improved attack complexities. Further, in our attacks, we are able to recover the actual secret key unlike the previous cryptanalytic attacks existing on them.

- In Chapter 7, we investigate the security of recently standardized Ukrainian encryption algorithm: Kalyna-128/256 and Kalyna-256/512. We present new 6-round distinguishers and use them to launch multiset based key recovery attacks on 9-round Kalyna-128/256 and Kalyna-256/512.

- Finally in Chapter 8, we conclude the thesis by discussing the results and giving some future directions of research.

In terms of either the attack complexity or the number of rounds broken, the attacks presented in this thesis are better than any previously published cryptanalytic results for the block ciphers concerned.

1.3 Contributions

Among the list of publications presented on page 4, all the results reported in [8, 10, 11, 37, 49] form the basis of this thesis. In the joint works, the author of the thesis has played a leading role in obtaining the results reported in this thesis. The main cryptanalytic results are as follows:

- Chapter 3 re-evaluates the security bound of full round AES against biclique based key recovery attacks. The results include:

    1. All the biclique attacks before this work were limited to balanced bicliques only, i.e., complete bipartite graphs in which the two set of vertices have exactly the same cardinality. In this work, we propose maximally unbalanced bicliques called *stars* where one set of vertices only contains a single element.
    2. We search for attacks with the minimal data complexity in accordance with the unicity distance. We find attacks which just require 2 (for AES-128 and AES-192) or 3 (for AES-256) known plaintexts with success probability 1. We utilize star based bicliques to achieve these results.
    3. We search for attacks with data complexity strictly less than the full codebook. Among this class, we find attacks that have lower data complexity for AES-128 and AES-192 (as compared to the original attacks in [39]) and lower computational complexity for AES-256. This shows that the attacks reported in [39] were not optimal.

4. We search for the fastest biclique key recovery attacks in the entire class when there is no restriction on the amount of data required. These attacks provide an important insight into the limits of the independent-biclique approach developed so far.
5. An interesting outcome of the above class of constructions is that they utilize the longest biclique covered in the full AES attack so far. For AES-128, the longest biclique has length of 3 rounds whereas for AES-192 and AES-256 the longest bicliques cover 5 rounds each.

- Chapter 4 investigates the security of AES based hash functions against biclique attacks in known-key scenario, where the IV is fixed and cannot be changed by the attacker. Though our results do not significantly decrease the attack complexity factor as compared to brute force but, they highlight the actual security margin provided by these constructions against second preimage attack. Our results include:

  1. We re-evaluate the offered security of full 10 rounds AES-128 based hash functions against second preimage attack. The previous best result reported in [149] could only work on 7 rounds.
  2. Our analysis works on all 12 PGV modes of the hash function constructions.
  3. We propose new biclique trails to achieve the above results.
  4. All the trails have been obtained by implementing C programs with suitable restrictions imposed on the search space to yield the best attack complexities.

- Chapter 5 investigates GFN based hash functions in known-key scenario against sliced biclique cryptanalysis. Our results include:

  1. We apply sliced biclique technique to construct an 8-round distinguisher on 4-branch, Type-2 Generalized Feistel Network.
  2. We use the distinguisher so constructed to demonstrate an 8-round collision attack on 4-branch, Type-2 GFN based compression functions under known key settings. The attack can be directly translated to collision attacks on Matyas-Meyer-Oseas (MMO) and Miyaguchi-Preneel (MP) mode based hash functions and pseudo-collision attacks on Davies-Meyer (DM) mode based hash functions.
  3. When the round function F is instantiated with double SP layer, we demonstrate the first 8-round collision attack on 4-branch, Type-2 GFN with double SP layer. This improves upon the previous best 6-round attack reported in [148].
  4. We also further show that presence of multiple SP layers in the round function F does not always provide better resistance against some attacks.

5. We investigate CLEFIA which is a real world-implementation of 4-branch, Type-2 GFN and demonstrate an 8-round collision attack on it.

- Chapter 6 investigates the effectiveness of mutiset attacks on the block cipher ARIA. The results include:

  1. We show the best 7-round key recovery attacks on ARIA-192/256 and 8-round attack on ARIA-256.
  2. We construct a new 4-round distinguisher for ARIA-192 and ARIA-256.
  3. Our attack complexities on ARIA-192 have better time, data and memory complexities than the previous best reported in [168].
  4. Our attack complexities on ARIA-256 have better time and memory complexities than the previous best reported in [168].
  5. We present the first actual recovery of the master key on our attacks on ARIA-192 and ARIA-256.

- Chapter 7 investigates the effectiveness of mutiset attacks on the block cipher Kalyna. The results include:

  1. We present the first 9-round key recovery attack on Kalyna-128/256 and Kalyna-256/512. This improves upon the previous best 7-round attacks on the same.
  2. We construct new 6-round distinguishers on each of the above mentioned Kalyna variants.
  3. Our 9-round attack on Kalyna-256/512 has better time and data complexity than the previous best reported in [13].
  4. Our attacks show that Kalyna variants which have equal block and key sizes appear to be more secure and robust as compared to Kalyna variants where the key size is double the block size.

Chapter 2

Symmetric cryptosystems

A cryptosystem [97,133,164] is a general term referring to a set of cryptographic primitives used to provide information security services. Symmetric cryptology studies the design and analysis of symmetric cryptosystems which include block ciphers, stream ciphers, message authentication codes (MACs) and cryptographic hash functions. While block ciphers and stream ciphers are encryption primitives, MACs and hash functions provide data and data origin authentication. This thesis primarily focuses on block ciphers and block cipher based hash functions. In this chapter, we provide a brief overview of the necessary background required to understand the results included in this thesis.

2.1 What is a block cipher ?

A block cipher is a transformation E: $\{0,1\}^k \times \{0,1\}^n \rightarrow \{0,1\}^n$. The first input to E is a k-bit secret key and second input is the n-bit plaintext while the output is the n-bit ciphertext. [1] The key size k and the block size n are the parameters associated with a block cipher. A block cipher is a bijection for a fixed key, i.e., $E(K,\cdot)$ is a bijective function on $(0,1)^n$. For a fixed key $K \in \{0,1\}^k$, we sometimes denote $E(K,\cdot)$ by $E_K(\cdot)$ and its inverse permutation by $E_K^{-1}(\cdot)$. Currently, the widely used block lengths are 64 and 128-bits and the key lengths are typically 128 or 256-bits.

The need for formalizing security notions in communication systems was recognized quite early. Shannon, in his seminal work [157] first defined the concept of *perfect secrecy* for encryption ciphers. A cipher is called perfectly secure if the ciphertext does not give the adversary any additional information about the plaintext. Shannon showed that the ciphers must have a key space that is atleast as large as the message space for them to achieve perfect secrecy. This makes ciphers with perfect secrecy impractical in most settings where large amounts of data need to be encrypted. Block ciphers

[1] Often the term ciphertext is slightly abused to mean the output of encrypting a plaintext block using a reduced version of the block cipher concerned.

present more efficient ways to construct encryption algorithms under a weaker notion of security termed as *computational security* [97].

A standard assumption associated with block ciphers is that they are *pseudorandom permutations*. This means that an n-bit block cipher under a randomly-chosen secret key K is computationally indistinguishable from a randomly chosen n-bit permutation. By definition, in the ideal-cipher model [34, 54, 63, 67, 91], a block cipher E with k-bit key and n-bit block size is uniformly chosen from the set of all possible block ciphers of this form. For each key, there are $2^n!$ permutations, and since any permutation may be assigned to a given key, there are $(2^n!)^{2^k}$ possible block ciphers, e.g., AES with a 128-bit key is one choice from nearly $2^{2^{263}}$ total block ciphers possible. However, an ideal block cipher for a large block size is not practical from an implementation and performance point of view. This is because, for most practically relevant block ciphers, the amount of storage required to store all the possible permutations for a given key is way beyond the computational resources conveniently available [33]. In practice, an approximation of ideal block cipher system for large n is constructed, where the block cipher is built out of components that are easily realizable [88]. To build a block cipher, a set of permutation generators is taken and parameterized by the secret key [35] where the key space is much smaller than the message space. This reduces the possible block cipher choices significantly and the implementations are far from being random. In such cases, the security of a block cipher depends on its block size and the secret key size. The length of these parameters is so chosen that they are large enough to maintain pseudorandomness in presence of modern adversary. In terms of computational security, a block cipher E is called secure if no *attacks* on E exist that have *time complexities* [2] lower than brute force.

2.2 Anatomy of a block cipher

In practice, most block ciphers are *iterated block ciphers*. They are constructed by repeating a simple function called the *round function* multiple times. Parameters include the number of rounds r, the block bitsize n and the bitsize k of the input secret key from which r subkeys K_i (called the round keys) are derived through a key schedule algorithm. The basic idea is to make a strong encryption function out of a weaker round function (that is easy to implement) by repeatedly using it. For an r-round block cipher with i^{th} round function F_i and i^{th} subkey K_i, an iterated block cipher $E_K(\cdot)$ is defined as:

$$E_K(\cdot) = F_i(K_i, \cdot) \circ F_{i-1}(K_{i-1}, \cdot) \circ \ldots \circ F_2(K_2, \cdot) \circ F_1(K_1, \cdot)$$

where, $\circ$ denotes superposition of permutations. For every subkey, the round function must be invertible; if not, decryption is impossible. The number of rounds in an

[2] Attack and attack complexities are formally discussed in Section 2.4

iterated cipher depends on the desired security level and the consequent trade-off with performance. In most cases, an increased number of rounds will improve the security offered by a block cipher, but for some ciphers the number of rounds required to achieve adequate security will be too large for the cipher to be practical or desirable. Often in an iterated block cipher the first and/or the last round are not identical with the other ones.

A *key-alternating* block cipher [56, 57] is an iterative block cipher with special-type of round function F_i :

$$F_i(K_i, x) = F_i'(K_i \oplus x)$$

where, $F_i'(\cdot)$ is a round function not dependent on the round key. The round functions are interleaved with simple xoring of round keys to the current state. Advanced Encryption Standard (AES) is a key-alternating block cipher.

Round Function F. It is necessary to build the round function F as *nonlinear*. This eliminates the possibility of representing F as a system of linear equations with plaintext, ciphertext and key bits acting as variables and solving them with standard algebraic methods to recover the secret key. It is generally desired that the round function F should provide good diffusion and confusion properties. Good diffusion means spreading the influence of each input bit to preferably all output bits in a random way, whereas good confusion means making the relationship between ciphertext and subkey bits as complex as possible such that it is difficult to deduce the secret key [163]. If the round function achieves good diffusion and confusion properties, after sufficiently many rounds, each instantiation of the block cipher is expected to behave like a random permutation and resist various cryptanalytic attacks.

2.3 Construction of iterated block ciphers

Iterated block ciphers can be built in many different ways. The two most common approaches are Feistel Networks (FNs) and Substitution Permutation Networks (SPNs).

Feistel Network. Feistel network, more commonly known as Feistel cipher has been named after *Horst Feistel*, one of the IBM researchers who designed LUCIFER [162]– the precursor to Data Encryption Standard (DES) [62]. In a Feistel cipher, the plaintext is split into two equal halves. The round function is applied to one half and the output of the round function is bitwise xor'ed with the other half; finally, the two halves are swapped and they become the two halves of the next rounds. This process is iterated until the final ciphertext is produced. Typically, in a Feistel cipher, number of rounds $r \geq 3$ and is often even. Decryption is achieved using the same r-round process but with subkeys used in reverse order. As a result, the round function F need not be a

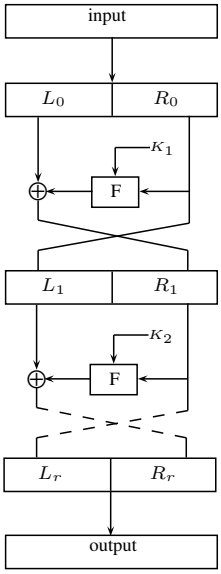
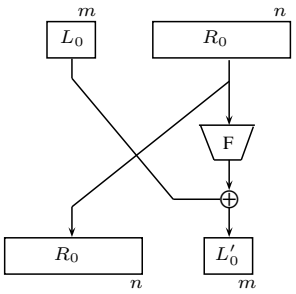


Figure 2.1: An example of Feistel construction. The input is split into two halves: $L_0 \,||\, R_0$ and iterated r times to produce the final output.

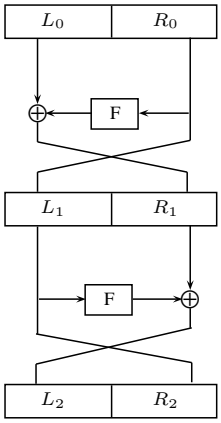
bijection for Feistel constructions. An illustration of Feistel cipher is given in Fig. 2.1. Concrete examples of Feistel cipher include - TEA [175], XTEA [135], ISO standard Camellia [16], GSM standard KASUMI [3], NSA released SIMON [22] etc.

Generalized Feistel Network (GFN). Beside the classical Feistel networks as discussed above, several other generalizations of Feistel networks exists. All of them are encompassed under the broader category of *Generalized Feistel Networks* (GFNs). In *Unbalanced Feistel networks* [155], unlike the classical ones, each state is divided into two unequal parts, e.g., Skipjack [2] whereas in *alternating Feistel networks*, the right and left parts are alternately used as input to the round function F, e.g., LION [14]. In *type-1*, *type-2*, and *type-3* Feistel networks [180] that are variants of Feistel networks with more than two branches, the input message is partitioned into k sub-blocks with $k > 2$, e.g., RC6 [145], CLEFIA [160], TWINE [167], MARS [45] etc. Fig. 2.2, illustrates the various GFNs. For a detailed security analysis of generalized feistel networks, one can refer to [42, 85]. In [166], Suzaki et. al. specifically investigated the diffusion properties of k-branch, Type-2 GFN, where $k > 2$ and showed that the classical k-block left or right cyclic shift had poor diffusion abilities. They then suggested improved k-block shuffling that led to better resistance against various cryptanalytic attacks. Block cipher TWINE [167], which is a 16-branch, Type-2 GFN adopts one of these improved k-block shuffling in its design.

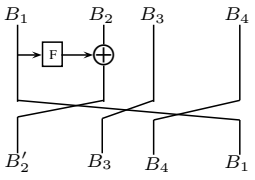
Substitution Permutation Network (SPN). Another approach to construct a block cipher consists of building a round function by combining layers of simple in-



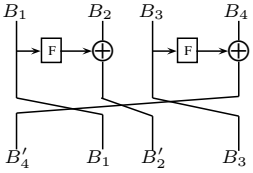
(a) Unbalanced Feistel Network. Here, $|L_0| < |R_0|$.



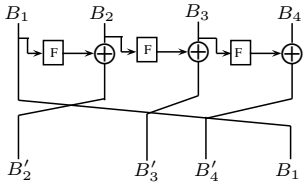
(b) Alternating Feistel Network.



(c) 4-branch, Type-1 GFN



(d) 4-branch, Type-2 GFN



(e) 4-branch, Type-3 GFN

Figure 2.2: Illustration of Generalized Feistel Networks

vertible functions: substitutions and permutations. In the first layer, the subkey K_i is xor'ed with the intermediate state which provides key dependency. In the second layer, i.e., the substitution layer, nonlinear functions (S-boxes) acting on parts of the state are applied in parallel. If each S-box operates on b out of n bits, then there are $m = \frac{n}{b}$ b-bit S-boxes working in parallel. In the third layer, i.e., the permutation layer, these m parts are diffused using a linear mapping. The substitution layers acting on small units of data (rarely more than eight consecutive bits) introduce local confusion into the cipher. The permutation layers, on the other hand, operate on the complete block and thus diffuse the effect of the substitutions. In practice, S-boxes are often implemented as look-up tables as table implementation gives better performance in software. Linear permutations range from simple transpositions such as bit-wise permutations to complex mathematical functions, such as MDS matrix [144, 170]. While bit level permutations are easy to achieve in hardware with zero cost (simple realigning of wires), they have serious performance implications at software level. This is because, in software, manipulating individual bits is not natural and this slows down the performance of the cipher. Hence, permutation functions such as the MDS matrix

which works at byte level or 32-bit word level provide more flexibility in software. In SP networks, the round function has to be necessarily bijective for decryption to be possible. Prominent SPN block ciphers include: AES [57], PRESENT [40], ARIA [114] etc. Often the F-functions of many Feistel ciphers also consist of a small SP network. An example of SP network is given in Fig. 2.3.

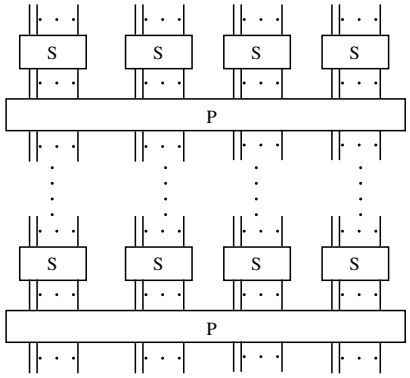


Figure 2.3: Substitution-Permutation (SP) Network

Besides these popular design approaches, there exists other block ciphers with other round structures as well, e.g., block cipher IDEA [116] that follows Lai-Massey scheme [171], NSA standardized SPECK [22] that follows ARX (addition, rotation, XOR)-based structure [172] and KATAN/KTANTAN family of lightweight block ciphers which base their round function on LFSRs [47].

Key Schedule Algorithm (KSA). KSA plays an important role in iterated block cipher design construction. Reusing the user supplied key bits as much as possible while generating the round keys is usually considered a good design principle. Two main approaches of designing a key schedule algorithm are - 1) *Affine key schedule*, where, the subkeys are derived as an affine transformation of the user supplied key, e.g., DES and 2) *Non linear key schedule*, where, the subkeys are generated as non-linear transformations of the user supplied key, e.g., AES, PRESENT etc. It is usually desired that a key schedule algorithm should avoid weak keys, equivalent keys, related keys etc. and satisfy Strict Avalanche Criterion and Bit Independence Criterion [7].

In resource constrained environments, where efficiency and security are competing goals, designers often opt for a simple key schedule to ensure compact implementation. For example, key schedules of block ciphers such as PICCOLO [158] and TEA [175] do simple permutation or linear operation on the master key whereas some block ciphers such as LED [81] and PRINT [108] have no key schedule at all and just use the master key directly in each round. Security in these block ciphers is ensured by utilizing specialized design tricks during the state update such as - a serially computable MDS

matrix used in block cipher LED, double permutation layer (bit-wise shuffling followed by a 3-bit keyed permutation) in block cipher PRINT etc. which provide good diffusion but require small gate count in hardware implementations. Most of these block ciphers are well analyzed against the classical attacks and resistant against KSA targeting attacks such as related key and slide attacks. However, in some cases, the simplicity of these key schedule algorithms (i.e., slow diffusion between the subkeys) has been exploited to launch attacks on reduced rounds, e.g., MITM attacks [90, 156], invariant subspace attack [120] etc.

Due to a lack of systematic guidelines, no formal principles on designing a key schedule algorithm have yet been promulgated and this is a pressing research issue.

2.4 Block Cipher Cryptanalysis

Block cipher cryptanalysis is multifaceted. An attack may show a weakness that the designer overlooked, disprove an assumed security level or show an assumed property to be untrue. Broadly speaking, the attack goals can be classified into the following categories [106]:

- *Total Break* - Here, an adversary recovers the full secret key.
- *Global Deduction* - The adversary finds an algorithm that is functionally equivalent to either encryption or decryption algorithm.
- *Local Deduction* - The adversary can obtain the plaintext (or ciphertext) corresponding to a ciphertext (or plaintext) that has not been previously queried.
- *Distinguishing Algorithm* - The adversary can effectively differentiate between two black boxes; one containing the target block cipher with a randomly chosen secret key and the other containing a randomly chosen permutation.

It is generally agreed that any cryptosystem should meet Kerckhoffs' principle [95]. According to Kerckhoff's principle, a cryptosystem should be secure even if an attacker knows everything about the cryptographic algorithm except the user-supplied secret key. Following the Kerckhoff's principle, there are four widely discussed attack scenarios that an attacker may launch. In the order of severity these are:

1. *Ciphertext-only Attack.* This is the most basic type of attack. Here, the adversary just observes one or more ciphertexts and attempts to determine the corresponding plaintexts that were encrypted. She is assumed to have some information about the plaintexts, e.g., the particular natural language in which the plaintexts are written.

2. *Known Plaintext Attack.* Here, the adversary learns one or more pairs of plaintexts-ciphertexts encrypted under the same key. Her aim is then to determine the plaintext for some other ciphertext for which she does not know the corresponding plaintext.

3. *Chosen Plaintext (or Ciphertext) Attack.* In this attack, the adversary has the ability to obtain the encryption (or decryption) of any plaintext(s)(or ciphertexts) of her choice. She then attempts to determine the plaintext (or ciphertext) for some other ciphertext (or plaintext) that has not been previously queried.

4. *Adaptively Chosen Plaintext (or Ciphertext) Attack.* It is the same attack scenario as above except that in this case an adversary can adaptively ask for encryptions and/or decryptions of her choice based on the knowledge obtained from the previous queries.

The strength of an attack is usually evaluated using the following metrics:

- *Time Complexity.* This measures the amount of computations required for execution of an attack. It is usually measured in terms of how many encryption/decryption calls to the target block cipher are made.

- *Data Complexity.* This calculates the number of plaintexts/ciphertexts required to execute the attack. In an exhaustive search attack, *unicity distance* is a parameter of a block cipher that is generally used to determine the minimum number of ciphertexts required such that only one correct value of the secret key is obtained (other spurious keys are eliminated) [142]. E.g., for a 128-bit block cipher with a 256-bit secret key, unicity distance is 3.

- *Memory Complexity.* This calculates the amount of memory storage required to carry out an attack. Often, it is measured in units of n-bit blocks (where, n denotes block size) of data required to be stored.

2.4.1 Fundamental Generic Cryptanalysis Techniques

There are three fundamental cryptanalytic techniques that can be applied to any block cipher. These attacks do not depend on the internal structure of the cipher and can only be avoided or rendered impractical by choosing appropriate external parameters. We assume an n-bit block cipher with a k-bit secret key.

- *Dictionary Attack.* Here, the attacker builds a table containing all 2^k possible ciphertexts corresponding to a particular plaintext with various key choices acting as the indices. When a ciphertext corresponding to that particular plaintext is intercepted, she can simply look up the precomputed table and deduce the key with high probability (given, $k \leq n$, otherwise there are 2^{k-n} expected keys). This attack has a data complexity of 2^k ciphertexts, a 2^k n-bit memory complexity and negligible time complexity.

- *Codebook Attack.* This attack is related to the block length of the cipher. If the attacker is able to capture the ciphertexts of all possible 2^n plaintexts, she can construct a table (sorted by the plaintext) and use it to decrypt any future message encrypted with the same secret key. Such an attack has a data complexity of 2^n plaintext/ciphertext pairs, a 2^n n-bit memory complexity and a negligible time complexity. Generally, due to birthday paradox, if more than $2^{n/2}$ randomly generated n-bit ciphertexts are captured, then an attacker can expect a repetition to occur with high probability. This may in turn leak information about the plaintexts. This is a potentially relevant setting as in many cryptographic constructions, a source of randomization is fed into the input of the block cipher, e.g., CBC mode with random IV. Hence, it is advisable not to encrypt more than $2^{n/2}$ plaintexts under the same key. In case n is relatively small, codebook attack becomes practically feasible.

- *Exhaustive Search.* This attack depends upon the key length of the secret key used. In an exhaustive key search (also known as brute force attack), given a valid plaintext-ciphertext pair, an attacker tries all possible 2^k key choices to deduce the correct key. It is assumed that only the correct key will yield the correct correspondence. If more than one candidate key is produced, then the wrong key choices can be eliminated by using another valid plaintext-ciphertext pair. Such an attack has a 2^k time complexity and negligible data and memory complexities. An attack is generally considered effective if its time complexity is faster than exhaustive key search.

While exhaustive search requires essentially no memory and 2^k work, dictionary attack requires 2^k memory and negligible time. Hence, these attacks represent the two extremes for an adversary and bring into picture the concept of *time-memory tradeoff*. The classic time-memory tradeoff was given by Martin E. Hellman and for further details, one can refer [83].

2.4.2 Shortcut Attacks

Shortcut Attacks exploit specific internal structure properties of a block cipher to recover the secret key with a complexity better than brute-force. Most of the attacks that are discussed in the subsequent subsections follow the following 2-step strategy:

1. *Distinguisher Construction.* In this phase, given a set of plaintext-ciphertext pairs, the aim of the attacker is to construct a distinguisher D that is able to differentiate between a random permutation and the target block cipher (that looks like a random permutation to the attacker) in polynomial time [3] and with a non-marginal success probability. For further details on various statistical distinguishers, one can refer [94].

[3] polynomial in the block size n

2. *Round Key Recovery.* Let us suppose that an attacker is able to successfully construct a distinguisher D on a $(r-1)$ reduced round block cipher (out of full r-rounds). Given plaintext-ciphertext pairs for full cipher, the attacker guesses (parts of) the last round key, (partly) decrypts the last round, and uses her distinguisher on the first $(r-1)$ rounds to check whether the guess could have been correct. Once, she has recovered the (parts of) last round subkey, she can: a) Deduce some parts of the secret key K if the key schedule is invertible or, b) Peel off the last round and continue searching in a similar manner or, c) Conduct an exhaustive search for the rest of the key bits to recover the secret key K. Sometimes, depending on the block cipher structure, an adversary may cover more than one round in the key recovery phase, both from the top as well as bottom.

In the subsequent subsections, we briefly review a range of shortcut attacks that are relevant for this thesis.

2.4.3 Differential Cryptanalysis

Differential cryptanalysis was first presented by Biham and Shamir at CRYPTO'90 to attack DES [28] and eventually the details of the attack were published as a book [29]. The wide applicability of this technique along with linear cryptanalysis [131] to numerous block ciphers have solidified the position of these techniques as the threshold evaluation metric for the security analysis of any block cipher. One of the major aims of all block cipher designers is to develop constructions that can thwart differential and linear cryptanalysis.

Differential Cryptanalysis takes advantage of how a fixed plaintext difference propagates through the rest of the cipher in a non-random way to recover the secret key. The notion of difference can be defined in several ways but the most widely discussed is with respect to the XOR operation. For any value X during the encryption of P, and the corresponding value X' during the encryption of P' (where, size of P, P', X, X' is n-bits), let us denote the difference by $\Delta X = X \oplus X'$. The differences are linear in linear operations, and it is easy to predict the output difference of linear operations given the input difference. However, in case of non-linear operations, say S-box (S), this is not true. Specifically, $S(X) \oplus S(X') \neq S(X \oplus X')$. Let, $S(X) \oplus S(X') = \Delta Y$. Given, a $(\Delta X, \Delta Y)$ pair, in an ideal randomizing cipher, the probability that a particular output difference ΔY occurs for a given ΔX is $1/2^n$. However, since block ciphers in practice are not truly random, differential cryptanalysis seeks to exploit cases where the probability of $\Delta X \rightarrow \Delta Y$ is much greater than $1/2^n$. The pair $(\Delta X, \Delta Y)$ is called a *differential characteristic* across the operation $S(\cdot)$.

Given a $m \times n$ S-box (S), the probability of the differential characteristic over S $(\Delta X, \Delta Y)$ is defined as:

$$Pr(\Delta X \rightarrow \Delta Y) = \frac{\mid \{x \in \{0,1\}^m : S(x) \oplus S(x \oplus \Delta X) = \Delta Y\} \mid}{2^m}$$

The aim of the attacker is to search for differential characteristics which have good probabilities. To achieve so, generally a *Difference Distribution Table* (DDT) is built by her. The *Difference Distribution Table* for an $(m \times n)$ S-box (S) is a table storing all possible pairs of input and output differences (ΔX, ΔY) and the numbers of m-bit blocks x ($\in \{0,1\}^m$) such that $S(x) \oplus S(x \oplus \Delta X) = \Delta Y$.

In a differential attack, the attacker first starts by searching a differential characteristic such that only a small part of the nonlinear components are *activated* (have non zero difference) and at the same time maximum rounds of the target cipher are covered. To estimate the probability of a characteristic, it is assumed that its evolution over successive rounds is independent. Thus, the cumulative probability of a $(r-1)$-round differential characteristic (over $(r-1)$-rounds of the cipher) can be computed as the product of probabilities of one-round characteristics. Once the attacker finds a good differential characteristic that covers the desired rounds of the cipher, she can then use it to distinguish the block cipher from a random permutation and carry out a key recovery attack as described at the start of the Section 2.4.2. It can be seen that in general, in differential attacks, the attacker is usually concerned with the difference in the partially encrypted inputs after $(r-1)$ rounds only rather than input differences at each intermediate round. There may be various sequences of intermediate differences that give rise to the same output difference. Thus, instead of working with a particular differential characteristic, differential attacks often employ *differentials*. A *differential* over $(r-1)$ rounds is a set of all differential characteristics with the same input and output difference.

Various enhancements and variants of differential attack have been introduced and studied extensively in literature. Few among them are higher order differential cryptanalysis [105, 181], truncated differential cryptanalysis [107], impossible differential cryptanalysis [23, 126, 140], differential-linear cryptanalysis [119], boomerang attacks [173] etc. In the next subsections, we discuss two of these attacks - truncated differential cryptanalysis and boomerang attacks.

2.4.4 Truncated Differential Cryptanalysis

Truncated Differential Cryptanalysis is a generalized version of differential cryptanalysis technique. This technique was developed by Lars Knudsen in 1994 [107]. It works exactly the same way as differential cryptanalysis except the fact that in this technique, instead of analyzing the full difference between the two texts, only partial differences are determined after each round. For example, consider a particular 4-bit S-box, say S. Let

$$0010 \xrightarrow{S} 0010 \qquad \text{have a probability of } 2/16$$
$$0010 \xrightarrow{S} 1000 \qquad \text{have a probability of } 6/16$$
$$0010 \xrightarrow{S} 0110 \qquad \text{have a probability of } 4/16$$
$$0010 \xrightarrow{S} 0100 \qquad \text{have a probability of } 4/16$$

In case of truncated differential,

$$0010 \xrightarrow{S} *** 0 \qquad \text{has a probability 1}$$

At byte level, truncated differentials usually work with a set of differences. A byte is usually termed as *active* or *non-active* just indicating whether a non-zero difference exists in the byte or not, without focusing on the actual value of that difference. S-boxes do not mix active and non-active bytes. Thus, it is easier to study and exploit the properties of the diffusion layer in case of truncated differentials. However, truncated differentials quickly spread to the whole internal state due to diffusion and this happens faster than for fixed differentials. This makes the probabilistic trail short and the attack less powerful. If the diffusion at byte/word is relatively slow, truncated differentials may cover larger number of rounds. E.g., in case of block cipher Crypton, due to weaker diffusion, as many as 8 rounds were attacked with truncated differentials [64].

2.4.5 Boomerang Attack

Boomerang attack was proposed by David Wagner at FSE'99 [173]. It differs from the classical differential attack approach in that it allows differentials covering only a part of the cipher to be used in the attack. The block cipher is treated as a cascade of two sub-ciphers, each having a high probability short differential of its own. These differentials are then combined in a chosen plaintext and ciphertext attack setting to first construct a boomerang distinguisher and then use the distinguisher to recover the secret key.

Let us suppose, the block cipher $E_K(\cdot)$ is split into two halves - $E_K^1 \circ E_K^0$, where E_K^0 covers the first s rounds of encryption while E_K^1 covers the rest $(r-s)$ rounds of encryption. Let us further suppose, that there exists a differential $\alpha \rightarrow \beta$ through E_K^0 with a high probability p. Similarly, there exists a differential $\gamma \rightarrow \phi$ through $(E_K^1)^{-1}$ which has a high probability q. The boomerang attack (as shown in Fig. 2.4) then proceeds as follows:

1. Consider two plaintexts P, P' such that $P' = P \oplus \alpha$. Obtain their corresponding ciphertexts C, C' respectively.

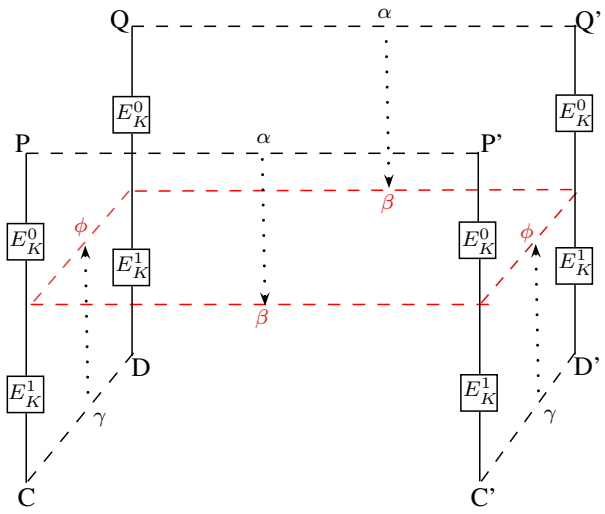


Figure 2.4: The Boomerang Distinguisher

2. The probability that $E_K^0(P) \oplus E_K^0(P') = \beta$ is p.

3. Obtain, $D = C \oplus \gamma$ and $D' = C' \oplus \gamma$. If we apply $(E_K^1)^{-1}$ to each of the pairs (C, D) and (C', D'), then with probability q^2, $(E_K^1)^{-1}(C) \oplus (E_K^1)^{-1}(D) = \phi$ and $(E_K^1)^{-1}(C') \oplus (E_K^1)^{-1}(D') = \phi$.

4. Then, the following statement holds true: With probability pq^2, $(E_K^1)^{-1}(D) \oplus (E_K^1)^{-1}(D') = \beta$. This is because,

$$\begin{aligned}(E_K^1)^{-1}(D) \oplus (E_K^1)^{-1}(D') &= (E_K^1)^{-1}(C) \oplus (E_K^1)^{-1}(D) \oplus (E_K^1)^{-1}(C) \oplus (E_K^1)^{-1}(C') \\ &\quad \oplus (E_K^1)^{-1}(C') \oplus (E_K^1)^{-1}(D') \\ &= \phi \oplus \beta \oplus \phi \\ &= \beta\end{aligned}$$

5. Thus, with probability p^2q^2, $E_K^{-1}(D) \oplus E_K^{-1}(D') = Q \oplus Q' = \alpha$.

6. Now if, $(pq) > 2^{-n/2}$, then a valid distinguisher is constructed. This is because, for a random permutation, the expected probability that $Q \oplus Q' = \alpha$ is 2^{-n}.

Therefore, if p^2q^2 is sufficiently large, then the boomerang distinguisher can effectively distinguish between $E_K(\cdot)$ and a randomly chosen permutation, given a sufficient number of adaptively chosen plaintexts and ciphertexts. Once the distinguisher is built, the attacker can use it to carry out a key recovery attack as described at the start of the Section 2.4.2. The plaintext tuple (P, P', Q, Q') is termed as a *quartet* and satisfies the following property:

$$P \oplus P' \oplus Q \oplus Q' = 0.$$

Following the advent of boomerang attack, many other extensions of this attack were proposed. E.g., *amplified boomerang attack* [98] which is a full chosen plaintext attack variant, *rectangle attack* [24] which allows any value of β and ϕ to occur as long as $\beta \neq \phi$ and *impossible boomerang attack* [125] which defines a right quartet to be one that satisfies $\beta \oplus \beta' \oplus \phi \oplus \phi' \neq 0$ in the middle.

2.4.6 Meet-in-the-Middle Attack

Meet-in-the-Middle Attack (MITM) technique, first proposed by Diffie and Hellman in [66] is a divide-and-conquer approach which splits an invertible transformation into two parts and finds parameters that are involved in the computation of only one. These parameters are then calculated independently and checked for a match in the middle to obtain the right combination. The advantage of this technique is that it reduces the time complexity significantly as compared to brute force search over these parameters. For example, let us have a look at MITM attack on Double DES [66]. For Double DES, the following equation holds true:

$$DES_{K_1}(DES_{K_2}(P)) = C \tag{2.1}$$

where, $P =$ plaintext, $C =$ ciphertext, $(K_1, K_2) =$ keys used for encryption/decryption and $\mid K_1 \mid = \mid K_2 \mid = 56$ bits. From Eq. (2.1), it can be seen that :

$$DES_{K_2}(P) = DES^{-1}_{K_1}(C) \tag{2.2}$$

i.e., given a plaintext-ciphertext pair, an attacker can compute K_1 and K_2 independently and look for a match in the intermediate ciphertext. Finding the right combination (K_1, K_2) will take the attacker $2^{K_1} + 2^{K_2}$ time instead of $2^{K_1} \times 2^{K_2}$ time (needed by brute force search). Hence, instead of providing 112-bit security, MITM attack showed that Double DES only provides 57-bit security.

The same methodology can be exploited at round level as well. The aim is to find an internal state inside the cipher such that rounds involved in the forward computation from the plaintext and in the backward direction from the ciphertext do not depend on particular key bits. These particular key bits are called *neutral key bits*. In general, let an n-bit block cipher $E_K(\cdot)$ with k-bit key K is divided into two functions F_1 and F_2. Let us further suppose that K is grouped into three sets K_1, K_2 which are used only in F_1, F_2 independently (neutral key bits) and K_3 which denotes the other key bits of K. Then for each value of K_1, we compute $F_1(P)$ in the forward direction and for each value of K_2, we compute $F_2^{-1}(C)$ in the backward direction. If the guessed key is the correct one, the equation $F_1(P) = F_2^{-1}(C)$ holds true. After MITM stage, 2^{k-n} $(= \frac{2^{|K_1|+|K_2|}}{2^n} \times 2^{|K_3|})$ keys will survive. The attacker then exhaustively searches a

correct key from the surviving key candidates by using additional plaintext-ciphertext pairs. The required complexity C_{comp} is estimated as:

$$C_{comp} = 2^{|K_3|}(2^{|K_1|} + 2^{|K_2|}) + (2^{k-n} + 2^{k-2n} + \ldots) \tag{2.3}$$

The number of required plaintext/ciphertext pairs is $\lceil k/n \rceil$.

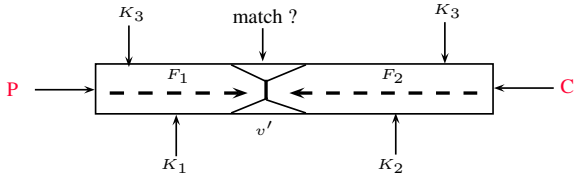


Figure 2.5: MITM Attack with partial matching. Here, v' is the b-bit partial intermediate matching state.

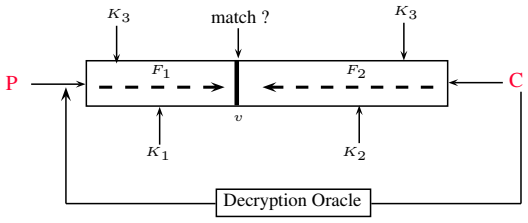


Figure 2.6: Splice-and-Cut Framework. Here, v is the n-bit intermediate matching state.

Sometimes, instead of full n-bit state matching, attacker performs matching on some part of the intermediate state only. Such an attack is termed as MITM with *partial matching* [17]. For a match on b-bit ($b < n$) partial intermediate state (as shown in Fig. 2.5), the required complexity C_{comp} is now estimated as:

$$C_{comp} = 2^{|K3|}(2^{|K1|} + 2^{|K2|}) + (2^{k-b} + 2^{k-2b} + \ldots) \tag{2.4}$$

with the data complexity of the attack increasing to $\lceil k/b \rceil$. Another property which an attacker sometimes exploits in MITM attacks is the *Splice-and-cut Technique* [17]. This technique considers the first and last step of a block cipher as consecutive steps. This is achieved by joining the two steps through an encryption/decryption oracle query (as shown in Fig. 2.6). Both these properties provide an attacker additional flexibility to start at any step she wishes and cover maximum rounds possible to recover the secret key.

Meet-in-the-middle attacks on block ciphers received less attention compared to differential and linear cryptanalysis. The limited use of these attacks can be attributed to the fact that these attacks require large parts of the cipher to be independent of certain key bits whereas, the design paradigm of block ciphers requires usage of all the key bits in the first few rounds only. As such, finding an internal state which depends on only few key bits in either direction is difficult. As a result, the number of rounds broken with this technique is rather small. MITM attacks could break significant rounds only in some block ciphers, e.g., full cipher KTANTAN [41] and maximum steps in XTEA, LED and Piccolo [90].

2.4.7 Square Attack

Square Attack was first proposed by Daemen et al. in [55] as a dedicated attack on block cipher SQUARE, a forerunner of AES. It was shown to be applicable to AES as well. This attack consists of choosing a special set of plaintexts and studying its propagation through the block cipher. The attack on AES [110] is illustrated as follows:

Consider a set of 2^8 plaintexts in which the first byte takes all possible 256 values and the remaining bytes take any constant value that remains same throughout the set. We call such a set of plaintexts as λ-set. The byte which takes all possible 256 values is termed as the *active byte*. Rest of the bytes are termed as *passive bytes*. The transformation of this λ-set after one round of AES encryption is as shown in Fig. 2.7.

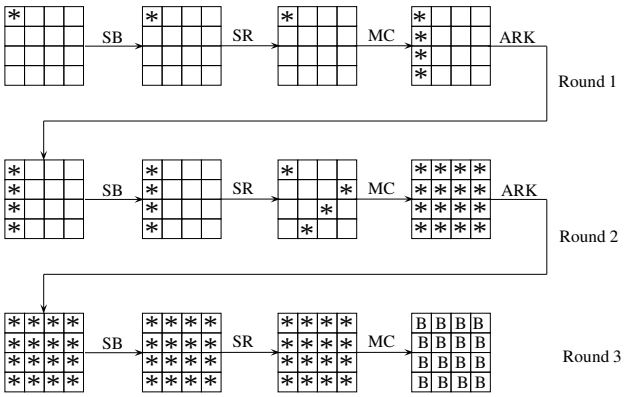


Figure 2.7: 3-round AES. Here, $*$ denotes the byte is active and B denotes the byte is balanced.

Since, S-box is a bijective mapping, hence intermediate state after S-Box (SB) operation in round 1 remains a λ-set. Further, as branch factor of AES MixColumns (MC) operation is 5, all the four bytes of the first column of the intermediate state after MC become active. Similar explanation holds true for second round transformations as well as transformation till before MixColumns (MC) operation in the third round. Let us consider the intermediate state after MixColumns operation in the third round. Let the bytes of the first column after third round MC be denoted as y_0^i, y_1^i, y_2^i and y_3^i where, i represents the i^{th} text. Let x_0^i, x_1^i, x_2^i and x_3^i represent the corresponding first column bytes before third round MC operation. Then, as per MixColumns operation definition [57]:

$$y_0^i = 02_x \cdot x_0^i \oplus 03_x \cdot x_1^i \oplus x_2^i \oplus x_3^i$$

If we xor all the values in the first byte position after the MixColumns operation, then it can be shown that the xor'ed sum is always zero as follows:

$$
\begin{aligned}
y_0^0 \oplus y_0^1 \oplus \ldots y_0^{255} &= 02_x \cdot x_0^0 \oplus 03_x \cdot x_1^0 \oplus x_2^0 \oplus x_3^0 \oplus \\
&\quad 02_x \cdot x_0^1 \oplus 03_x \cdot x_1^1 \oplus x_2^1 \oplus x_3^1 \oplus \\
&\quad \vdots \\
&\quad 02_x \cdot x_0^{255} \oplus 03_x \cdot x_1^{255} \oplus x_2^{255} \oplus x_3^{255}
\end{aligned}
$$

This can be re-written as:

$$
\begin{aligned}
y_0^0 \oplus y_0^1 \oplus \ldots y_0^{255} &= 02_x \cdot \bigoplus_{i=0}^{255} x_0^i \oplus 03_x \cdot \bigoplus_{i=0}^{255} x_1^i \oplus 01_x \cdot \bigoplus_{i=0}^{255} x_2^i \oplus 01_x \cdot \bigoplus_{i=0}^{255} x_3^i \\
&= 02_x \cdot 00_x \oplus 03_x \cdot 00_x \oplus 00_x \oplus 00_x \\
&= 00_x
\end{aligned}
$$

This shows that the set of values in the first byte position after third round MC operation forms a *balanced* set (set where XOR of all values becomes zero) and that too with probability 1. This property is preserved after key addition as well and holds true for all the other bytes as well. Thus, this property can be exploited by an attacker to distinguish 3-round AES from a random permutation, since in the case of random permutation, the probability that the xor'ed sum in all the bytes becomes zero is 2^{-128}.

Apart from AES, square attack was used to attack other ciphers as well, e.g., CRYPTON [64], IDEA [93] etc. Stefan Lucks extended the attack to non-SQUARE-like ciphers by attacking the block cipher Twofish [128] and named his generalized attack as *saturation attack*. Later, Knudsen and Wagner assembled the various square attack variants together into a single framework and termed it as *integral cryptanalysis* [111].

The Square attack can be considered as a specialized version of a broader class of attacks called the *multiset* attacks. A multiset is a list of values, each of which can appear multiple times, but the order in which they appear is irrelevant, e.g., {1, 2, 2, 3, 3, 4}. Apart from balanced multisets, these attacks can utilize other multisets as well [32]. Some other types of multisets are:

- *Constant Multiset*: A multiset M of m-bit values is a constant multiset if it consists of a single value repeated an arbitrary number of times.
- *Permutation Multiset*: A multiset M of m-bit values is a permutation multiset if it contains each of the 2^m possible values exactly once.

- *Even Multiset*: A multiset M of m-bit values is an even multiset if each value occurs an even number of times (including no occurrences as well).

The advantage of multiset attacks is the fact that the transformations of these multisets over a limited number of rounds is non-probabilistic as compared to differential and linear trails. Further, most of the components commonly found in a block cipher either preserve the multiset properties or convert them into some other multiset properties. E.g., a constant multiset remains so after passing through an S-box or a balanced multiset is preserved by any linear operation. These benefits allow an attacker to launch efficient key recovery attacks on a block cipher. Several variants of multiset attacks have been proposed in literature [60,61,71,79,111,127]. Currently, the second best attacks on Advanced Encryption Standard (in terms of number of rounds attacked) belong to a subclass of this line of attacks.

2.5 Block Cipher Based Hash Functions

Cryptographic hash functions, one of the widely used primitives in numerous standards and applications, are one-way functions that take a message of arbitrary length as input and produce a fixed length output called *message digest* or *hash*. The term one-way implies that such a function is easy to compute but computationally hard to invert, i.e., for a given output it is hard to find an input that maps to that output. The message digests produced by hash functions are typically used to verify the integrity of the data and detect unauthorized changes in it. In general, for a given message m, the sender computes its hash value and appends it with the message. The integrity of the hash value is protected in some manner. She then transmits the message and its appended hash over an insecure channel. At the receiver's side, the receiver recomputes the hash of the received message and checks if the computed hash matches with the received hash. If yes, then the receiver is ensured that the message has not been altered.

For a given hash function H which takes a message input m and produces a hash output h, i.e., $h = H(m)$, its security depends on the output length of the hash value h. A n-bit hash function H is said to be cryptographically secure, if it satisfies the following three properties:

- *Preimage Resistance* - For a given H and one of its hash output h, it is computationally infeasible to find a message m such that $H(m) = \mathrm{h}$.
- *Second Preimage Resistance* - For a given H and a message m, it is computationally infeasible to find another message m' (where $m' \neq m$), such that $H(m) = H(m')$.
- *Collision Resistance* - For a given H, it is computationally infeasible to find two messages m and m' (where $m' \neq m$), such that $H(m) = H(m')$.

By computationally infeasible we mean, that it should be intractable for an adversary to find a preimage or a second preimage in less than 2^n hash computations and a collision in less than $2^{n/2}$ hash computations.

In practice, most of the hash functions are designed as iterated hash functions which divide the whole input into some fixed size blocks and then process the blocks in an iterative way. The most common approach followed to construct an iterative hash function is the *Merkle-Damgård* (MD) construction [59,134]. In Merkle-Damgård algorithm, a hash input M of arbitrary length is first divided into fixed-length blocks M_i. This step often involves padding operation to ensure that the overall message length is a multiple of block length size. Each block M_i then serves as an input to an internal fixed-size n-bit function f called the *compression function*. The compression function then computes a new intermediate result as a function of the previous intermediate result and the message block m_i. This process is repeated until the entire message has been processed. For a hash function H, compression function f and message input M split into t message blocks, the MD-construction can be modeled as follows:

$$\begin{aligned} h_0 &= IV \\ h_i &= f(h_{i-1}, M_i), \qquad i \leq i \leq t \\ H(M) &= h(M_t) \end{aligned}$$

Here, h_{i-1} is called the *chaining variable* between round $(i-1)$ and round i and h_0 is called the *initializing vector* (IV) which is pre-defined and fixed. Since it has been proved that a MD based hash function is collision resistant if the underlying compression function is collision resistant, the majority of currently used hash functions like MD5 [143], SHA-1 [74], SHA-2 [73] etc. are based on Merkle-Damgård construction.

In addition to the classical security attacks on hash functions, there are few other slightly modified versions of these attacks that are also often considered. In these modified attacks, the IV value is assumed to be freely chosen by the attacker. Some of these attacks are listed below as follows:

1. *Pseudo-Preimage Attack*: Given a hash function H and one of the hash outputs h, the attacker finds a (IV', m) pair such that, $H(IV', m) = h$ and $IV' \neq IV$.

2. *Semi-Free Start Collision Attack*: Given a hash function H, the attacker finds two pairs (IV', m) and (IV', m') such that, $H(IV', m) = H(IV', m')$, $m \neq m'$ and $IV' \neq IV$. In this attack, the initialization vector chosen by the attacker, i.e., IV' is same for both the messages.

3. *Free Start Collision Attack*: Given a hash function H, the attacker finds two pairs (IV', m) and (IV'', m') such that, $H(IV', m) = H(IV'', m')$, $m \neq m'$ and

$IV' \neq IV'' \neq IV$. In this attack, the initialization vector chosen by the attacker is different for both the messages. Such an attack is also called a *pseudo-collision* attack.

All the six attacks discussed on hash functions above work on compression function as well, with IV, H and h being replaced by h_{i-1}, f and h_i respectively. Though these attacks do not reflect any direct weakness in the hash function design, they nonetheless show certificational weaknesses in them giving useful insights on the actual security provided by these hash functions [12]. E.g., if the compression function f of an n-bit iterated hash function H does not have brute-force security (2^n) against pseudo-preimage attacks, then preimages for H can be found in fewer than 2^n operations by adopting a meet-in-the-middle approach [133] as shown in Fig. 2.8.

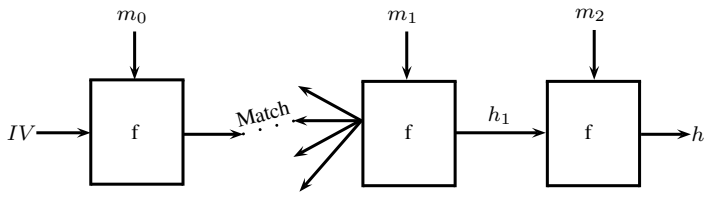
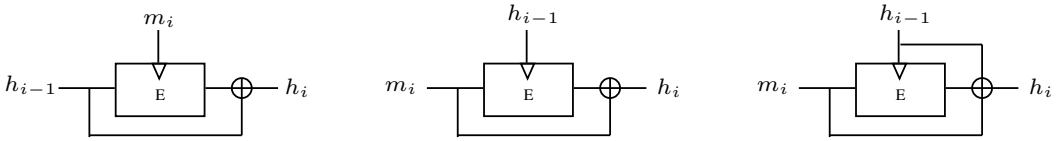


Figure 2.8: Finding preimage through MITM approach

Let us suppose that a pseudo-preimage $(h_0, (m_1, m_2))$ can be found with a complexity of 2^x (where, $x < n$). The attacker computes $2^{(n-x)/2}$ such pseudo-preimages and stores them in a table. He then computes $2^{(n+x)/2}$ $f(IV, m_0)$ for random m_0. With high probability, he'll get a value of m_0 for which $f(IV, m_0)$ matches with one of the entries stored in the table. That value of $m_0 \mid\mid m_1 \mid\mid m_2$ forms the preimage for hash function H. The complexity of this attack is $2^{(n-x)/2} \times 2^x + 2^{(n+x)/2} = 2 \cdot 2^{(n+x)/2}$.

Hash function construction using a block cipher as the underlying compression function is one of the most classical approach of designing iterated hash functions. The motivation behind constructing hash functions from block ciphers is that if an efficient implementation of a block cipher is already available then, a hash function can be easily built using that block cipher with little additional costs. This proves beneficial in settings like resource constrained environments. In the classical single block length (SBL) hash function, the hash output size is equal to the underlying block size. In [141], Preneel et al. studied 64 basic ways to construct a n-bit compression function from a n-bit block cipher (under a n-bit key) and mentioned 12 of them to be secure. These modes are commonly termed as PGV hash modes. Black et al. in [34] formally proved the security of these 12 constructions. The proofs are based on the assumption that the underlying block cipher is ideal. The three most popularly used PGV constructions are Davies-Meyer (DM), Matyas-Meyer-Oseas (MMO) and Miyaguchi-Preneel (MP) modes (as shown in Fig. 2.9).



(a) Davies-Meyer Mode (b) Matyas-Meyer-Oseas Mode (c) Miyaguchi-Preneel Mode

Figure 2.9: The three popular PGV modes

- *Davies-Meyer Mode*: DM mode is defined as: $h_i = E_{m_i}(h_{i-1}) \oplus h_{i-1}$. In this mode, the message input to the compression function acts as the key input of the underlying block cipher whereas the chaining variable acts as the plaintext input of the underlying block cipher.

- *Matyas-Meyer-Oseas Mode*: MMO mode is defined as: $h_i = E_{h_{i-1}}(m_i) \oplus m_i$. In this mode, the chaining variable acts as the key input of the underlying block cipher whereas message input acts as the plaintext input of the underlying block cipher.

- *Miyaguchi-Preneel Mode*: MP mode is defined as: $h_i = E_{h_{i-1}}(m_i) \oplus m_i \oplus h_{i-1}$. The construction is same as MMO mode except that in this mode, the chaining variable is also xor'ed with message and the ciphertext to produce the next chaining variable.

Apart from single block length hash functions, another research direction in case of block cipher based hash functions is the design and analysis of *double block length* (DBL) hash functions. In a double block length hash function, a block cipher with an n-bit block is used to compute a hash value of length $l = 2n$. Some of the popular DBL hash functions known are: MDC-2 [44], ABREAST-DM [117], TANDEM-DM [117], Hirose's construction [84] etc. For further details on double block length hash functions one can refer to [139]. In this thesis, we focus on single block length (SBL) based hash functions only.

2.5.1 Rebound Attack

Rebound Attack is one of the most efficient attack for generating collisions on hash functions constructed from AES-based primitives. The attack was first proposed by Mendel et al. [118] for the analysis of the AES-based hash functions Whirlpool [21] and Grøstl [78]. This attack consists of two main phases, called the *inbound* phase and the *outbound* phase as shown in Fig. 2.10.

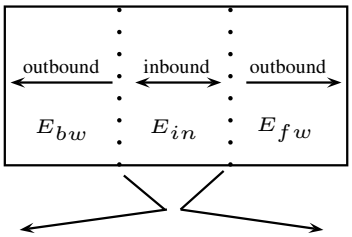


Figure 2.10: A schematic view of rebound attack.

The block cipher E (or, the compression function) is first divided into three parts - $E_{fw} \circ E_{in} \circ E_{bw}$.The part of the inbound phase is placed in the middle of the cipher and the two parts of the outbound phase are placed next to the inbound part. The main idea is to use high-differential sub-trails and connect these trails in the middle using the available degrees of freedom by choosing the values of the state. The key input to the block cipher is either fixed to a constant and known to the attacker or chosen by her. The rebound attack consists of the following 3 main parts [118]:

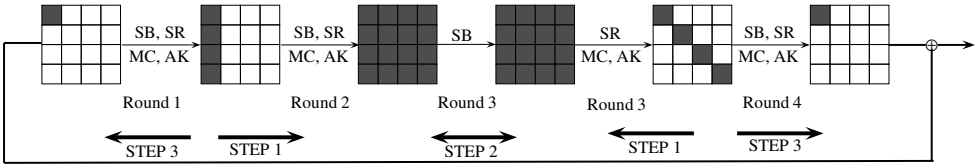


Figure 2.11: Rebound Attack on 4-round AES

1. *Constructing a truncated differential trail.* The attacker first chooses a truncated differential trail which preferably has a small number of active S-boxes and propagates it through the inbound phase. E.g., in case of 4-round AES as shown in Fig. 2.11, the attacker starts with 4-byte truncated differences at the end of round 1 and round 3 and propagates it forward and backward to the S-box layer of round 3.

2. *Inbound Phase.* She then tries to construct solutions (right pairs) for the middle part of the truncated differential trail. For a good trail, she should be able to construct many solutions for the inbound phase with a low average complexity. E.g., in Fig. 2.11, she tries to find solutions for the input-output difference of each of the 16 bytes of the S-box layer in round 3.

3. *Outbound Phase.* In this phase, the truncated differential trail is extended both in the forward and backward directions in a probabilistic way. The solutions obtained in the inbound phase are propagated outwards in both the directions and the attacker tries to link the beginning and the end of the trail using the feed-forward operation of the hash function. To ensure that atleast one solution is

obtained, the differential trails in the outbound phase are so chosen such that they have high probability of propagation. E.g., in Fig. 2.11, the inbound truncated differential trail (from round 2 till round 4) $4 \rightarrow 16 \rightarrow 4$ (which has probability 1) [4] is extended to the outbound phase as $1 \leftarrow 4 \rightarrow 16 \rightarrow 4 \rightarrow 1$ (with probability $2^{-(24+24)} = 2^{-48}$).

Apart from the AES-like primitives, the rebound attacks have been applied to other structures as well. For example, it has been applied to the ARX based hash function Skein [102] and to the 4-bit S-box based design Luffa [101]. *Differential Enumeration Technique* proposed by Dunkelman et al. in [71] also uses concepts similar to the inbound phase of the rebound attack for launching key recovery attacks on AES.

[4] $4 \rightarrow 16 \rightarrow 4$ depicts the active S-boxes in Round 2, 3 and 4 respectively.

Chapter 3

Improved Biclique Cryptanalysis of AES

In this chapter, we discuss a new variant of meet-in-the-middle attack termed as *Biclique Cryptanalysis*. Biclique cryptanalysis was first introduced by Khovratovich et al. in [103] for preimage attack on hash functions Skein [136] and SHA-2 [73]. The concept was then carried over by Bogdanov et al. for launching key recovery attacks on AES [39]. Its application to AES garnered considerable interest amongst the cryptographic community, since it was the first attack that challenged the theoretical security of full AES in the single key model. Biclique key recovery is based on the meet-in-the-middle approach at its core but borrows an important twist - *initial structures*[1] - from the domain of hash function cryptanalysis. Namely, biclique cryptanalysis uses the fact that, for some ciphers such as AES, the adversary can efficiently prepare structures of internal states that cover many keys. The work [39] scrutinized the notion of initial structures for block ciphers, formalized it to bicliques (complete bipartite graphs) which are efficient to construct and proposed key recovery attacks with computational complexities below brute force for all three variants of the full AES. This attack may be considered as an advancement in the field of symmetric-key cryptography but it has been prepared by a considerable number of works in the area of meet-in-the-middle (MITM) attacks on block ciphers [41, 43, 51, 70, 89] and hash function cryptanalysis [17, 18, 80]. Since the introduction of bicliques, an entire line of research emerged aiming to apply the technique to various block ciphers [4, 5, 52, 53, 86, 90, 100, 129, 174]. In this chapter, we discuss biclique attack on block ciphers in detail and present new improved results on AES.

This chapter is organized as follows: In Section 3.1, we introduce biclique attack on block ciphers giving an high level overview of how bicliques are constructed. In Section 3.2, we describe how this attack works in general. Section 3.3, describes AES in detail and discusses the existing biclique attacks on it in literature. This is fol-

[1]The concept of initial structures will be discussed briefly in the next chapter

lowed by Section 3.4, where we present the biclique key recovery attacks existing on other block ciphers. We then move on to Section 3.5, where we discuss the problems existing with the current biclique attack on AES. In Section 3.6, we introduce the concept of stars followed by Section 3.7, where we utilize star based bicliques to launch key recovery attacks with lowest possible data complexity on all three AES variants. We, next discuss the automated search technique deployed by us to find star based bicliques as well as other promising class of biclique attacks in Section 3.8. Section 3.9 demonstrates biclique attacks that have optimal time complexities for all the three AES variants whereas Section 3.10 demonstrates biclique attacks that have the fastest time complexities for all the three AES variants. In the concluding section, we summarize the whole chapter. The original contribution of this thesis is from Section 3.5 to Section 3.10.

3.1 Framework of Biclique Key Recovery Attack

In biclique attack, the aim of the attacker is to construct a biclique structure (a complete bipartite graph) over some rounds. Meet-in-the-middle procedure is performed on the rest of the rounds. A biclique connects 2^d pairs of intermediate states with 2^{2d} keys for some value d. The main source of computational advantage in the key recovery attack comes from the fact that by constructing a biclique on 2^d vertices, one can cover 2^{2d} keys in time 2^d rather than 2^{2d}. Here, d is called the *dimension* of the biclique. This dimension is related to the cardinality of the biclique elements, a higher cardinality implies a higher advantage over brute force. A biclique is characterized by its length (number of rounds covered) and dimension d.

3.1.1 What is a biclique structure on block ciphers ?

Let f be a subcipher that maps an internal state S to a ciphertext C under the key K, i.e., $f_K(S) = C$. Suppose f connects 2^d intermediate states $\{S_j\}$ (for, $0 \leq j \leq 2^d - 1$) to 2^d ciphertexts $\{C_i\}$ (for, $0 \leq i \leq 2^d - 1$) with 2^{2d} keys $\{K[i, j]\}$ where,

$$\{K[i,j]\} = \begin{bmatrix} K[0,0] & \cdots & K[0,2^d-1] \\ \vdots & \vdots & \vdots \\ K[2^d-1,0] & \cdots & K[2^d-1,2^d-1] \end{bmatrix}.$$

The 3-tuple of sets $[\{S_j\}, \{C_i\}, \{K[i, j]\}]$ is called a d-dimensional biclique (as shown in Fig. 3.1), if,

$$\forall i, j \in \{0, \ldots, 2^d - 1\} : C_i = f_{K[i,j]}(S_j).$$

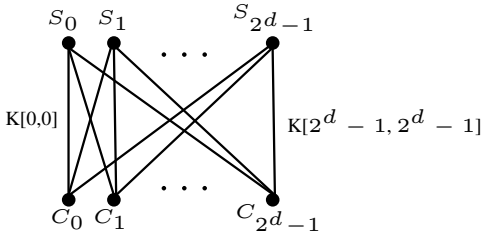


Figure 3.1: d-dimensional biclique.

i.e., in a biclique, the key $K[i, j]$ maps the internal state S_j to the ciphertext C_i and vice versa.

3.1.2 Construction of biclique

To construct a biclique, firstly, the whole key space is partitioned into 2^{k-2d} groups of 2^{2d} keys each, where size of secret key K is k bits. Each key in a group can be represented relative to the base key of the group i.e., $K[0, 0]$ and two key differences Δ_i^k and ∇_j^k such that:

$$K[i, j] = K[0, 0] \oplus \Delta_i^k \oplus \nabla_j^k$$

For each group, let the intermediate state S_0 be transformed to the ciphertext C_0 under the key $K[0, 0]$:

$$S_0 \xrightarrow[f]{K[0,0]} C_0 \tag{3.1}$$

This computation is called the *base computation*. Then, 2^d Δ_i differentials in the forward direction are constructed from S_0 as follows:

$$S_0 \xrightarrow[f]{K[0,0]\oplus\Delta_i^k} C_i \quad \text{or,} \quad 0 \xrightarrow[f]{\Delta_i^k} \Delta_i \tag{3.2}$$

Similarly, 2^d backward differentials ∇_j from C_0 are constructed as follows:

$$S_j \xleftarrow[f^{-1}]{K[0,0]\oplus\nabla_j^k} C_0 \quad \text{or,} \quad \nabla_j \xleftarrow[f^{-1}]{\nabla_j^k} 0 \tag{3.3}$$

If the above two differentials are *independent*, i.e., they do not share any active non-linear components for all i and j, then the following relation is satisfied:

$$S_0 \oplus \nabla_j \xrightarrow[f]{K[0,0]\oplus\Delta_i^k\oplus\nabla_j^k} C_0 \oplus \Delta_i \quad \text{or,} \quad \nabla_j \xrightarrow[f]{\Delta_i^k\oplus\nabla_j^k} \Delta_i \quad \forall i, j \in \{0, \ldots, 2^d - 1\} \tag{3.4}$$

Finally, Eq. 3.4 can be written as:

$$S_j \xrightarrow[f]{K[i,j]} C_i \tag{3.5}$$

where,

$$\begin{aligned} S_j &= S_0 \oplus \nabla_j \\ C_i &= C_0 \oplus \Delta_i \\ K[i,j] &= K[0,0] \oplus \Delta_i^k \oplus \nabla_j^k \end{aligned}$$

3.1.2.1 Relation between Biclique Construction and Boomerang Attack

The proof for Eq. 3.4 comes from the theory of boomerang attacks as follows: Consider the boomerang distinguisher constructed in Fig. 3.2. Here, the quartet states (u, v, x, w) can be represented as (u, $u \oplus \phi$, $u \oplus \beta$, $u \oplus \phi \oplus \beta$) respectively as shown in Fig. 3.3

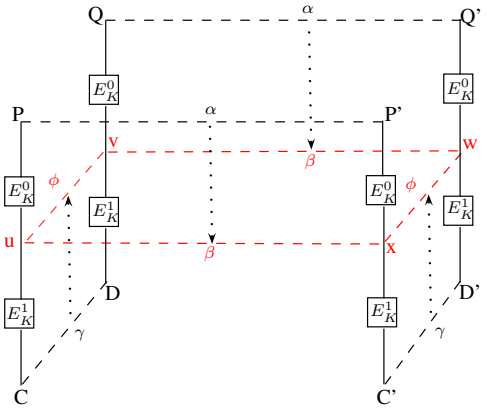


Figure 3.2: A boomerang distinguisher.

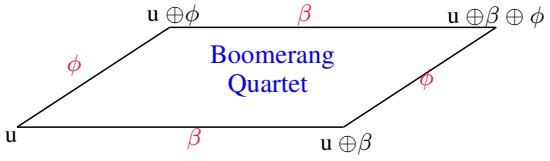
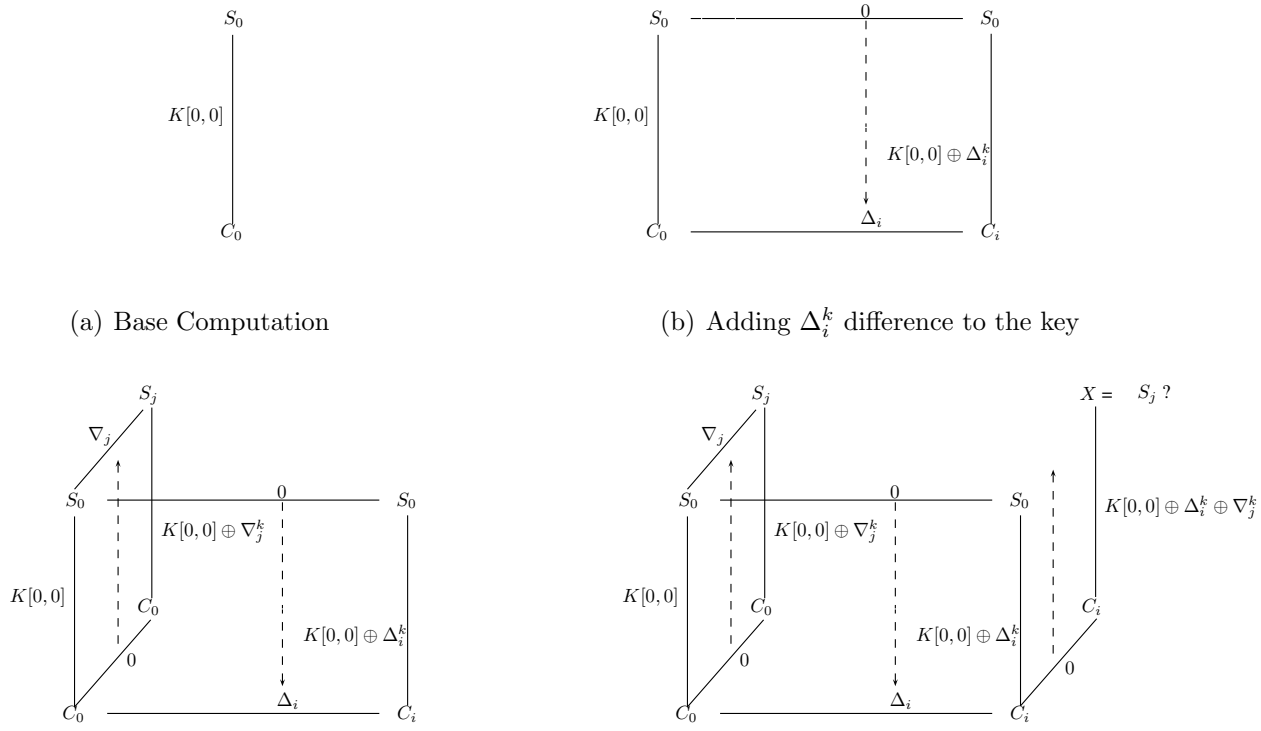


Figure 3.3: Boomerang Quartet.

The steps of biclique construction (from eqs. (3.1) to (3.5)) can be illustrated as shown in Fig. 3.4.



(a) Base Computation

(b) Adding Δ_i^k difference to the key

(c) Adding ∇_j^k difference to the key

(d) Biclique Structure

Figure 3.4: Formation of boomerang rectangle while constructing a biclique

The probability that $X = S_j$ is 1 (in Fig. 3.4(d)), only if both the Δ_i trail ($0 \rightarrow \Delta_i$) and ∇_j trail ($0 \rightarrow \nabla_j$) do not share any active non-linear component. This is because, if both the trails are independent, then a boomerang quartet (similar to Fig. 3.3) is formed at every step (as shown in Fig. 3.5) with probability 1.

Otherwise, if both the trails share some non-linear component (say S-box), then a boomerang quartet may or may not form. For example, as shown in Fig. 3.6, for a given S-box, say SB:

$$SB(s) \oplus SB(s \oplus c7) \oplus SB(s \oplus 02) = SB(s \oplus s \oplus c7 \oplus s \oplus 02)$$

the above equation will be satisfied only for some value of s. Depending upon, whether this value of s is obtained during the attack determines whether the boomerang quartet is formed at S-box operation step. Therefore, probability of boomerang quartet

formation at this step is not 1. Hence, independence of Δ_i and ∇_j trails is needed to ensure that a boomerang based on these trails always returns from the ciphertext with probability 1.

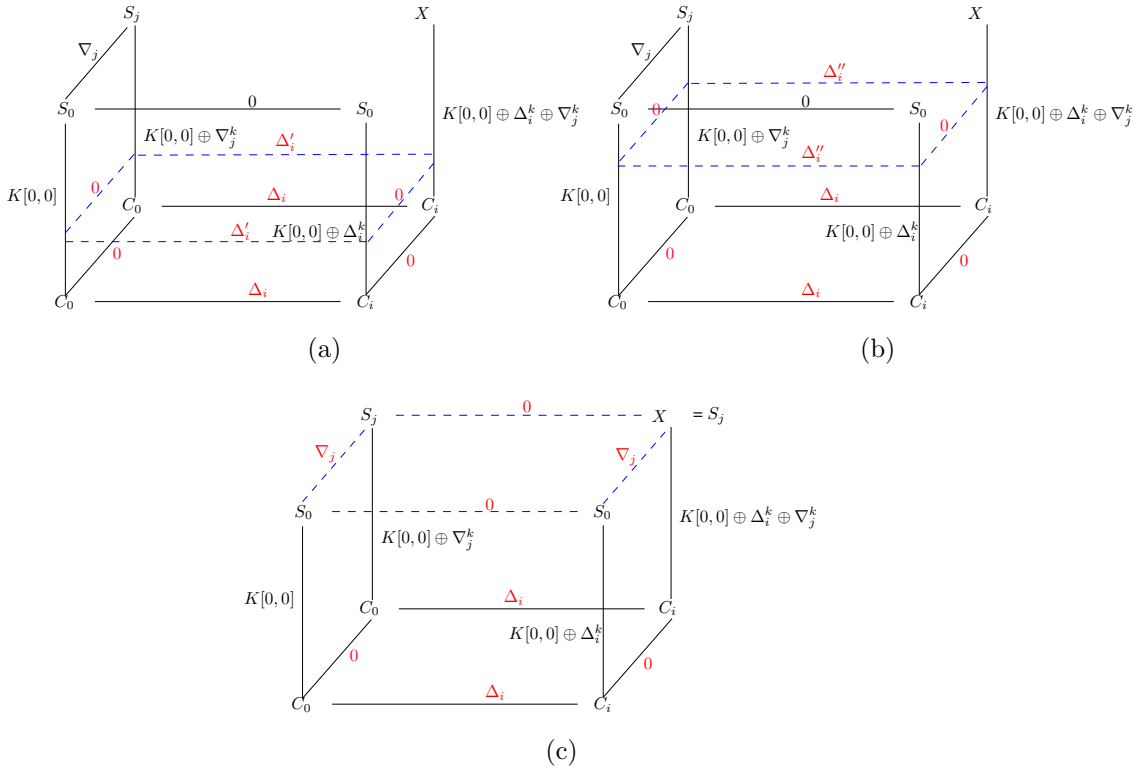


Figure 3.5: Formation of boomerang rectangle at every step of the construction of a biclique

The constraint on the two trails being independent limits the length of the biclique. The longer the trails, the higher are their chances of being dependent.

Complexity of biclique construction. Let the total rounds in block cipher $E_K(\cdot)$ be r. Let the number of rounds covered in the biclique phase be x. For each key group in the biclique phase, since $\Delta_i \neq \nabla_j$ and Δ_i trails are independent of ∇_j trails, the construction of biclique is simply reduced to computation of Δ_i and ∇_j trails independently which requires no more than 2.2^d computations of f, i.e.,

$$\text{Complexity of biclique phase} = 2^d \times \frac{x}{r} + 2^d \times \frac{x}{r} = 2^{d+1}(\frac{x}{r}) \ll 2^{2d}(\frac{x}{r}).$$

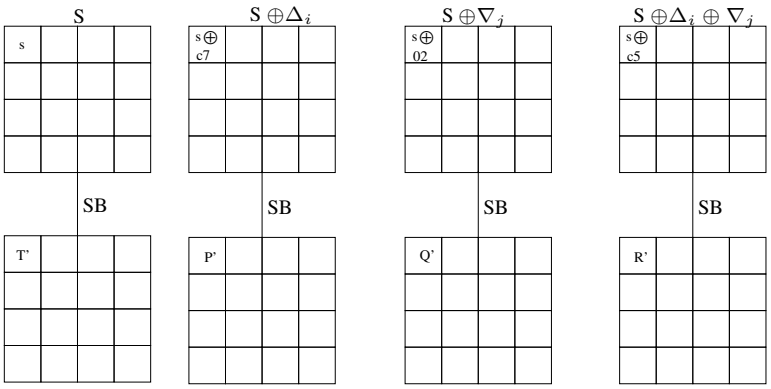


Figure 3.6: Boomerang rectangle may not form at the S-box (non-linear) operation with probability 1

3.2 Steps of the Biclique Attack

Let the block cipher $E_K(\cdot)$ be defined as a composition of two subciphers: $E_K(\cdot) = f \circ g$.

1. For each group of keys, the attacker first builds a biclique structure of 2^d ciphertexts C_i which map to 2^d intermediate states S_j with respect to the 2^{2d} keys over the subcipher f.

2. She then obtains plaintexts P_i from ciphertexts C_i through the decryption oracle.

3. She then chooses an intermediate state v in the subcipher g such that computation of v in one direction is independent of Δ_i trail and is independent of ∇_j trail in the other direction.

4. She then checks if a key in a group satisfies the following relation (as illustrated in Fig. 3.7):

$$P_i \xrightarrow[r]{K[i,\cdot]} \overrightarrow{v} = \overleftarrow{v} \xleftarrow[t^{-1}]{K[\cdot,j]} S_j \tag{3.6}$$

   where, $g = r \circ t$

5. If a key in a group satisfies Eq. (3.6), then the attacker proposes the key as a candidate key. If a right key is not found in the current group, then another group is chosen and the whole process is repeated.

We now move on to discussing the existing biclique based key recovery attacks on AES and other block ciphers.

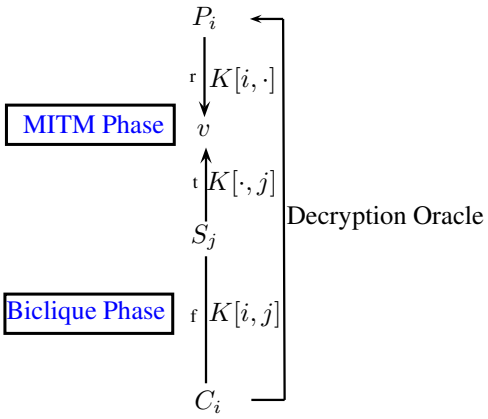


Figure 3.7: Biclique attack in a nutshell

3.3 Biclique Attacks on AES

We first start with a brief description of AES to help understand the subsequent sections. For full information on AES, one can refer to [57]

3.3.1 Description of AES

The block cipher Rijndael was designed by Joan Daemen and Vincent Rijmen and standardized by NIST in 2000 as the Advanced Encryption Standard (AES) [137]. AES adopts the classical substitution-permutation network structure and defines 3 key sizes: 128-bit, 192-bit and 256-bit with the block size limited to a fixed 128-bit size for all the three alternatives. By design, AES is byte-oriented and follows operations in GF(2^8). Each AES variant has different number of rounds per full encryption, i.e., 10, 12 and 14 rounds for AES-128, AES-192 and AES-256 respectively. AES operates on a state array of 4×4 byte matrix and a key array of 4×4, 4×6 and 4×8 byte size respectively. Each round consists of 4 steps: SubBytes, ShiftRows, MixColumns and AddRoundKey. In SubBytes (SB), which is the only non linear layer, each byte in a state is replaced by another byte from an invertible AES S-box. In ShiftRows (SR), the bytes of row m are rotated to the left by m positions with $m \in \{ 0, 1, 2, 3\}$. In MixColumns (MC), which is a linear diffusion layer and works columnwise, the state array obtained after ShiftRows is multiplied (in GF(2^8)) to a fixed and invertible 4 x 4 MDS matrix as shown below.

$$\begin{pmatrix} 2 & 3 & 1 & 1 \\ 1 & 2 & 3 & 1 \\ 1 & 1 & 2 & 3 \\ 3 & 1 & 1 & 2 \end{pmatrix}$$

The most important property of the MixColumns transformation is its branch num-

ber. Branch number of any linear transformation is defined as the minimum number of non-zero active bytes (in any differential trail) at the input and output differences of the transformation. For AES MixColumns operation, the branch number is 5. This implies that if a state is applied to MixColumns operation with a single active byte, then the output state will have atleast 4 active bytes. At last, the final state transformation is xor'ed with the 16 byte subkey (AK). Each round follows the same steps as listed above except the last round where MixColumns operation is not performed. A pre-whitening key is also added prior to the first round.

The round subkeys in each round are generated through a key schedule algorithm. The key schedule of AES recursively generates a new 128-bit round key K_i from the previous round key. In case of AES-128, the pre-whitening key K_0 is the 128-bit master key of AES-128. The key schedule algorithm of AES-128 takes as input a 4-word [2] master key and expands it to a linear array of 44 words. It works as follows:

1. The secret key is first copied into the first four words of the expanded key. The rest of the expanded key is filled four words at a time.
2. Each word $w[i]$ (where, $4 \leq i \leq 43$) depends on the immediately preceding word, $w[i-1]$ and the word four positions back, $w[i-4]$.
3. For a word whose position in the array is a multiple of 4, a complex function t (as shown in Fig. 7.4) is used. The function t consists of the following sub-functions:
   - A one-byte circular left shift on a word.
   - This is followed by byte substitution on each byte of the input word using the AES S-box.
   - The result of above two steps is then xor'ed with a round constant.
4. In rest of the words, a simple XOR is used.

Fig. 7.4 shows the generation of first eight words of the expanded key. The key schedule algorithm of AES-192 and AES-256 also works similarly. For further information on AES, one can refer [57].

For describing the biclique attack on AES, we follow the following notations as adopted in [39]. Briefly, in a differential trail, the state in the r^{th} round is denoted by S_r. S_r^{SB}, S_r^{SR}, S_r^{MC} and S_r^{AK} represent the state S_r after SB, SR, MC and AK operations respectively. In a differential trail, # 1, #2 represent the state before SubBytes and after MixColumns for Round 1, #3, #4 represent the state before SubBytes and after MixColumns for Round 2 and so on. The 128-bit subkeys are denoted as \$0, \$1, \$2 .... and so on. Bytes are addressed column-wise (0-3#first column), (4-7#second column), (8-11#third column) and (12-15#fourth column). The i^{th} byte in state S is represented as S_i. The i^{th} byte in subkey \$K is represented as $\$K_i$.

[2] One word = 32-bits

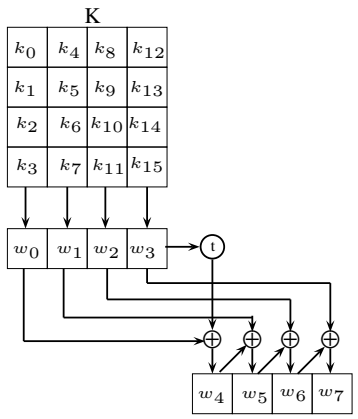


Figure 3.8: AES-128 Key Expansion

3.3.2 Precomputation Technique for the Matching Part of Meet-in-the-Middle Attack

In Section 3.2, it was mentioned that the attacker would try to find an intermediate state which involves independent computation of chunks in either direction. However, non-linear key schedule of AES prevents finding such intermediate states with the desired property. Therefore, the attacker is forced to do brute-force search over rest of the rounds (not covered by biclique) for finding the correct key, i.e.,

$$P_i \xrightarrow[r]{K[i,j]} \overrightarrow{v} = \overleftarrow{v} \xleftarrow[t^{-1}]{K[i,j]} S_j,$$

However, Bogdanov et al. in [39] suggested *matching with precomputations* to reduce the brute-force complexity in MITM stage. In this approach, the attacker first *precomputes* and stores in memory 2^{d+1} full computations upto the matching state v:

$$\forall i, P_i \xrightarrow{K[i,0]} \overrightarrow{v} \quad \text{and} \quad \forall j, \overleftarrow{v} \xleftarrow{K[0,j]} S_j.$$

Since in a group $K[i,0]$ and $K[i,j]$ values only change at parts affected by Δ_i^k (similar property is true for $K[0,j]$ and $K[i,j]$), for any particular i and j, the adversary checks the matching at v by *recomputing* only those parts of the cipher which differ from the stored ones. The amount of recomputation depends on the diffusion properties of both the internal rounds and the key schedule of the cipher. The recomputations are estimated in terms of S-box calculations. The complexity of biclique attack in case of AES is estimated as:

$$C_{full} = 2^{k-2d}(\underbrace{C_{biclique} + C_{precomp} + C_{recomp} + C_{falsepos}}_{Time-Complexity})$$

where,

- 2^{k-2d} denote total key groups.
- $C_{biclique}$ is the complexity of constructing a biclique.
- $C_{precomp}$ is the complexity of calculating v for 2^{d+1} times.
- C_{recomp} is the complexity of recomputing v for 2^{2d} times.
- $C_{falsepos}$ is the complexity to eliminate false positives.

As mentioned in [39], the full key recovery complexity is dominated by $2^{k-2d} \times C_{recomp}$.

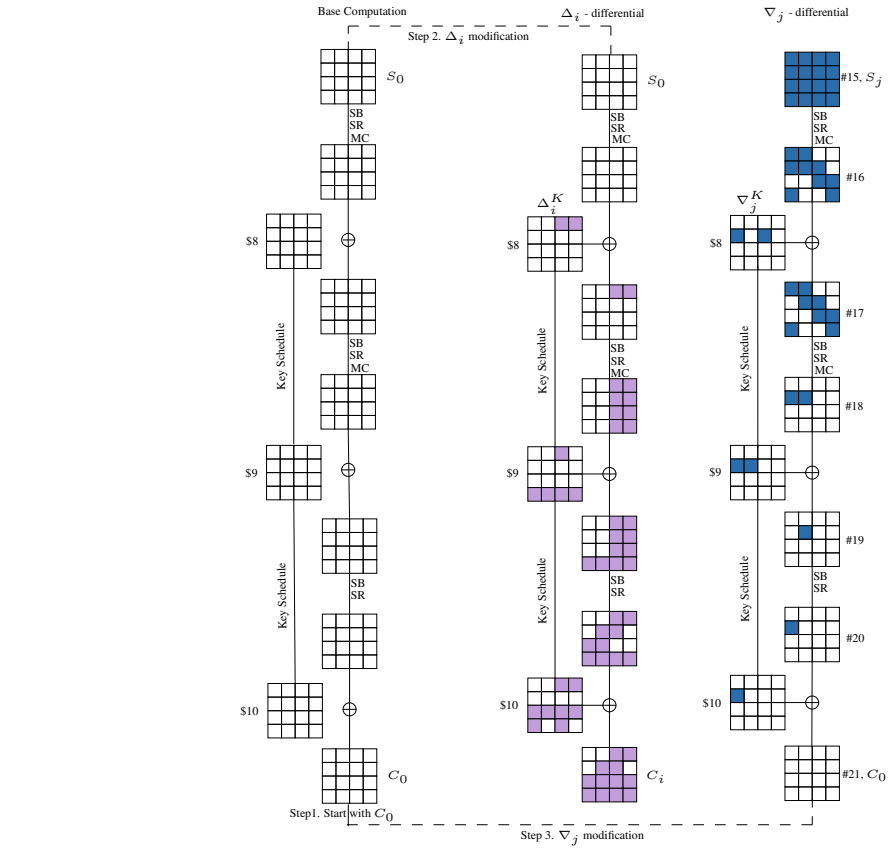
Previous Biclique Attacks on AES-128. Equipped with this technique, the biclique key recovery attack on AES-128 in [39] succeeded with a time complexity of $2^{126.1}$. An example of biclique constructed on AES-128 in [39] along with recomputations is shown in Fig. 3.9. Round 8 subkey (denoted as \$8) is taken as the base key here with $\Delta_i^k = (0\ 0\ 0\ 0 \mid 0\ 0\ 0\ 0 \mid i\ 0\ 0\ 0 \mid i\ 0\ 0\ 0)$, $\nabla_j^k = (0\ j\ 0\ 0 \mid 0\ 0\ 0\ 0 \mid 0\ j\ 0\ 0 \mid 0\ 0\ 0\ 0)$ and $(0 \leq i, j \leq 2^8)$ [3]. Thus, a biclique of dimension, $d = 8$ is constructed with $2^{128-16} = 2^{112}$ key groups where, total keys in one key group $= 2^{16}$.

As can be seen in Fig. 3.9(a), Δ_i and ∇_j trails are independent and do not share any active non-linear component (i.e., S-boxes with non-zero difference) between them. Thus, a biclique over last 2.5 rounds is formed. Figs. 3.9(b) and 3.9(c) show the recomputations performed in the backward and forward direction respectively. The 12^{th} byte of #5 intermediate state is taken as the matching variable v here. Let us look at how the forward computation $P_i \xrightarrow{K[i,j]} \overrightarrow{v}$ differs from the stored one $P_i \xrightarrow{K[i,0]} \overrightarrow{v}$. It can be seen that the difference between the above two computations is determined by the influence of the difference between the keys $K[i,j]$ and $K[i,0]$, when applied to the plaintext P_i. Due to difference inducted in the base key \$8, the pre-whitening subkeys of $K[i,j]$ and $K[i,0]$ differ only in 9 bytes. Due to this difference, a total of $9 + 4 = 13$ S-boxes need to be recomputed in the forward direction. For similar reasons, the number of S-box recomputations in the backward direction involve $4 + 16 + 16 + 4 + 1 = 41$ S-Boxes computations. Thus, a total of 54 S-box recomputations are required. One full AES-128 requires 200 S-boxes computations [4]. As each group has 2^{16} keys, therefore, for each group, $C_{recomp} = 2^{16} \times \frac{54}{200} = 2^{14.14}$. Since we match on 1 byte, i.e., 8-bits in v, we have 2^8 false positives on an average. The cost of biclique construction is calculated as $C_{biclique} = 2^9 \times \frac{2.5}{10} = 2^7$ and $C_{precomp} = 2^9 \times \frac{7.5}{10} = 2^{8.58}$. Hence, total time complexity of this attack is:

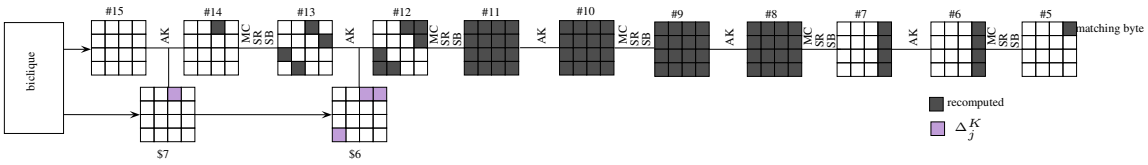
$$C_{full} = 2^{112} \times (2^7 + 2^{8.58} + 2^{14.14} + 2^8) \approx 2^{126.1}.$$

[3] The 16 byte subkey K for any round can be represented bytewise in the following form: $K = (K_0K_1K_2K_3 \mid K_4K_5K_6K_7 \mid K_8K_9K_{10}K_{11} \mid K_{12}K_{13}K_{14}K_{15})$

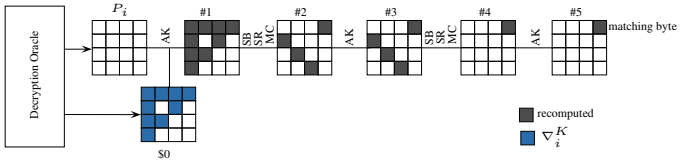
[4] 160 S-boxes in state update + 40 S-boxes in key schedule



(a) Biclique over last 2.5 rounds



(b) Recomputations in backward direction



(c) Recomputations in forward direction

Figure 3.9: Biclique attack on AES-128 with time $2^{126.16}$ [39].

In all 2^{112} groups, as shown in Fig. 3.9(a), C_i's differ only in 12 bytes. Out of these 12 bytes, 2 bytes - C_{10} and C_{14} are always equal. Hence effectively C_i's differ

only in 11 bytes. As a result, the data complexity of the attack does not exceed 2^{88} ciphertexts. The memory complexity of the attack is estimated by the storage of 2^8 full computations of g which in this case is $2^8 \times 7 \times 16$ bytes $\approx 2^{15}$ bytes.

The biclique attack demonstrated in [39] on AES-128 had a data complexity of 2^{88} which is very high. Bogdanov et al. in [38] constructed a new biclique on AES-128 which had a significantly lower data complexity of 2^4. However, in this case the dimension of the biclique was reduced to $d = 2$ (as compared to $d = 8$ in [39]) leading to an increase in the time complexity of $2^{126.89}$. In [5], Abed et al. presented an automated framework *Janus* which searched bicliques programmatically. Their tool accepted user-chosen set of parameters, e.g., number of rounds covered by biclique, dimension of the biclique, strategy to generate key differences etc. and provided the best biclique structure with minimum time complexity to the user along with a pictorial rendering of the same. They applied their tool to find bicliques for block ciphers such as AES, ARIA and BKSQ. Their work also highlighted an error in computational complexity reported for AES-192 in [39]($2^{189.74}$) which was then corrected to $2^{190.16}$. They also showed biclique attacks on AES-128 and AES-192 with reduced data complexity as compared to original attack in [39]. These values are reported in Table 3.1.

In [48], a new technique known as *Sieve-in-the-middle*(SIM) was proposed by Canteau et al. This technique differs from the traditional meet-in-the-middle process in the sense that it searches for the existence of valid transitions through some middle S-box instead of matching at some intermediate state. Canteaut et al. [48] presented analysis of sieve-in-the-middle process on many block ciphers including AES-128 and showed that for AES-128 there is a decrease in the total time complexity from $2^{126.1}$ to $2^{125.69}$ when applied with biclique attack. The application of this technique essentially involves choosing a set of intermediate states which will form a super S-box. A super S-box is a term used to represent two rounds of AES that have only one non-linear layer in between them. Formally, a two round AES can be written as:

$$SB \rightarrow SR \rightarrow MC \rightarrow AK \rightarrow SB \rightarrow SR \rightarrow MC \rightarrow AK$$

or,

$$SR \rightarrow \underbrace{SB \rightarrow MC \rightarrow AK \rightarrow SB}_{SuperS-box} \rightarrow SR \rightarrow MC \rightarrow AK \ ^{5}$$

where Super S-box = $SB \rightarrow MC \rightarrow AK \rightarrow SB$. A look-up table for that super S-box (say SS), is then constructed where all its possible input-output transitions (i.e., x, y where $y = \mathrm{SS}(x)$) are precomputed and stored. For each ($K[i,0]$,$K[0,j]$) pair, where $K[i,0]$ forms the key for the chunk in the forward direction and $K[0,j]$ forms the key for the chunk in the backward direction, the input state x is calculated by the forward computation and the output state y is computed by the backward computation. It

[5]Note that Sub Bytes and Shift Row operations in the first round have been interchanged as these functions commute with each other

is then checked through table lookup if a valid transition from $x \mapsto y$ exists. If not, then the corresponding key pair is discarded and another $(K[i,0],K[0,j])$ pair is picked up for testing. The process iterates until a valid key pair is obtained. This saves the recomputation of S-boxes involved in the super S-box each time leading to a slight decrease in the overall cost complexity. [6]

We illustrate the application of this process on AES-128. Let us consider the backward recomputations of AES-128 (as shown in Fig. 3.10) discussed in [39].

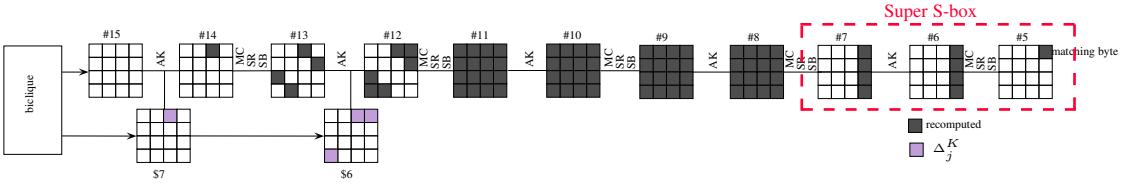


Figure 3.10: Backward Recomputations in AES-128 in [39]

In this case, let us further consider states #11 to #14 (as shown in Fig. 3.11). The states enclosed in the rectangle form the super S-box (of size 32 x 32). It can be seen in Fig. 3.11 that the super S-box is key dependent i.e., each 32-bit output of the super S-box depends on 32-bit input and 32-bit key. Hence for each guess of the 2^{32} values of key bits, a lookup table having 2^{32} entries is constructed where all 32-bit input-output transitions: $x \mapsto y$ are precomputed and stored.

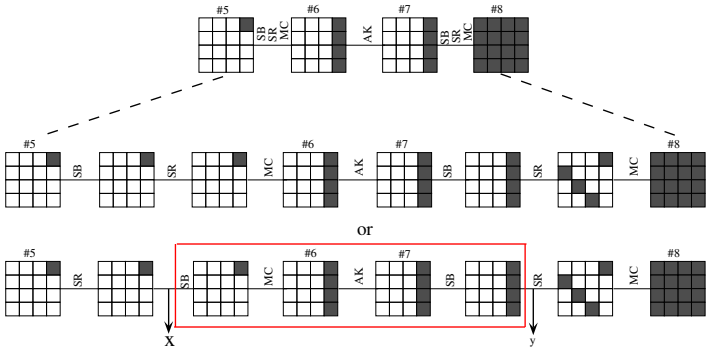


Figure 3.11: Super S-box.

In the recomputation stage of biclique attack, S-boxes till the state just before #5 are recomputed in the forward computation. In the backward computation, S-boxes till

[6]Many a times, during forward and backward computations, instead of calculating full (x, y), partial intermediate states (u,v) where u is a m-bit $(m \leq |x|)$ part of x and v is a p-bit $(p \leq |y|)$ part of y respectively are computed. It is then checked if the given (u,v) pair forms a valid part of some (x, y) by table lookup [48].

state #8 are recomputed. Then, for each biclique group we will choose the table corresponding to the 32-bits of the base key involved in the Super S-box for that biclique. Through lookup table it is then checked if a valid transition from $\#5_{12} \mapsto \#7_{12,13,14,15}$ exists. If such a transition exists, the corresponding $(K[i,0], K[0,j])$ key pair forms a valid candidate. Through this process, 5 S-boxes, (i.e., $\#5_{12}$, $\#7_{12}$, $\#7_{13}$, $\#7_{14}$, $\#7_{15}$) need not be recomputed each time. Hence, instead of 54 S-boxes (calculated in [39]), only 49 S-boxes need to be recomputed in all. This translates to a computational complexity of $2^{125.98}$ instead of $2^{126.1}$. [7]The same procedure can be applied to all other bicliques of AES-128, AES-192 and AES-256 but in [48], it was only shown on AES-128. It was also pointed out in [48] that this attack is faster only in those platforms where lookup in a table of size 2^{32} is faster than five S-box evaluations.

Table 3.1 summarizes all the existing biclique attacks on AES-128 and its other variants.

Table 3.1: Key recovery with bicliques for full AES. The terms CC and CP denote chosen ciphertext and chosen plaintext respectively.

Algorithm	Rounds	Data Complexity	Time Complexity	Biclique length (rounds)	Ref.
AES-128	10	2^{88} CC	$2^{125.69\dagger}$	2.5	[48]
	10	2^{88} CC	$2^{126.16}$	2.5	[39]
	10	2^{72} CC	$2^{126.72}$	2.5	[5]
	10	2^{4} CP	$2^{126.89}$	2.5	[38]
AES-192	12	2^{80} CC	$2^{190.16}$	3.5	[39] [5]
	12	2^{48} CC	$2^{190.28}$	3.5	[5]
AES-256	14	2^{40} CC	$2^{254.42}$	3.5	[39]
	14	2^{64} CC	$2^{254.53}$	3.5	[5]

† Our analysis estimates the cost as $2^{125.98}$.

3.4 Biclique attack on other block ciphers

Soon after the introduction of bicliques on AES, an entire line of research emerged aiming to apply the technique to various other block ciphers for recovering the key. In Table 3.2, we summarize all the biclique attack results existing on these other block

[7]In [48], the attack complexity for AES-128 is mentioned as $2^{125.69}$, however we could not validate it. Our analysis estimates this complexity to be $2^{125.98}$

ciphers as well as on AES-192 and AES-256.

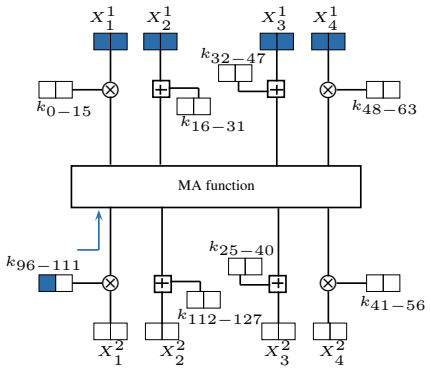
Algorithm	Rounds (full)	Data Complexity	Time Complexity	Biclique length (rounds)	Ref.
PRESENT-80	31	2^{25}	$2^{79.49}$	3	[4]
	31	2^{23}	$2^{79.76}$	3	[92]
PRESENT-128	31	2^{23}	$2^{127.32}$	4	[4]
	31	2^{19}	$2^{127.81}$	4	[92]
LED-64	32	2^{8}	$2^{63.58}$	4	[4]
LED-128	48	2^{8}	$2^{127.42}$	8	[4]
	48	2^{64}	$2^{127.37}$	8	[92]
KLEIN-64	12	2^{39}	$2^{62.81}$	3	[9]
KLEIN-80	16	2^{48}	2^{79}	3	[4]
KLEIN-96	20	2^{32}	$2^{95.18}$	3	[4]
PICCOLO-80	25	2^{24}	$2^{79.1}$	6	[92]
PICCOLO-128	31	2^{24}	$2^{127.35}$	7	[92]
ARIA-256	16	2^{80}	$2^{255.2}$	2	[52]
BKSQ-96	10	2^{80}	$2^{94.47}$	3	[5]
BKSQ-144	14	2^{96}	$2^{142.63}$	4	[5]
BKSQ-192	18	2^{96}	$2^{190.78}$	5	[5]
TWINE-80	36	2^{60}	$2^{79.1}$	8	[53]
TWINE-128	36	2^{60}	$2^{126.82}$	11	[53]
IDEA	8	2^{52}	$2^{126.2}$	1.5	[100]
HIGHT	32	2^{48}	$2^{126.4}$	8	[86]
Square	8	2^{48}	2^{126}	3	[129]
Prince	10	2^{40}	$2^{62.72}$	1	[6]

Table 3.2: Biclique attack on other block ciphers.

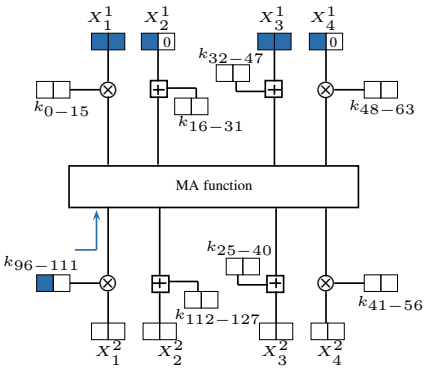
Bogdanov et al. in [39] reasoned that biclique attack on AES works because AES was not designed to be strongly resistant against attacks which work on smaller number of rounds - a fact which is true for other ciphers as well and this limitation was exploited by them to construct bicliques. It was also supported by the fact that difference diffusion in the key schedule was slow and diffusion rate decreased as key size increased, e.g., a 5-round biclique could be constructed in case of AES-256 as compared to a 3-round biclique in AES-128. These factors allowed biclique technique to recover the key with a complexity lesser than brute force.

Narrow Bicliques. In all the block ciphers mentioned in 3.2, biclique key recovery attack was applied exactly as discussed for AES except in the case of IDEA block cipher. For IDEA, Khovratovich et al. in [100] proposed a variation of biclique attack technique and termed it as *narrow biclique cryptanalysis*. They argued that IDEA was designed to achieve full diffusion in a single round. This posed a challenge as only one round biclique could be constructed and it led to data complexity being full codebook. The authors in [100] suggested constructing bicliques with independent differential (Δ_i and ∇_j) trails that occur with probability less than 1 to mitigate full diffusion.

With these probabilistic differentials, an attacker can enforce certain plaintext/ ciphertext bits to be always zero thus reducing the data complexity (as shown in Figs. 3.12(a), 3.12(b)). The attacker is thus restricting the diffusion of differential trail as per her requirements and hence the term - *narrow biclique*. She is successful in recovering the key by utilizing available degree of freedom in other parts of the cipher that balance out the probabilistic nature of the biclique so constructed.



(a) Full diffusion in 1-Round IDEA.



(b) Limited diffusion in 1-Round IDEA.

Figure 3.12: Biclique Attack on block cipher IDEA. Due to difference injected in key bits 96-111, all the plaintext bits are affected as shown in (a). In (b), the attacker is interested only in those differentials for which plaintext bits marked as '0' are not affected.

3.5 Improved biclique based key recovery attacks on AES

In the earlier sections, we saw successful application of biclique attacks to AES and many other prominent block ciphers. However, the original work on AES in [39] introducing biclique key recovery leaves several questions unanswered though, which are crucial to judge the real-world security of AES and implications of the biclique cryptanalysis in general:

- *Is there much potential in minimizing the data complexity* of the biclique attacks? In fact, it is low data complexity attacks that are most relevant in practice, especially in the context of efficient implementation of the attacks – the point clearly made in [38]. Actually, the data complexity of the original biclique attack makes any practical implementation of them highly unreasonable since the standard brute force is very likely to be both cheaper and faster in reality (mainly due to the high requirements in terms of storage or oracle access).

- Though the new technique has been coined after bicliques, the initial structures are explicitly limited to balanced bicliques only, i.e., complete bipartite graphs in which the two set of vertices have exactly the same cardinality. So the question remains: Can one take any advantage of using other types of bicliques as initial structures in AES?

- Finally, no comprehensive investigation of *attack optimality* in terms of computational complexity, data complexity or both has been performed. So it is still not clear if there are *faster biclique attacks*, even in the same class of the attacks as proposed in [39].

We aim to bridge these gaps and answer all the three questions in positive for all the variants of AES namely: AES-128, AES-192 and AES-256.

3.5.1 Our Contribution

1. As regards to more general initial structures, we drop the balancedness requirement for bicliques. We propose to use stars, which are the most unbalanced bicliques, having only one vertex in one of the two disjoint sets of biclique vertices. This allows us to come up with a star-based biclique key recovery technique for block ciphers that inherently has the minimal theoretically attainable data complexity - the one due to the unicity distance.

2. In terms of the attack space exploration for biclique cryptanalysis, we limit ourselves to the most promising class of attacks as applied to AES: Namely, we enumerate all truncated independent balanced bicliques and stars whose key modification trails have upto three active bytes in some state of the expanded key. For the sake of conciseness, we will refer to this class of attacks as based on *tight truncated independent bicliques and stars* (see Section 3.8). Clearly, this exploration does not cover many advanced and inherently harder-to-analyze attack vectors such as long-bicliques (bicliques of a lower dimension whose key modification differentials share active S-boxes) or narrow bicliques [116]. That is why, one cannot claim any formal bounds on the complexity of biclique attack complexity in general.

   Nonetheless, it is the tight truncated independent-biclique approach to key recovery that has resulted in the fastest attack on the full AES-128, AES-192 and

AES-256 so far and we believe that this investigation does provide important new insight into the limits of the current techniques of biclique cryptanalysis especially when applied to AES.

3. Using stars as initial structures, we propose the first key recovery attack faster than brute force on AES-128, AES-192 and AES-256 with the minimal theoretically possible data complexity. AES-128 requires 1 or 2 known plaintexts and has computational complexity $2^{126.67}$. Similar results are found for AES-192 ($2^{190.9}$) and AES-256 (2^{255}).

4. Next, we exhaustively enumerate all attacks based on tight truncated independent bicliques and stars for all AES variants which have a data complexity lower than the full codebook. It turns out that for AES-128, the ones of computational complexity $2^{126.16}$ are fastest. Interestingly, this exactly corresponds to the original key recovery on AES-128 [39].We further investigate the data complexity of these attacks for the bicliques of dimension $d = 8$ and show that the minimum data complexity is 2^{64} (cf. 2^{88} in the original attack). This implies that the original attack did not have the optimal data complexity. We find similar results for AES-192. The fastest attacks have a computational complexity of $2^{190.16}$. However, amongst these, the one having 2^{48} (cf. 2^{80} in the original attack) data complexity has the lowest data requirements.

5. Interestingly, for AES-256 we find that the fastest attacks having a data complexity lower than full codebook have a computational complexity of $2^{254.31}$. This turns out to be lower than $2^{254.42}$ reported in the original attack in [39], implying again that the original attack did not mention the minimal values. We also find some discrepancies in the complexity estimate of AES-256 in [39]. We show that the time complexity of the attack should be $2^{254.52}$ as against $2^{254.42}$ stated in [39].

6. To investigate the limits of this class of biclique cryptanalysis, we abandon all restrictions on the data complexity and search for the fastest attacks on AES in this class. We find that the ones with computational complexity of $2^{125.56}$, $2^{189.51}$ and $2^{253.87}$ for AES-128, AES-192 and AES-256 respectively are the fastest (though requiring the full code book). An interesting outcome of these constructions is that they utilize the longest biclique covered in the full AES attack so far. For AES-128, the longest biclique has length of 3 rounds whereas for AES-192 and AES-256 the longest bicliques cover 5 rounds each.

Using sieve-in-the-middle technique (discussed in Section 3.3.2), the above computational complexities can be reduced further. The cryptanalytic results of our work combined with and without sieve-in-the-middle (SIM) technique are summarized in Table 3.3.

Table 3.3: Key recovery with bicliques for full AES

Algorithm	data	computations without SIM	memory	computations with SIM	memory	success prob	biclique length (rounds)	property shown	reference
AES-128	2^{88} CC	$2^{126.16}$	2^8	-	-	1	2.5	-	[39]
	2^4 CP	$2^{126.89}$	2^8	-	-	1	2	-	[38]
	2^{88} CC	-	-	$2^{125.69\dagger}$	2^{64}	1	2.5	-	[48]
AES-128	Unic. dist: 2 KP	$2^{126.67}$	2^8	$2^{126.59}$	2^{64}	1	1	fastest with minimum data	§ 3.7.1
	2^{64} CC	$2^{126.16}$	2^8	$2^{125.98}$	2^{64}	1	2.5	fastest with $< 2^{128}$ data	§ 3.9.1
	2^{128}	$2^{125.56}$	2^8	$2^{125.35}$	2^{64}	1	3	fastest	§ 3.10.1
AES-192	2^{80} CC	$2^{190.16}$	2^8	-	-	1	3.5	-	[39] [5]
	2^{48} CC	$2^{190.28}$	2^8	-	-	1	3.5	-	[5]
AES-192	Unic. dist: 2 KP	$2^{190.9}$	2^8	$2^{190.83}$	2^{64}	1	1.5	fastest with minimum data	§ 3.7.2
	2^{48} CC	$2^{190.16}$	2^8	$2^{190.05}$	2^{64}	1	3.5	fastest with $< 2^{128}$ data	§ 3.9.2
	2^{128}	$2^{189.51}$	2^8	$2^{189.31}$	2^{64}	1	5	fastest	§ 3.10.2
AES-256	2^{40} CC	$2^{254.42\ddagger}$	2^8	-	-	1	3.5	-	[39]
	2^{64} CC	$2^{254.53}$	2^8	-	-	1	3.5	-	[5]
AES-256	Unic. dist: 3 KP	2^{255}	2^8	$2^{254.94}$	2^{64}	1	1.5	fastest with minimum data	§ 3.7.3
	2^{64} CC	$2^{254.31}$	2^8	$2^{254.24}$	2^{64}	1	3.5	fastest with $< 2^{128}$ data	§ 3.9.3
	2^{128}	$2^{253.87}$	2^8	$2^{253.82}$	2^{64}	1	5	fastest	§ 3.10.3

† Our analysis shown in the Section 3.3.2 estimates the cost as $2^{125.98}$.
‡ Our analysis shown in the Section 3.9.3 estimates the cost as $2^{254.52}$.

In Table 3.4, a summary of some of the other existing cryptanalytic results on AES in single key model (other than the biclique attacks that have already been listed in Table 3.1) is provided to ease the comparison with our results.

3.6 Stars

We now introduce *stars* - unbalanced bicliques which are trees with one node and many leaves - for the analysis of AES. We start with the observation that the biclique does not have to be balanced, i.e., contain 2^d states in each of its two vertex sets - to cover 2^{2d} keys. Indeed, there is a biclique with just one state in one vertex set and 2^{2d} states in the other one: $S_x = \{x\}$, $S_y = \{y_{i,j}\}$, $i, j \in \{0, 2^d - 1\}$, where each $y_{i,j}$ is obtained

Table 3.4: Summary of non-biclique cryptanalytic attacks on AES in the single key model.

Attack Type	Rounds	Data	Time	Memory	Reference
AES-128					
Square	6	2^{32}	2^{72}	2^{32}	[55]
Square	6	6×2^{32}	2^{44}	2^{32}	[75]
Square	7	$2^{127.997}$	2^{120}	2^{64}	[75]
Impossible Differential	7	$2^{115.5}$	2^{119}	2^{109}	[20]
Impossible Differential	7	$2^{112.2}$	$2^{117.2}$	$2^{112.2}$	[126]
Impossible Differential	7	$2^{106.2}$	$2^{110.2}$	$2^{90.2}$	[130]
Meet-in-the-Middle	7	2^{99}	2^{99}	2^{96}	[61]
AES-192					
Square	7	2^{32}	2^{182}	2^{32}	[127]
Square-collision	7	2^{32}	2^{140}	2^{84}	[79]
Square	7	$2^{128} - 2^{119}$	2^{120}	2^{64}	[75]
Impossible Differential	7	$2^{115.5}$	2^{119}	2^{45}	[179]
Impossible Differential	7	$2^{113.8}$	$2^{118.8}$	$2^{89.2}$	[126]
Meet-in-the-Middle	7	2^{99}	2^{99}	2^{96}	[61]
Meet-in-the-middle	8	2^{113}	2^{172}	2^{129}	[71]
Meet-in-the-middle	8	2^{107}	2^{172}	2^{96}	[61]
Meet-in-the-middle	9	2^{117}	$2^{186.5}$	$2^{177.5}$	[122]
AES-256					
Square-collision	7	2^{32}	2^{184}	2^{140}	[79]
Square	7	2^{36}	2^{172}	2^{32}	[75]
Impossible Differential	7	$2^{113.8}$	$2^{118.8}$	$2^{89.2}$	[126]
Meet-in-the-Middle	7	2^{99}	2^{98}	2^{96}	[61]
Meet-in-the-middle	8	2^{113}	2^{196}	2^{129}	[71]
Meet-in-the-middle	8	2^{107}	2^{196}	2^{96}	[61]
Meet-in-the-middle	9	2^{120}	2^{203}	2^{203}	[61]
Meet-in-the-middle	9	2^{121}	$2^{203.5}$	$2^{169.9}$	[122]
Meet-in-the-middle	10	2^{111}	2^{253}	$2^{211.2}$	[147]

by encrypting x with key $K[i,j]$, covering 2^{2d} keys. This biclique is called a *star* of dimension d (as shown in Fig. 3.14).

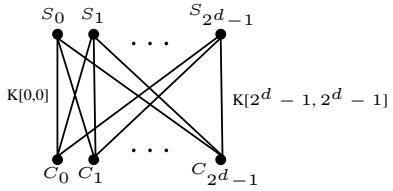


Figure 3.13: Balanced biclique of dimension d

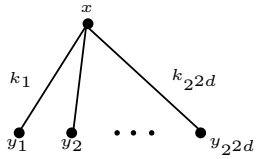


Figure 3.14: Star: maximally unbalanced biclique of dimension d for the minimum data complexity

If we place the star at the beginning of the cipher, and let x be the *plaintext* (or ciphertext) - the data complexity of the MITM part of the key recovery will be exactly 1. Note that x can be any value and, thus, we deal with a known-plaintext key recovery here. The overall data complexity is solely defined by the unicity distance of the cipher and, therefore, minimal theoretically attainable.

3.6.1 Stars from independent differentials

Similar to balanced bicliques, stars can be constructed efficiently from independent sets of differentials. Unlike balanced bicliques, however, the necessary form of differentials is different. Suppose we have a set of $2^d - 1$ distinct related-key Δ-differentials from x to $y_{i,j}$:

$$(0, \Delta_i^K) \longmapsto \Delta_i$$

and a set of $2^d - 1$ distinct related-key ∇-differentials from over the same part of the cipher:

$$(0, \nabla_j^K) \longmapsto \nabla_j.$$

We assume that the Δ-differentials and ∇-differentials do not share any active nonlinear components. If input x, output $y_{0,0}$ and key $K[0,0]$ conform to both Δ- and ∇ differentials, then the values:

$$\begin{aligned} & x, \\ y_{i,j} &= y_{0,0} \oplus \Delta_i \oplus \nabla_j, \text{ and} \\ K[i,j] &= K[0,0] \oplus \Delta_i^K \oplus \nabla_j^K \end{aligned}$$

form a star of dimension d, with $\Delta_0 = \nabla_0 = \Delta_0^K = \nabla_0^K = 0$.

3.7 Minimum data complexity key recovery for AES

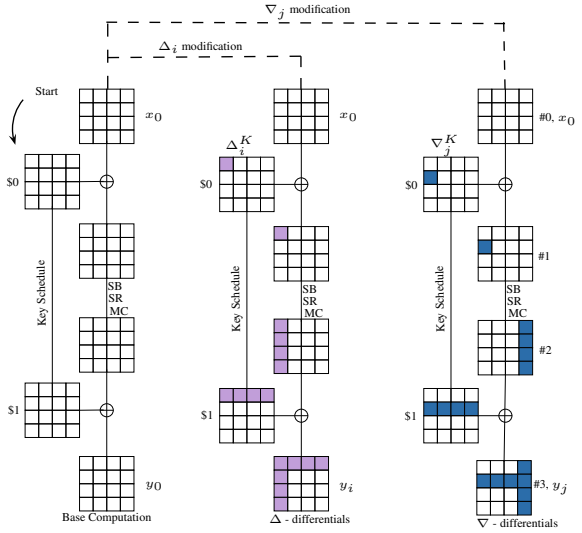
In this section, we demonstrate star-based independent-biclique key recoveries for full AES-128, AES-192 and AES-256. The star-based biclique can be placed either at the initial rounds of AES or at the last few rounds of the same. We tested the attack complexity for both locations through our C-program and only report the best values for all the three AES variants in the subsequent writeup.

3.7.1 AES-128

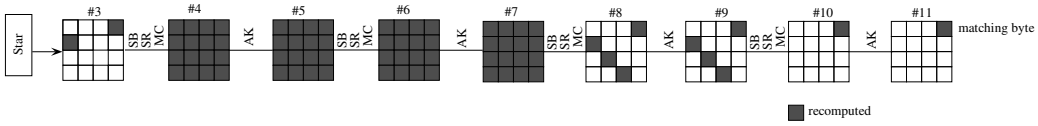
In AES-128, it is possible to construct a star of dimension 8 over the first round. The master key \$0, i.e., the first subkey is taken as the base key. The index i is placed in byte 0 whereas index j is placed in byte 1. The base keys are all 16-byte values with two bytes (i.e., bytes 0 and 1) fixed to 0 whereas the remaining 14-bytes take all possible values. Thus, the 128-bit key space is divided into 2^{112} groups with 2^{16} keys in each group. Δ-trail activates byte 0 of key \$0 and ∇-trail activates byte 1 of key \$0 (as shown in Fig. 3.15(a)). Difference propagation in these differentials over one round is non-overlapping till the end of round 1. In state #3, there is a linear overlap between those and, already in round 2, one has to recompute 2 S-boxes for each key (shown in Fig. 3.15(a)). Rather surprisingly, even if the length of the star is just one round, the form of its trails is such that this short biclique still allows the adversary to obtain a reasonable computational advantage over brute force.

In the forward direction of matching, starting in round 2, a part of the state has to be recomputed for each key. In round 2, only 2 S-boxes have to be recomputed. Starting in round 3 and forwards, the propagation affects the whole state (shown in Fig. 3.15(b)). In the backward direction of matching, one starts with the ciphertext obtained using the encryption oracle under the right key for plaintext x. The Δ- and ∇-propagations in the key schedule are such that only 5 bytes of the \$10 depend on both Δ and ∇. This means that only 5 S-boxes have to be recomputed in round 10. Starting in round 9 and backwards, the propagation affects the full state (as shown in Fig. 3.15(c)). We match on byte 12 in state #11 of round 5, in which only one S-box needs to be recomputed. In round 4 and round 6, only 4 S-boxes, respectively, are recomputed. The S-boxes in the four remaining rounds need to be recomputed completely (another 64 S-boxes). No S-box recomputations are needed in the key schedule.

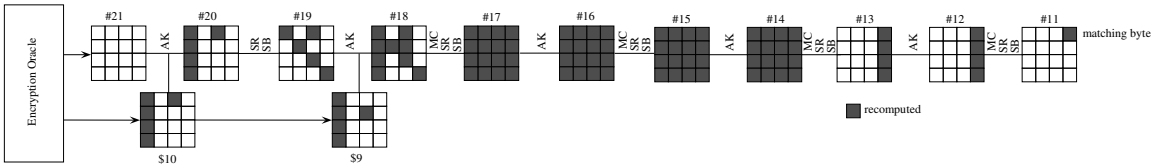
The whole process yields a recomputation of 80 out of 200 S-boxes. Thus, $C_{recomp} \approx 2^{14.67}$ in one key group. About 2^8 keys will be suggested in each key group after the meet-in-the-middle filtering, thus $C_{falsepos} = 2^8$. The complexity of precomputations and star generation is upper-bounded by $C_{precomput} \approx 2^{8.5}$ full AES computations. Thus, $C_{full} \approx 2^{126.67}$. The data complexity exactly corresponds to the unicity distance of AES-128 – the minimal data complexity theoretically attainable. One known plaintext-ciphertext pair can sometimes be enough (with success probability



(a) Star over first round



(b) Forward Recomputations



(c) Backward Recomputations

Figure 3.15: Fastest biclique attack on AES-128 with minimum data: time $2^{126.67}$ and data 1 or 2 ciphertexts.

of $1/e \approx 0.3679$). Two known plaintext-ciphertext pairs yield a success probability of practically 1. The memory complexity is upper bounded by 2^8 computations of sub-cipher involved in the precomputation stage, i.e., $\approx 2^{16}$ bytes.

Comparison with brute force attack. In star biclique attack, the cost of one round computations is saved by constructing a star biclique over that round. Since, we apply the *precomputations and recomputations* strategy in the MITM phase (as discussed in Section 3.3.2), the computational complexity in the 9 rounds is bit lower than the exhaustive key search complexity over these rounds. Hence, overall the complexity of a star attack will always be lower than exhaustive key search attack.

3.7.2 AES-192

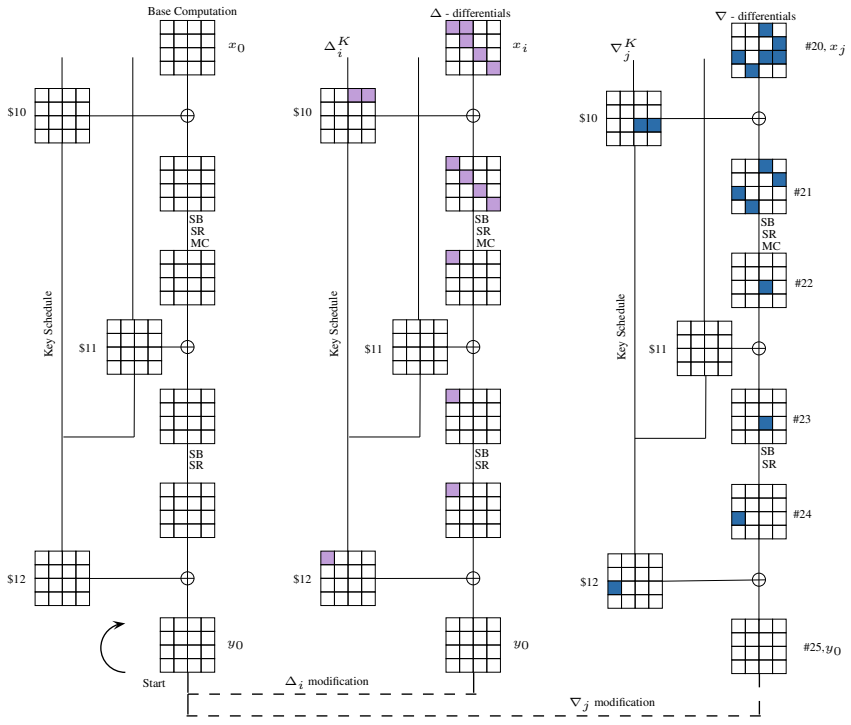
In AES-192, we construct a star of dimension 8 over the last 1.5 rounds (shown in Fig. 3.16(a)). Δ-trail activates byte 0 of sub-key \$12 and ∇-trail activates byte 2 of \$10 sub key. Non-overlapping trails cover rounds 11 and 12. We define the key groups with respect to the expanded key block K^7 which consists of two right columns of \$10 (further denoted by 10_R) and \$11 subkeys. The index i is placed in bytes 0 and 4 whereas index j is put in bytes 2 and 6. The base key in each group is chosen such that the key coverage is complete and there are no intersections between the key groups. The base keys are all 24 -byte values with two bytes (i.e., bytes 0 and 2) fixed to 0 whereas the remaining 22-bytes taking all possible values. This yields a partition of AES-192 key space into 2^{176} groups with 2^{16} keys in each.

In the matching phase we match on byte 12 in state #11. During the forward recomputations (in Fig. 3.16(b)), we start with plaintext obtained from ciphertext y using the decryption oracle under the right key. The Δ and ∇ propagations in the key schedule are such that there are no overlapping bytes in \$0 which depend on both Δ and ∇ trails. Hence, no recomputations are required in round 1, i.e., state #1. Starting from round 2, 16 + 16 + 16 + 4 = 52 S-boxes are required to be recomputed. Backward recomputations starting from round 10 require 16 + 16 + 16 + 4 + 1 = 53 S-boxes to be recomputed in Fig. 3.16(c). No recomputations in key schedule are required. Hence, a total of 105 out of 224 S-boxes are required for recomputation. Thus $C_{recomp} \approx 2^{14.82}$. The precomputations and star generation (based on S-box calculations) are upper bounded by $C_{precomp} \approx 2^{8.5}$ full AES computations. Thus $C_{full} \approx 2^{190.9}$. Two ciphertexts are required to carry out the attack with a success probability of 1.

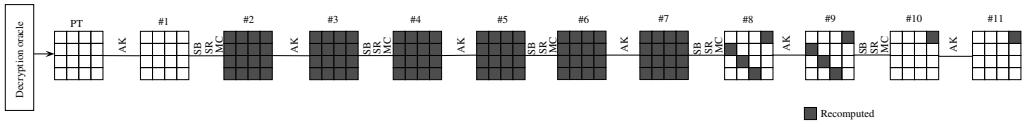
3.7.3 AES-256

In AES-256, a star of dimension 8 over the last 1.5 rounds (shown in Fig. 3.17(a)) is constructed. Δ-trail activates byte 0 of sub-key \$13 and ∇-trail activates byte 5 of \$13 sub key. Non-overlapping trails cover rounds 13 and 14. We define the key groups with respect to the expanded key block K^6 which consists of \$13 and \$14 subkeys. The index i is placed in byte 16 and index j is put in byte 21. The base keys are all 32 -byte values with two bytes (i.e., bytes 16 and 21) fixed to 0 whereas the remaining 30-bytes taking all possible values. This yields a partition of AES-256 key space into 2^{240} groups with 2^{16} keys in each.

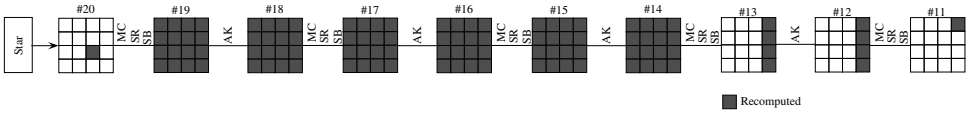
In the matching phase, we match on byte 1 of state #11. 52 S-boxes (16 + 16 + 14 + 4) in the forward direction (shown in Fig. 3.17(b)) and 85 S-boxes (1 + 4 + 16 + 16 + 16 + 16 + 16) in the backward direction (Fig. 3.17(c)) are recomputed. Together with 1 S-box recomputation in key schedule, a total of 138 out of 276 S-boxes are recomputed. Thus, $C_{full} \approx 2^{255}$. Two known plaintext-ciphertext pair can



(a) Star over last 1.5 rounds in AES-192



(b) Forward Recomputations



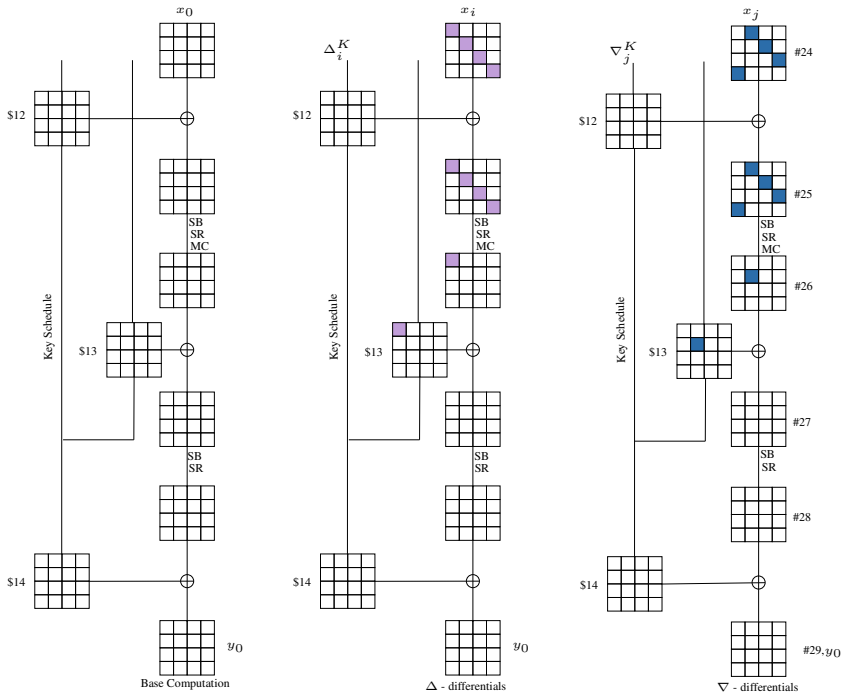
(c) Backward Recomputations

Figure 3.16: Fastest biclique attack on AES-192 with minimum data: time $2^{190.9}$ and data 2 ciphertexts.

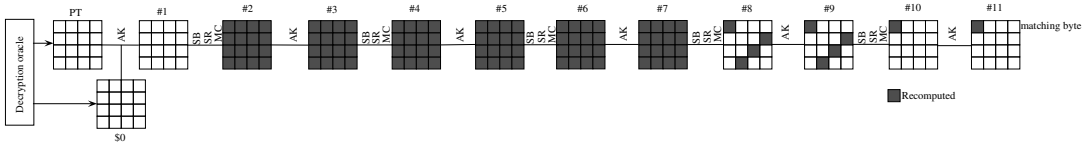
sometimes be enough (with success probability of $1/e \approx 0.3679$) whereas three known plaintext-ciphertext pairs yield a success probability of practically 1 with memory not more than 2^{16} bytes required.

3.8 A search technique for biclique attacks on AES

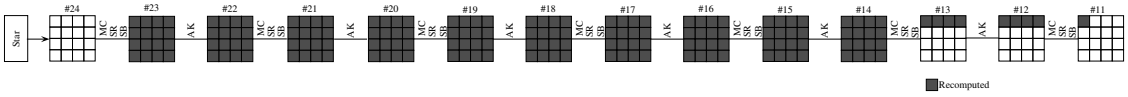
In this section, we describe how we enumerate all biclique key recoveries in a large promising class of biclique attacks.



(a) Star over last 1.5 rounds in AES-256



(b) Forward Recomputations



(c) Backward Recomputations

Figure 3.17: Fastest biclique attack on AES-256 with minimum data: time 2^{255} and data 2 or 3 ciphertexts.

3.8.1 Enumerating bicliques

Clearly, going over all possible initial structures, even without enumerating possibilities for the actual key recovery, would be infeasible for the AES. So we have to confine the search space of attacks by imposing some limitations. We now describe our search strategy along with some justifications for our choices.

- First, we consider bicliques (complete bipartite graphs) as initial structures. We stress that we include *both balanced bicliques and stars* in our search.

- Second, we restrict the search to *independent-bicliques only*. This constraint excludes such bicliques such as narrow-bicliques [116], which are especially challenging to enumerate. However, despite not being optimal in the number of rounds covered, it is the independent-bicliques that attain the highest advantages over brute force for full AES-128 and its variants so far.

- Third, we confine the search to independent related key-differentials that have a key state in their trails with exactly one or two [8] or three active bytes [9]. Note that these bytes do not have to be the bytes where the key difference is injected and the key difference can still be injected in multiple bytes. We also consider the special rules defined in [39] for AES-192 as mentioned below and apply it to other AES variants also.

  - We test differential trails in which double byte differences (i_1,i_2) are injected in Δ trail and single/double/triple byte differences are injected in ∇ trail. These (i_1,i_2) are all possible columns that have one zero byte after applying $MixColumns^{-1}$, e.g.,

$$\begin{pmatrix} 0 \\ i_1 \\ i_2 \\ 0 \end{pmatrix} = MixColumns^{-1} \begin{pmatrix} * \\ i \\ * \\ 0 \end{pmatrix} or \begin{pmatrix} i_1 \\ i_2 \\ 0 \\ 0 \end{pmatrix} = MixColumns^{-1} \begin{pmatrix} i \\ * \\ 0 \\ * \end{pmatrix}.$$

  - Similarly, we also test differential trails in which triple byte differences (i_1,i_2,,i_3) are injected in Δ trail such that all possible (i_1,i_2,i_3) have two zero bytes after applying $MixColumns^{-1}$, i.e.,

$$\begin{pmatrix} i_1 \\ i_2 \\ i_3 \\ 0 \end{pmatrix} = MixColumns^{-1} \begin{pmatrix} 0 \\ i \\ * \\ 0 \end{pmatrix} or \begin{pmatrix} 0 \\ i_1 \\ i_2 \\ i_3 \end{pmatrix} = MixColumns^{-1} \begin{pmatrix} 0 \\ 0 \\ i \\ * \end{pmatrix}.$$

- Finally, to keep the search space from exploding, we consider the trails of the bicliques in a *truncated* manner: We do not differentiate between the active values of the key modification trails in our bicliques (values of differences in the related-key differentials). In particular, it means that once activated, a difference in a byte of a trail cannot be canceled out. This is a significant but necessary limitation since we believe it is infeasible to run the exhaustive search otherwise for excessively high computational complexities.

We implemented these restrictions in a C program and were able to successfully enumerate all the *tight truncated independent balanced bicliques and stars* of AES-128, AES-192 and AES-256.

[8] Such trails do not collapse into a single active byte in any of the key states.

[9] Such trails do not collapse into a single active byte or two active bytes in any of the key states.

3.8.2 Searching for key recoveries

Having enumerated all the bicliques as described above exhaustively, we apply meet-in-the-middle (MITM) technique to each of the initial structures obtained to evaluate their time and data complexities. This is done as follows. First of all, we set the optimization goal as *minimizing the time complexity for a given data complexity restriction*. That is, in each search for a key recovery, we fix an upper bound on the data complexity. Then we perform the exhaustive search over all possibilities for matching. In terms of key enumeration, we impose the restriction that the forward and backward key modifications should have at least one state of linear intersection. This enables full key space coverage and success probability of 1. The MITM technique used includes partial matching (the matching is performed on a byte of the state to save computations) and the cut-and-splice technique (so that trails can go over the encryption/decryption oracles to win degrees of freedom).

To evaluate the time complexity of a key recovery attack efficiently on-the-fly, the computational model proposed in [39] is used: All linear operations (AddRoundKey, ShiftRows, and MixColumns) are ignored and one counts only the number of required S-box computations. As an example, one full AES-128 evaluation requires 200 S-box computations – a metric that proved to be meaningful in practice [38]. Similarly, one complete evaluation of AES-192 and AES-256 corresponds to 224 and 276 S-boxes respectively. The time complexity is measured as the number of S-box computations that have to be performed per key tested. Again, this is the parameter that has lead to the fastest attacks so far since it makes the key group larger and minimizes the impact of biclique construction on the total complexity. Depending on the data complexity restriction, the program can find the optimal attack, i.e., the attack with the lowest measured time complexity under the data complexity restriction.

As a second optimization goal, we focus on *minimizing the data complexity for a given time complexity*. This second optimization is applied once the lowest computational complexity for recovering the key has been found in the previous step. At this point, we already know that there are no faster key recoveries in our search space. So we check if the data complexity of the fastest attack identified can be reduced. For this task, we fix the computational complexity to the value that we obtained in the previous step, and then among all the bicliques having that computational complexity, we search for the one that has the lowest data complexity. This task typically requires much less computations.

3.8.3 Applications to find attacks with minimal data and time complexities

We implemented our program to search for three data complexity restrictions:

- *Minimum data complexity:* The minimum data complexity attacks for AES-

128/192/256 were discovered using this program by setting the upper bounds of the data complexity to its theoretical minimum of the unicity distance. So we can claim that this is the fastest biclique key recovery with the minimal data complexity of exactly the unicity distance in the class of bicliques covered by our program.

- *Data complexity strictly lower than the full codebook:* This restriction is a standard line that is informally drawn between interesting attacks – that require less than the full codebook of texts - and less interesting attacks – that can only work with the full codebook. It is found that the fastest biclique key recoveries in the covered class with these restrictions have lower computational complexities (for AES-256) and lower data complexities (for AES-128 and AES-192) as compared to the original attack.

- *No data complexity constraint:* The program finds the fastest biclique key recovery in the entire class of biclique attacks covered when there is no restriction on the amount of data required. This attack provides an important insight into the limits of the independent-biclique approach developed so far.

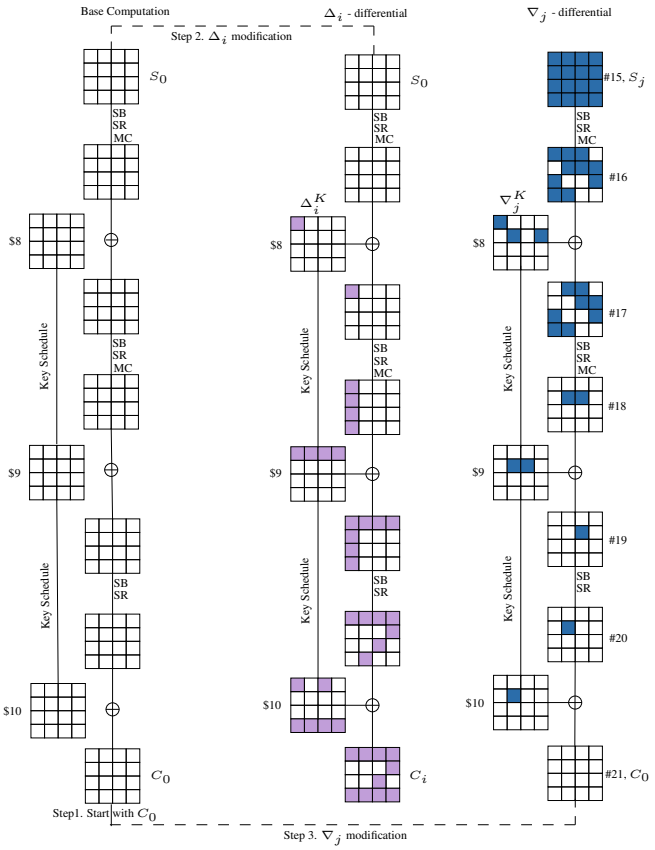
The fastest key recoveries corresponding to minimum data complexity for AES-128, AES-192 and AES-256 are already discussed in Section 3.7. Rest of the above mentioned categories are analyzed for all AES variants and their details are covered in the subsequent sections.

3.9 Fastest biclique key recovery with less than the full codebook of data

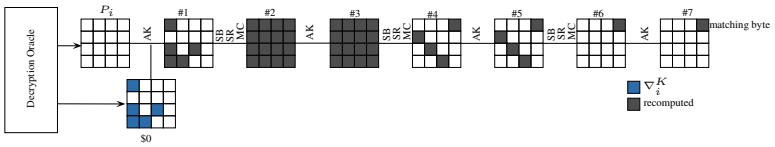
In this section, we demonstrate the biclique key recoveries with optimal time complexity for the full AES-128, AES-192 and AES-256.

3.9.1 AES-128

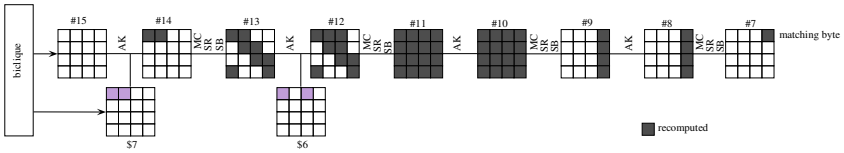
This attack is based on a balanced biclique of dimension 8 over the last 2.5 rounds of AES-128 (shown in Fig. 3.18(a)). The forward and backwards trails in the biclique have an intersection in byte 0 of \$8. However, this intersection is in a linear operation (xor) of the key schedule and does not affect the biclique property. The key is enumerated in \$9 which is the only key state that is linear in the key modification, both in forward and backward trails. The bytes of key enumeration with i and j differences are non-intersecting. The index i is placed in bytes 0,4,8, and 12 while index j is put in bytes 5 and 9. The 2^{112} base keys are all 16-byte values with two bytes (i.e., bytes 0 and 5) set to 0 whereas the remaining 14-bytes taking all possible values. This yields 2^{16} keys in each group.



(a) Biclique over last 2.5 rounds



(b) Forward Recomputations



(c) Backward Recomputations

Figure 3.18: Fastest biclique attack on AES-128 with less than full codebook: time $2^{126.16}$ and data 2^{64}.

For key recovery, in the MITM stage, partial matching is done in byte 12 of data state #7 of round 4, where only one S-box needs recomputation. In round 3 and round 5, only 4 S-boxes, respectively and in round 7, only 8 S-boxes are recomputed. In

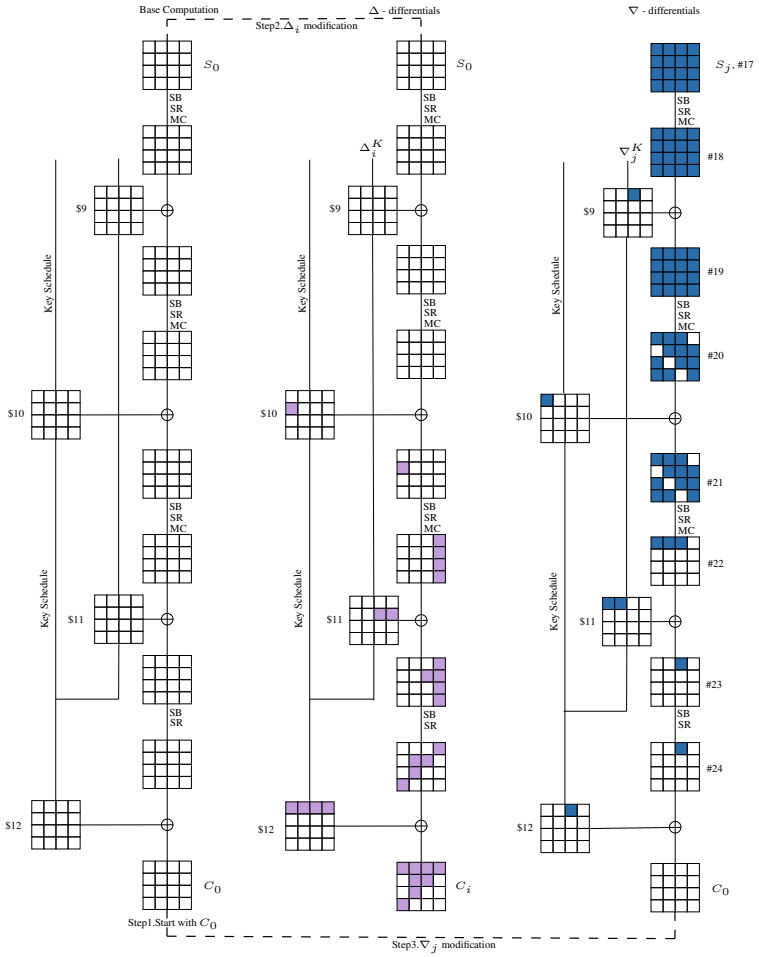
round 1, 5 S-boxes are recomputed as the plaintext is influenced by 5 active bytes of the backward key modification through the key schedule. The S-boxes of rounds 2 and 6 have to be recomputed completely (as shown in Figs. 3.18(b) and 3.18(c)). In total, also counting the necessary recomputations in the key schedule, we arrive at 55 S-boxes that have to be recomputed for each key, resulting in $C_{recomp} \approx 2^{14.14}$. As in the previous attacks, $C_{falsepos} \approx 2^8$ and $C_{precomp} \approx 2^{8.5}$. This yields $C_{full} \approx 2^{126.16}$. Furthermore, since $\Delta_i^K(\$10_3) = \Delta_i^K(\$10_{11}) = \Delta_i^K(\$10_{15})$, the ciphertext bytes C_3, C_{11} and C_{15} are always equal. Hence, the data complexity is 2^{64} chosen ciphertexts. This is lower than (2^{88}) obtained in the original attack [39]. As in all our attacks, the success probability is 1 and memory complexity is $2^8 \times 16 \times 7 = 2^{15}$ bytes.

3.9.2 AES-192

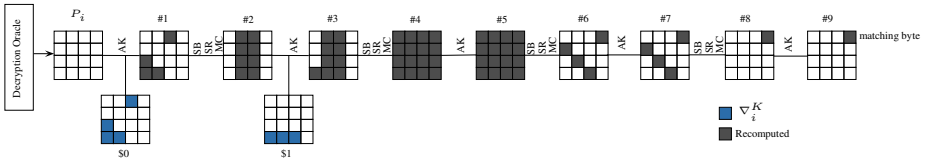
This attack is based on a balanced biclique of dimension 8 over the last 3.5 rounds of AES-192 (Fig. 3.19(a)). Δ-trail activates byte 1 of sub-key \$10 and ∇-trail activates byte 8 of \$12 sub-key. Non-overlapping trails cover rounds 9 to round 12. We define the key groups with respect to the expanded key block K^7 which consists of two right columns of \$10 (further denoted by 10_R) and \$11 subkeys. The index i is placed in bytes 17 and 21 whereas index j is put in bytes 8 and 12. The base keys are all 24-byte values with two bytes (i.e., bytes 8 and 17) fixed to 0 whereas the remaining 22-bytes taking all possible values. This allows a partition of AES-192 key space into 2^{176} groups with 2^{16} keys in each. In the matching phase, 33 S-boxes in the forward direction (Fig. 3.19(b)) and 29 S-boxes in the backward direction (shown in Fig. 3.19(c)) are recomputed yielding a total of 62 out of 224 S-box recomputations. Thus, $C_{full} \approx 2^{190.16}$. The success probability of the attack is 1. Since $\Delta_i^K(\$12_0) = \Delta_i^K(\$12_4) = \Delta_i^K(\$12_8)$, the ciphertext bytes C_0, C_4 and C_8 are always equal. Hence, the data complexity is 2^{48} chosen ciphertexts. The data complexity of this attack is lower than 2^{80} obtained in the original attack [39].

3.9.3 AES-256

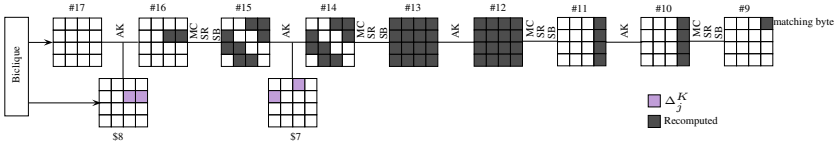
Through our automated program we detected certain discrepancies in the cost calculation in [39]. According to our calculations of the same, the computational complexity should be $2^{254.52}$ (c.f. $2^{254.42}$ in the original attack). The details of the same are as follows. Firstly, in Fig. 12 in [39], \$0 and \$1 subkeys have been marked as \$1 and \$2 subkeys respectively. Secondly, the required S-box calculation is given as 5.4375 Sub-Bytes operations which is 87 S-boxes operations in [39] whereas it should be 6.3125 Sub-Bytes operation (101 S-boxes) i.e., $2^{14.5}$ runs of full AES-256 (as shown in Fig. 3.20). As a result, the full computational complexity should be $2^{240} \times 2^{14.52} = \mathbf{2^{254.52}}$. The data complexity does not exceed 2^{40} queries.



(a) Biclique over last 3.5 rounds



(b) Forward Recomputations



(c) Backward Recomputations

Figure 3.19: Fastest biclique attack on AES-192 with less than full codebook: time $2^{190.16}$ and data 2^{48}.

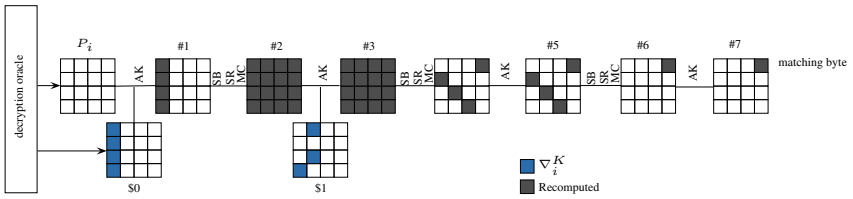


Figure 3.20: Corrected AES-256 forward computation.

For the fastest biclique key recovery requiring less than full codebook of data, we could construct a balanced biclique of dimension 8 over the last 3.5 rounds (shown in Fig. 3.21(a)) with lesser number of S-boxes that need to be recomputed. Δ-trail activates byte 13 of sub-key \$10 and ∇-trail activates bytes 0 and 4 of \$13 sub key. Non-overlapping trails cover rounds 11 to 14. Key groups are defined with respect to the expanded key block K^6 which consists of subkeys \$10 and \$11. The index i is placed in byte 13 whereas index j is put in bytes 16 and 24.The base keys are all 32-byte values with two bytes (i.e., bytes 13 and 24) fixed to 0 whereas the remaining 30-bytes taking all possible values. This allows a partition of AES-256 key space into 2^{240} groups with 2^{16} keys in each. In the matching phase, forward recomputations require 13 S-boxes (shown in Fig. 3.21(b)) and backward recomputations require 73 S-boxes (shown in Fig. 3.21(c)). No recomputations in key schedule are required. Hence, matching yields a recomputation of total 86 out of 276 S-boxes. Thus C_{full} is $\approx 2^{254.31}$. The data complexity as defined by the form of the biclique is 2^{64} with memory complexity being upper bounded by $2^8 \times 10 \times 16$ bytes $\approx 2^{16}$ bytes.

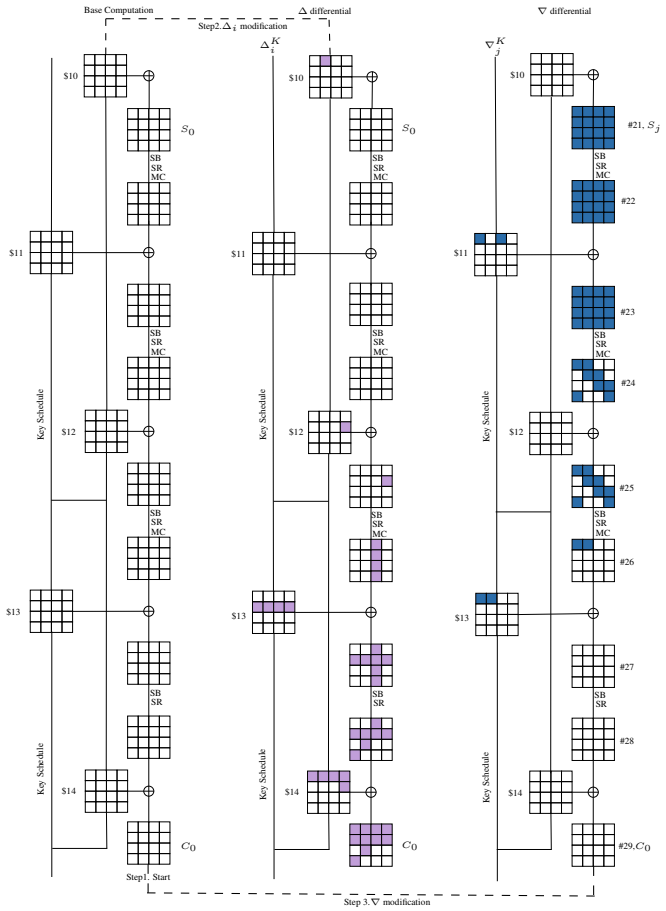
3.10 Fastest biclique key recovery in AES with no restriction on data complexity

In this section, we demonstrate the fastest biclique key recovery attacks on the full AES-128, AES-192 and AES-256.

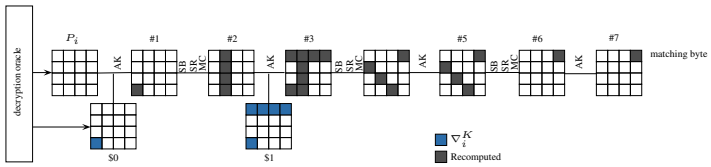
3.10.1 AES-128

When we drop the constraint of data complexity being below the full codebook, we can construct a balanced biclique of dimension 8 over 3 full AES-128 rounds and with the minimal recomputation of just one S-box in the fourth round, immediately after the biclique. The biclique is placed in rounds 2-4 which implies the data complexity of 2^{128} for the backward trail (as shown in Fig. 3.22(a)). In the forward recomputation, 12 S-boxes are recomputed (as shown in Fig 3.22(b)) whereas in the backward direction, 25 S-boxes are recomputed (shown in Fig. 3.22(c)) yielding a total of 37 S-box recomputations. Thus, $C_{full} \approx 2^{125.56}$. The data complexity in this attack is the

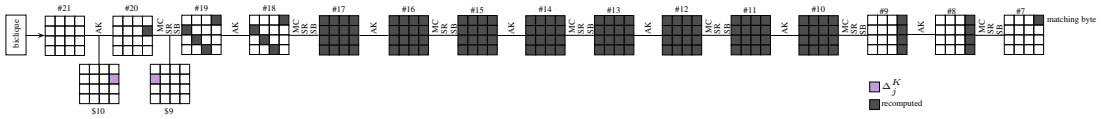
full codebook. The success probability is again 1 since key coverage is complete. The memory complexity stands at 2^8 memory blocks for precomputation stage.



(a) Biclique over last 3.5 rounds

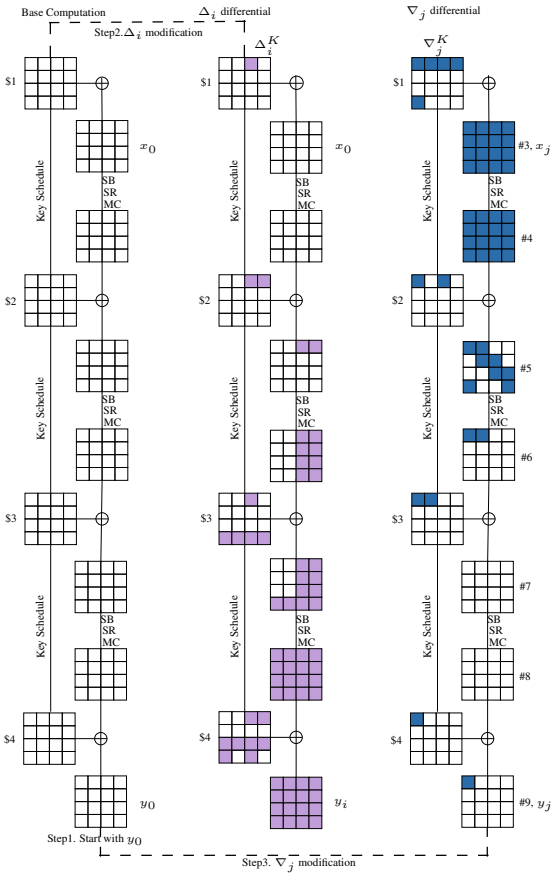


(b) Forward Recomputations

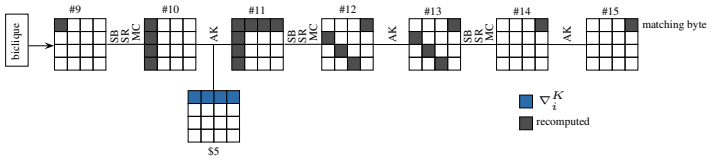


(c) Backward Recomputations

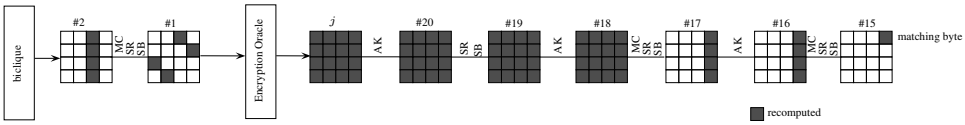
Figure 3.21: Fastest biclique attack on AES-256 with less than full codebook: time $2^{254.31}$ and data 2^{64}.



(a) Biclique over 3 rounds in the middle



(b) Forward Recomputations



(c) Backward Recomputations

Figure 3.22: Fastest biclique attack on AES-128: time $2^{125.56}$ and data 2^{128}. Here v represents the matching byte.

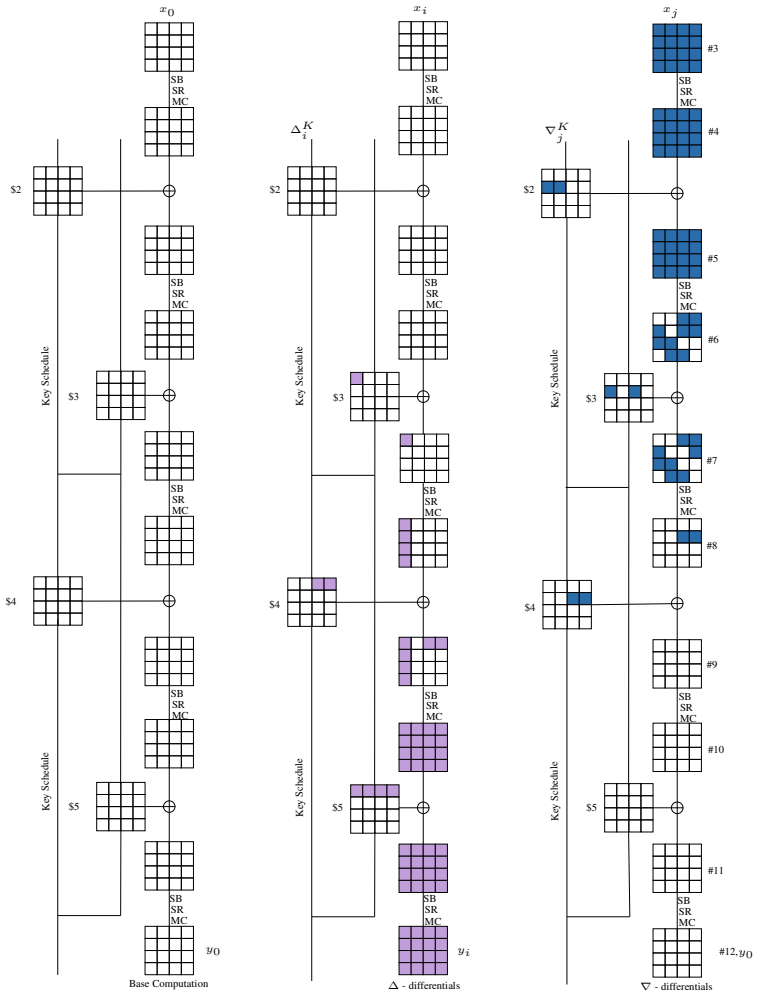
Preimage Attack on compression function. This key recovery can be converted into a preimage search for the compression function constituted by AES-128 in Davies-Meyer mode. Here, the attack works offline and does not have to make any online queries. This preimage attack requires $2^{125.56}$ AES-128 operations and finds a preimage with probability about 0.632. The generic preimage search would require 2^{128} time to succeed with probability 0.632.

3.10.2 AES-192

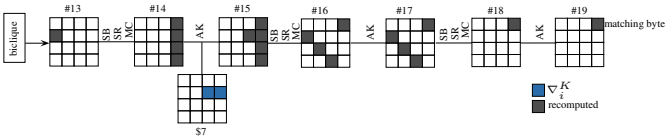
For AES-192, we could construct a balanced biclique of dimension 8 over 5 full rounds. The biclique is placed in rounds 2-6 shown in Fig. 3.23(a). Δ-trail activates byte 0 of subkey \$3 and ∇-trail activates byte 1 of \$6 subkey. In the matching phase, 10 S-boxes in the forward direction (Fig. 3.23(c)), 29 S-boxes in the backward direction (Fig. 3.23(b)) and 1 S-box in the key schedule are recomputed leading to a total of 40 out of 224 S-box recomputations. Hence $C_{full} \approx 2^{189.51}$ with data complexity being 2^{128} and memory complexity of 2^8. This key recovery when converted into a preimage search for the compression function constituted by AES-192 in Davies-Meyer mode requires $2^{125.51}$ AES-192 operations and finds a preimage with probability about 0.632.

3.10.3 AES-256

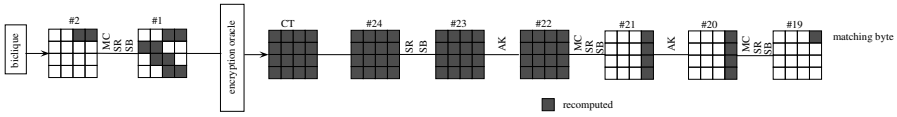
For AES-256, we could construct a balanced biclique of dimension 8 over 5 full rounds. The biclique is placed in rounds 2-6 (shown in Fig. 3.24(a)). Δ-trail activates byte 8 of sub-key \$1 and ∇-trail activates byte 0 of \$6 sub key. In the matching phase, 25 S-boxes in the forward direction (shown in Fig. 3.24(c)) and 41 S-boxes in the backward direction (shown in Fig. 3.24(b)) are recomputed leading to a total of 66 out of 276 S-box recomputations. Hence $C_{full} \approx 2^{253.87}$ with data complexity being 2^{128}. This key recovery when converted into a preimage search for the compression function constituted by AES-256 in Davies-Meyer mode requires $2^{125.93}$ AES-256 operations and finds a preimage with probability about 0.632.



(a) 5 round Biclique for AES-192

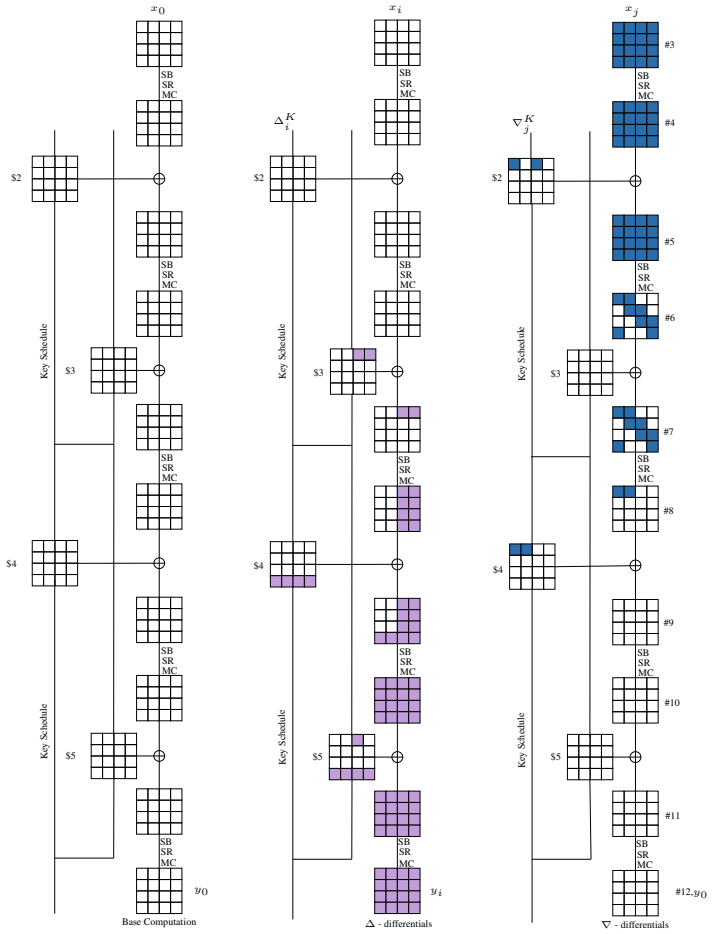


(b) Forward Recomputations

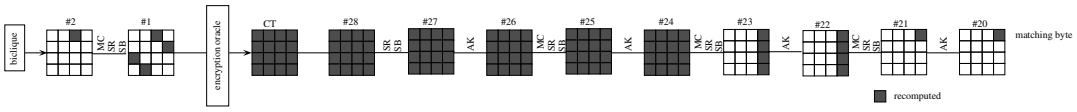


(c) Backward Recomputations

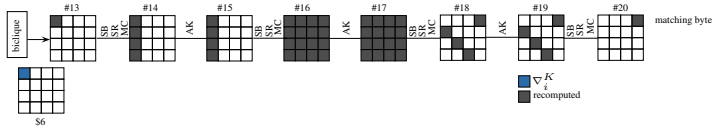
Figure 3.23: Fastest biclique attack on AES-192: time $2^{189.51}$ and full codebook.



(a) Biclique over 5 rounds in the middle



(b) Backwards recomputation



(c) Forwards recomputation

Figure 3.24: Fastest biclique attack on AES-256: time $2^{253.87}$ and full codebook.

3.11 Biclique based key recovery attacks on AES-128 with various data complexities

In the previous sections, we analyzed the biclique attacks for all the three AES variants under the following three categories:

1. Attack which has the lowest data complexity (in Section 3.7)
2. Attack which is optimum with respect to time as well as data complexity (in Section 3.9)
3. Attack which has the lowest time complexity with no restriction on the data complexity (in Section 3.10)

From the results obtained in the previous categories, it appears that for biclique based key recovery attacks, [10] data complexity is inversely proportional to the time complexity, i.e., as data complexity decreases, computational complexity increases. To verify this observation, we tried to analyze other biclique attacks for AES-128 where the data complexity lies between category 1 and category 3. The results of this analysis are shown in Table 3.5. We obtained these results by setting the optimization goal as: minimizing the time complexity for a given data complexity restriction in our C program.

As can be seen in table 3.5, in some cases, the attacks with a higher data complexity have higher time complexity as well, e.g., attack mentioned at row 5. This happened because in our tool, for each attack having a data complexity of 2^p, our program found an attack in which the Δ_i and ∇_j trails produced exactly p active bytes in the ciphertext.

However, in real-world, an attacker can always opt for the attack just preceding it which has a lower time as well as data requirements. For example, an attacker who has the ability to query up to 2^{32} ciphertexts (in row 5) may generate only 2^{24} queries and launch the attack having a time complexity of $2^{126.48}$. Thus, overall it can be said that the time complexity indeed decreases as data complexity increases. Moreover, these results can also prove useful in the attack scenarios where an attacker does not aim for the best attack but is limited by the environment she is in and the resources she can possess. Similar analysis can be done for AES-192 and AES-256 as well.

3.12 Summary

In this chapter, we first discuss the biclique cryptanalysis technique in general and then its application to AES and other block ciphers for recovering the secret key. We then

[10] involving bicliques of dimension 8

Table 3.5: Summary of biclique key recovery attacks on AES-128 with different data complexities obtained from our tool.

S. No.	Biclique Length (rounds)	Time Complexity	Data Complexity	S-box recomputations
1.	1	$2^{126.67}$	2	80
2.	1.5	$2^{126.56}$	2^{8}	74
3.	1.5	$2^{126.56}$	2^{16}	74
4.	1.5	$2^{126.48}$	2^{24}	70
5.	2.5	$2^{126.54}$	2^{32}	73
6.	2.5	$2^{126.31}$	2^{40}	62
7.	2.5	$2^{126.21}$	2^{48}	58
8.	2.5	$2^{126.37}$	2^{56}	65
9.	2.5	$2^{126.16}$	2^{64}	54
10.	2.5	$2^{126.72}$	2^{72}	82
11.	2.5	$2^{126.28}$	2^{80}	61
12.	2.5	$2^{126.16}$	2^{88}	54
13.	2.5	$2^{126.28}$	2^{96}	61
14.	2.5	$2^{126.26}$	2^{104}	60
15.	3	$2^{125.56}$	2^{128}	37

discuss some problems existing with the current biclique based key recovery attacks on AES. We then explore the space of independent bicliques as applied to key recovery for the full AES-128, AES-192 and AES-256. We put some reasonable restrictions on the bicliques to make the search feasible. The class of bicliques analyzed by a tool developed by us looks most promising in terms of cryptanalysis so far. In fact, the best key recoveries known so far for the full AES-128, AES- 192 and AES-256 belong to this class. Moreover, we utilize star structure (maximally unbalanced bicliques) to reduce the data complexity to the theoretically attainable minimum. We further note that the structure of the biclique is more important for the data complexity of the attack whereas the length of the biclique appears to be correlated with the computational complexity. We also propose biclique attacks which are fastest when there is no restriction on data complexity. We demonstrate that these attacks are the fastest among all independent-biclique attacks we study and might be considered as an indication of the limits beyond the current approaches to AES key recovery using bicliques.

Chapter 4

Biclique Cryptanalysis of AES-128 based Hashing Modes

Biclique Cryptanalysis, as discussed in the previous chapter, was first proposed by Khovratovich et al. in [103] for finding preimages for hash functions Skein [136] and SHA-2 [73]. This attack technique has not only led to the current best cryptanalytic results for AES [37,39] but also for SHA-2 [103] in terms of number of rounds attacked. In this chapter, we analyze the hash function settings and review the application of biclique cryptanalysis for finding preimages. In Chapter 2, we studied meet-in-the-middle attacks and discussed their limited applicability on block ciphers to recover the secret key. Contrary to block ciphers, MITM attacks have found more favor in the hash function domain. This is due to the fact that unlike block cipher cryptanalysis, the attacker can fully control the inner behaviour of the underlying compression function in hash function settings. Existing hash function designs usually involve step updations and message injections at round level. This gives an attacker an opportunity to efficiently utilize the available degrees of freedom on the input and find rounds which are independent of some message bits to apply MITM attack. Sasaki and Aoki [17, 152] exploited this fact to show some pioneering work in preimage attacks against MD4, MD5, SHA-0/1/2 etc. They proposed several new concepts such as *splice-and-cut framework*, *partial matching*, *partial fixing* etc. in the MITM framework that could be applied to hash functions as well as block ciphers [41, 90]. One such idea - the *initial structure* [152] forms the basis of biclique attack. In this chapter, we discuss this concept and its applicability in biclique cryptanalysis for finding preimages. We then present some second preimage attack results on AES-128 based hash functions.

The roadmap for this chapter is as follows: We first review some historical background pertaining to the origin of biclique cryptanalysis for hash functions in Section 4.1. We specifically discuss the concept of *initial structure* in this section. We then describe the generic technique for finding preimages in hash functions using biclique attack in Section 4.2. As an example, we also give a high level overview of the preimage attack on SHA-2 using biclique attack here. In Section 4.3, we discuss

the biclique attack on AES-128 based compression function for finding preimages and highlight some of the challenges involved in converting these attacks to preimage attack on AES-128 based hash functions. In Section 4.4, we describe the notations necessary to understand our work followed by Section 4.5, where we present the preimage attack on AES-128 based compression function using biclique cryptanalysis. We then demonstrate our biclique based second preimage attacks on AES-128 instantiated hash functions in Davies-Meyer (DM), Matyas-Meyer-Oseas (MMO) and Miyaguchi-Preneel (MP) modes respectively. These attacks are presented in Section 4.6 and work with a fixed IV which is not in the control of the attacker, mimicking the real life use case of these modes. The attacks in this section are demonstrated on 2-block message. Section 4.7 extends our attack to hash functions with message length ≥ 3 message blocks with same attack complexity as earlier. In the concluding section, we summarize our chapter. The original contribution of this thesis is from Section 4.3 to Section 4.7.

4.1 Origin of Biclique Cryptanalysis

Biclique cryptanalysis as stated in [39] originates from the splice-and-cut framework in hash function cryptanalysis and, more specifically from its element called the initial structure. The construction of *initial structure* is very design specific and hard to generalize. To facilitate understanding of this concept, we describe the construction of initial structure on MD5 hash function [143].

4.1.1 Short Description of MD5

The MD5 compression function (as shown in Fig. 4.1) follows Merkle-Damgard construction principle and takes 512-bit message block and 128-bit chain value as inputs and produces 128-bit output value. It consists of 4 rounds; each round consisting of 16 steps, i.e., 64 steps in total. The intermediate state consists of four registers (A_i, B_i, C_i, D_i) where, size of each register is 32-bits. Considering, IV (initial chaining value) = (IV_0, IV_1, IV_2, IV_3) and h (final hash output) = (h_0, h_1, h_2, h_3), the state update at each step (as shown in Fig. 4.1) is as follows:

$$\begin{aligned}
(A_0, B_0, C_0, D_0) &= (IV_0, IV_1, IV_2, IV_3) \\
A_{i+1} &= D_i \\
B_{i+1} &= (A_i + B_i + F_i(B_i, C_i, D_i) + K_i + M_{\pi(i)}) \lll s_i \\
C_{i+1} &= B_i \\
D_{i+1} &= C_i \qquad (1 \leq i \leq 64) \\
(h_0, h_1, h_2, h_3) &= (A_{64} \oplus A_0, B_{64} \oplus B_0, C_{64} \oplus C_0, D_{64} \oplus D_0)
\end{aligned}$$

Here, + denotes addition modulo 2^{32}, F_i and K_i are pre-defined step dependent function and constant respectively, $M_{\pi(i)}$ is the 32-bit message block to be injected at step i and $\lll s_i$ denotes the left rotation by s_j bits. For further details on MD5 construction, one can refer to [143]. As seen in Fig. 4.1, message word is injected for updating register B at each of the 64 steps. In this figure, the intermediate state at step i is denoted by $p_i = (A_i, B_i, C_i, D_i)$.

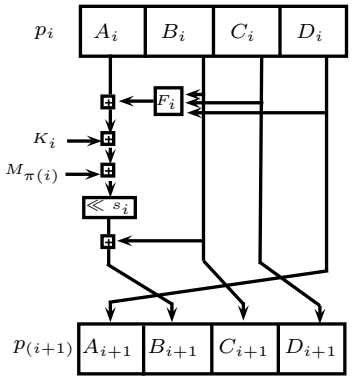


Figure 4.1: MD5 compression function.

Consider step-reduced MD5 comprising of first 29 steps. In Table 4.1, we show the message words used in each step of MD5.

Step i	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
$M_{(\pi)i}$	0	1	**2**	3	4	5	6	7	8	9	10	11	12	**13**	14	15
Step i	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32
$M_{(\pi)i}$	1	6	11	0	5	10	15	4	9	14	3	8	**13**	2	7	12
Step i	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47	48
$M_{(\pi)i}$	5	8	11	14	1	4	7	10	13	0	3	6	9	12	5	2
Step i	49	50	51	52	53	54	55	56	57	58	59	60	61	62	63	64
$M_{(\pi)i}$	0	7	14	5	12	3	10	1	8	15	6	13	4	11	2	9

Table 4.1: Message Schedule of MD5. Message block M = $m_0 \mid\mid m_1 \mid\mid m_2 \ldots m_{15}$ where each $\mid m_i \mid$ = 32-bits.

Now let us split the 29-step reduced MD5 into two chunks.

1^{st} *Chunk* : Step 1 - Step 13
2^{nd} *Chunk* : Step 14 - Step 29

It can be seen that 1^{st} Chunk is independent of message word m_{13}. Hence, if bits of m_{13} are flipped, it will not affect the functioning of 1^{st} Chunk. Similar property holds

true for message word m_2 in 2^{nd} Chunk . Such words are called *neutral words* [17] i.e., m_{13} is a neutral word for the 1^{st} Chunk and m_2 is a neutral word for the 2^{nd} Chunk .

For preimage attack, let the intermediate state p_{14} be the matching state. We first choose random values for m_i (i $\notin$ 2,13). Then, for all values of m_2, we calculate p_{14} in the forward direction and store the values in a table as (m_2, $\overrightarrow{p_{14}}$). After that for each value of m_{13}, we calculate p_{14} in the backward direction and check whether the computed $\overleftarrow{p_{14}}$ exists in the stored table or not (shown in Fig. 4.2). If we get a match, the corresponding message becomes the preimage.

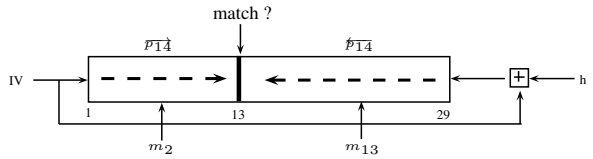


Figure 4.2: Overview of 29-steps attack on MD5

Let us have a look at another example. Consider MD5 from step 3 to step 61 (59 steps). Let,

1^{st} *Chunk*: Step 22 - Step 3 and Step 61 - Step 48, Neutral Word: m_{15}
Step 3 and Step 61 are considered consecutive steps as $p_{61} = h$ - p_3 (Splice-and-Cut).

2^{nd} *Chunk*: Step 23 - Step 44, Neutral Word: m_2

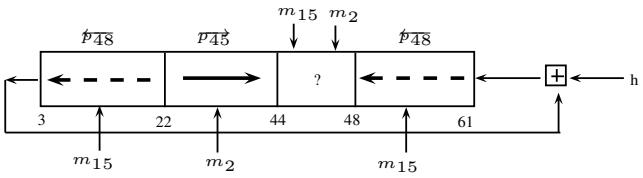


Figure 4.3: Overview of 59-steps attack on MD5

It can be seen in Fig. 4.3 that m_{15} comes before m_2 in steps 45-47 and hence involves computation of both the neutral words. Therefore, there does not exist any intermediate state in steps 45-47 that can be used as a matching state.

Sasaki et al. then suggested *partial matching* [17] instead of full 128-bit matching. They showed that if only one word (32-bit) is matched, computation of some steps can be skipped allowing more rounds to be covered. For example, in the above case,

through 1^{st} chunk we calculate $\overleftarrow{p_{48}}$ in the backward direction and in the forward direction through 2^{nd} Chunk, we compute $\overrightarrow{p_{45}}$. Now,

$\overleftarrow{p_{48}} = (A_{48}, B_{48}, C_{48}, D_{48}) = (\mathbf{B_{45}}, B_{48}, B_{47}, B_{46})$ and,
$\overrightarrow{p_{45}} = (A_{45}, B_{45}, C_{45}, D_{45}) = (B_{42}, \mathbf{B_{45}}, B_{44}, B_{43})$.

It can be seen that both $\overrightarrow{p_{45}}$ and $\overleftarrow{p_{48}}$ have B_{45} in common and thus B_{45} can become the matching variable. Hence, steps 45-47 need not be computed and can be skipped altogether enabling the attacker to cover 59 steps of MD5.

Similarly, several other techniques such as *local collision* [19, 150, 151], *partial fixing* [15, 17] etc. were proposed which allowed additional steps to be skipped, though the number of matching bits get reduced. With these techniques, preimages for 63-step MD5 [150], full MD4 (only one block) [150], 3-pass, 4-pass and 151-step 5-pass HAVAL [19, 151] were found.

4.1.2 Initial Structure

As mentioned in the above section, techniques mentioned above enable skipping of those steps which involved computation of neutral words. However, all these techniques have some limitations. For example, for partial matching approach to work, neutral words can be at most (L-1) steps away where L denotes the number of chaining variables, e.g., L = 4 for MD5. Similarly, local collision technique also requires neutral words to be some specific steps away from each other. Cases where independent chunks overlap each other in some part, i.e., (1^{st} chunk, (2^{nd} chunk, 1^{st} chunk), 2^{nd} chunk) and conditions required to apply the above discussed techniques are not satisfied create difficulties for the attacker because now she has no way to carry out the forward and backward computations independently.

Step i	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
$M_{(\pi)i}$	0	1	2	3	4	5	**6**	7	8	9	10	11	12	13	**14**	15
Step i	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
$M_{(\pi)i}$	1	**6**	11	0	5	10	15	4	9	**14**	3	8	**13**	2	7	12

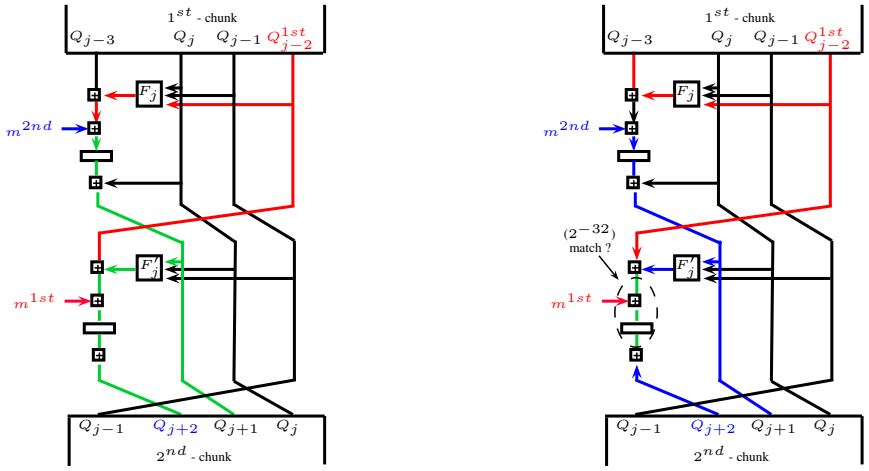
Table 4.2: Message Schedule of 31-step MD5.

For example, in Table 4.2, let,

1^{st} *Chunk*: Step 0 - Step 13, Neutral Word: m_6

2^{nd} *Chunk*: Step 18 - Step 31, Neutral Word: m_{14}

Now, in between the first and the second chunk, i.e., Step 14 - Step 17, m_{14} is used before m_6 and their positions are such that techniques like partial matching, partial fixing technique and local collision won't work (only 2 steps away from each other). The *initial structure* technique was proposed to solve such problems. The initial structure can be informally defined as an overlapping of chunks, where neutral bits, although formally belonging to both chunks, are involved in computation of the proper chunk only. To understand this concept let us have a look at a simple example shown in Fig. 4.4



(a) 2-step MD5.

(b) 2-step initial structure(IS) in MD5.

Figure 4.4: *Blue* denotes state influenced by m^{2nd}, *Red* denotes states updated by m^{1st} and *Green* denotes state updated by both m^{1st} and m^{2nd}.

Suppose the words (m^{1st}, Q_{j-2}) (highlighted in red) are neutral for the 1^{st} chunk and the words (m^{2nd}, Q_{j+2}) (highlighted in blue) are neutral for the 2^{nd} chunk. It can be seen in Fig. 4.4(a) that Q_{j+1} in the 2^{nd}-chunk is influenced by Q^{1st}_{j-2} (shown in green). This is undesirable since computation of subsequent steps in 2^{nd}-chunk will now depend on Q^{1st}_{j-2}. To carry out MITM attack, we need that Q_{j+1} depends only on Q_{j+2} and m^{2nd} (neutral words of the second chunk) and not on Q^{1st} (neutral word of the first chunk). *Initial Structure* technique helps us alleviate this problem. To make the 2^{nd} chunk independent of Q^{1st}_{j-2}, we choose Q_{j-3} such that it cancels the change of Q^{1st}_{j-2} (as shown in Fig. 4.4(b)) i.e.,

$$Q_{j-3} = -F_j(Q_j, Q_{j-1}, Q_{j-2}) - K_j - m^{2nd} + ((-Q_j) \ggg s_j).$$

By choosing Q_{j-3} according to the equation above makes Q_{j+1} independent of the influence of Q^{1st}_{j-2}. In the second chunk, we can now compute Q_{j+1} according to the

values of m^{2nd} and Q_{j+2} and carry out the computations without using Q_{j-2}^{1st}. Thus, a 2-step initial structure is formed which provides 2^{64} free bits for both the first and second chunks. This initial structure guarantees that the first and second chunks are independent of each other's neutral words and matching in the MITM phase succeeds with a probability of 2^{-32} for randomly given m^{1st}, Q_{j-2}, m^{2nd}, Q_{j+2}. Thus, *initial structure* is a group of few consecutive steps which includes neutral words in overlapping order. However, it allows computing of steps before and after the initial structure independent of neutral words of the second and the first chunk, respectively. With this technique, preimages could be found for full round of MD5 [152], 43-step SHA-256 [15] and 46-step SHA-512 [15] with a complexity of $2^{123.4}$, $2^{254.9}$ and $2^{511.5}$ respectively.

4.2 Biclique attack for finding preimages

The problem with the "initial structure" idea was that unlike other attacks, this approach was not generalizable and very construction specific. Khovratovich in [103] introduced the concept of biclique attack which is a more generic version of initial structure. Let us consider the Davies-Meyer mode: $h = E_M(CV) \oplus CV$, where CV is the chaining variable and E is the block cipher keyed with the message M. Let f be a sub-cipher of E, and $\mathcal{M} = \{M[i,j]\}$ be a group of messages which are parameters for f. Let the differences in a message group $\mathcal{M}$ be defined as:

$$\Delta_{i,j}^{M} = \Delta_{i}^{M} \oplus \nabla_{j}^{M}$$

Consider a single mapping,

$$Q_0 \xrightarrow[f]{M[0,0]} P_0 \tag{4.1}$$

which is called the *base computation*. Q is the input state of f and P is the output state of f. For each group, 2^d Δ_j forward differentials are constructed from Q_0 as follows:

$$Q_0 \xrightarrow[f]{M[0,j]} P_j \quad \text{or,} \quad 0 \xrightarrow[f]{\Delta_j^M} \Delta_j \tag{4.2}$$

Similarly, 2^d backward differentials ∇_i from P_0 are constructed as follows:

$$Q_i \xleftarrow[f]{M[i,0]} P_0 \quad \text{or,} \quad \nabla_i \xrightarrow[f]{\nabla_i^M} 0 \tag{4.3}$$

If the above two differentials do not share any active nonlinear components for all i and j, then *a biclique of dimension d* over f for $\mathcal{M}$ is formed as follows:

$$Q_i \xrightarrow[f]{M[i,j]} P_j \tag{4.4}$$

where,

$$\begin{aligned} Q_i &= Q_0 \oplus \nabla_i \\ P_j &= P_0 \oplus \Delta_j \\ M[i,j] &= M[0,0] \oplus \Delta^M_{i,j} \end{aligned}$$

The biclique attack works exactly the same way as described in the earlier chapter for block ciphers. For each group, once a biclique is formed, the attacker then selects a variable v outside of f and checks if:

$$P_j \xrightarrow{M[\cdot,j]} \overrightarrow{v} \stackrel{?}{=} \overleftarrow{v} \xleftarrow{M[i,\cdot]} Q_i \tag{4.5}$$

A positive answer yields a preimage candidate and should satisfy the following relation,

$$CV \xrightarrow{M[i,j]} Q_i \xrightarrow[f]{M[i,j]} P_j \xrightarrow{M[i,j]} H \tag{4.6}$$

otherwise the whole process is repeated for other message groups. To compute v from Q_i, the adversary first computes CV and then derives the output of E as $CV \oplus H$.

4.2.1 Biclique based Preimage Attack on SHA-2

In this section, we give a high level description of preimage attack on SHA-256 presented in [103]. The compression function of SHA-256 is based on Davies-Meyer mode of iteration and consists of 64-steps. It accepts a message input of 512-bits and produces an output of 256-bits. The intermediate state is divided into 8 registers $(A_i, B_i, \ldots, H_i)$, each of size 32-bits and at each step two registers, i.e., A_i and E_i are updated. A schematic view of SHA-2 compression function is shown in Fig. 4.5.

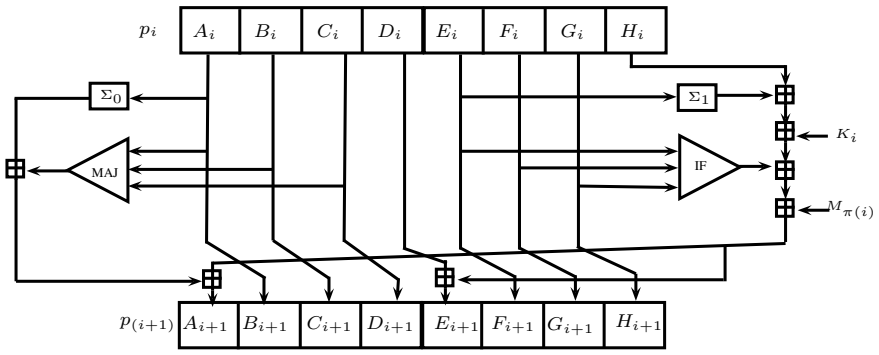


Figure 4.5: Compression Function of SHA-2.

Steps	State Registers							
	A	B	C	D	E	F	G	H
17						6, 11, 12, 16 17, 20, 23, 24 29, 30		
18	*				25, 26, 27		6, 11, 12, 16 17, 20, 23, 24 29, 30	
19	*	*			0, 1, 2, 14, 15, 16, 19, 20, 21, 22, 23, 31	25, 26, 27		6, 11, 12, 16 17, 20, 23, 24 29, 30
20	*	*	*		*	0, 1, 2, 14, 15, 16, 19, 20, 21, 22, 23, 31	25, 26, 27	
21	*	*	*	*	*	*	0, 1, 2, 14, 15, 16, 19, 20, 21, 22, 23, 31	25, 26, 27
22	*	*	*	*	*	*	*	0, 1, 2, 14, 15, 16, 19, 20, 21, 22, 23, 31

Table 4.3: Biclique trails in Steps $17-22$. $*$ stands for arbitrary difference. Δ_i-trail is shown in red and ∇_j-trail is shown in blue. The numbers mentioned in each cell denote the bits of the register (mentioned in the column header) affected by Δ_i and ∇_j trails. For example, the ∇ difference injected in the neutral message word m_{22} at bits 22, 23 and 31 is propagated to the bits 22, 23 and 31 of register H at step 22, i.e., $H^{22}_{22,23,31}$ from the backward direction.

For the preimage attack on SHA-256, a biclique is constructed over 6 steps (as shown in Table. 4.3). In this attack, message words m_{12} to m_{27} used in steps 12-27 are in the control of the attacker. Bits 25, 26, 27 of m_{17} are chosen as neutral bits for Δ_i trail in the forward direction whereas bits 22, 23, 31 of m_{22} are selected as neutral bits for ∇_j trail in the backward direction. As can be seen from the table, the bits influenced by Δ_i-trail and ∇_j-trail in the steps covered by the biclique are non-overlapping, thus making both the trails independent of each other. To ensure that Δ_i and ∇_j-trails affect non-overlapping bits in steps 17-22, certain conditions have been imposed on the initial chaining value and message words in the attack (42 conditions [103]). For the MITM stage, steps 2-16 form the 1^{st} *chunk* (independent of m_{17}) whereas steps 23-36 form the 2^{nd} *chunk* (independent of m_{22}). $A^{38}_{0,1,2,3}$, i.e., bits 0, 1, 2, 3 of register A in step 38 form the matching variable v. Some of the steps in each chunk are dependent on neutral words belonging to the other chunk due to message scheduling of SHA-2 [15].

To mitigate their effect, *message compensation* technique [15] is used. For example, at step 3 in the 1^{st} *chunk*, message word m_3 is used to update the intermediate states. Now, according to message schedule algorithm,

$$m_3 = m_{19} - \sigma_1(m_{17}) - m_{12} - \sigma_0(m_4) \tag{4.7}$$

where, $\sigma_0(\cdot)$ and $\sigma_1(\cdot)$ are pre-defined functions in SHA-2 algorithm. Now, since m_3 is influenced by m_{17}, which should not be the case, the attacker chooses m_{19} such that, $m_{19} = \sigma_1(m_{17})$. [1] This way, the effect of m_{17} is canceled and message word m_3 is made independent of m_{17} which in turn leads to Step-3 in 1^{st} *chunk* being independent of m_{17}. With this attack, preimages for 45 steps of SHA-256 and 50 steps of SHA-512 were found. Similarly, 22-steps of Skein-512 were also targeted. Table 4.4 summarizes the biclique based preimage attacks on Skein-512 and SHA-2 based hash functions.

Target	Steps	Complexity	Memory	Reference
Skein-512	22	2^{511}	2^6	[103]
SHA-256	45	$2^{255.5}$	2^6	[103]
SHA-512	50	$2^{511.5}$	2^4	[103]

Table 4.4: Biclique based Preimage Attacks on SHA-2 and Skein-512 family

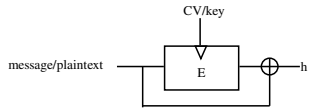
4.3 Preimage Attack on AES-128 based Hashing Modes

After the introduction of biclique attack for generating preimages on Skein and SHA- 2 based hash functions, the concept was utilized by Bogdanov et al. [39] to successfully recover the secret key for full rounds of all AES variants. A natural extension was to try generating preimages for AES based hash functions. Bogdanov et al. in [39] showed translation of biclique key recovery attack on AES to the corresponding preimage attack on AES instantiated compression function. The current best complexity of this attack as reported in [37] is $2^{125.35}$, $2^{125.51}$ and $2^{125.93}$ for AES-128, AES-192 and AES-256 instantiated compression functions respectively with a probability of 0.632.

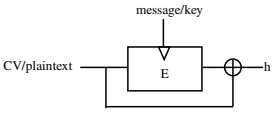
The above biclique based preimage attack on AES-128 instantiated compression function cannot be converted to preimage attack on the corresponding hash function (and hence second preimage attack as discussed in Section 4.6 later). This is due to the fact that in the preimage attack on compression function shown in [37,39], the attacker needs to modify the value of the chaining variable (CV) and the message input to obtain the desired preimage. However, in hash function settings, the initialization

[1] Recall, we had said m_{12} - m_{27} are in the control of the attacker.

vector (IV) is a publically known constant which cannot be altered by the attacker. Hence, the biclique trails used in the preimage attack on AES-128 based compression function in [37, 39] cannot be adopted to find preimage for the corresponding AES-128 based hash function. This is explained in more details next.



(a) MMO mode



(b) DM mode

Figure 4.6: Compression Function in MMO and DM mode respectively.

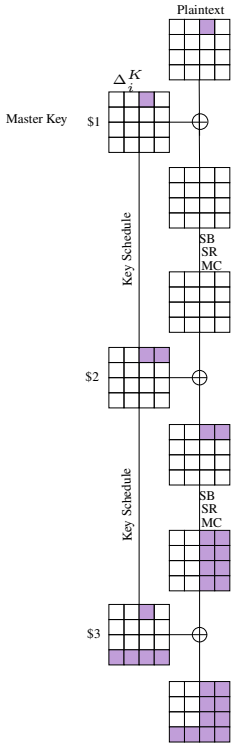


Figure 4.7: An example of the trail used in [39] for preimage attack on AES-128 instantiated compression function.

Let us consider Matyas-Meyer-Oseas (MMO) mode and Davies-Meyer (DM) mode based compression functions as shown in Fig. 4.6(a) and 4.6(b) respectively. In the case of MMO mode, the chaining variable acts as the key input to the underlying block cipher AES (as shown in Fig. 4.6(a)). If the chaining variable is used as the IV (in hash function settings) then it is fixed and cannot be modified. This means that the value of the key input to the block cipher should not change. However, the type of biclique trails used in [39] (as shown in Fig. 4.7) for compression function introduce a change both in the key input as well as all the intermediate states including the plaintext input ensuring that the final chaining variable so obtained after the attack will not be the desired IV. Hence, the kind of biclique trails we are interested in should only affect the intermediate states (an example of which is given in Fig. 4.8) and not the key input.

Similarly, in the DM mode, the chaining variable acts as the plaintext input to the

underlying block cipher (as shown in Fig. 4.6(b)). Therefore, if the chaining variable under consideration is the IV, then the chosen biclique trails should not inject any difference in the plaintext input of the block cipher (an example of the same is shown in Fig. 4.9). Again, the biclique trails adopted for preimage attack on AES-128 instantiated compression function do not satisfy this condition (as seen in Fig. 4.7).

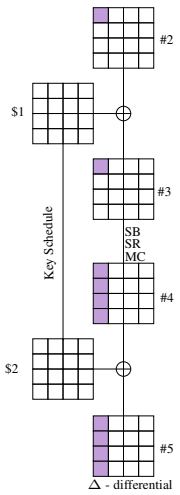


Figure 4.8: An example of the desired trails that will work for attacking MMO based hash function. It is to be noted only the plaintext input and subsequent intermediate states are affected in the trail considered whereas the key input is a fixed constant.

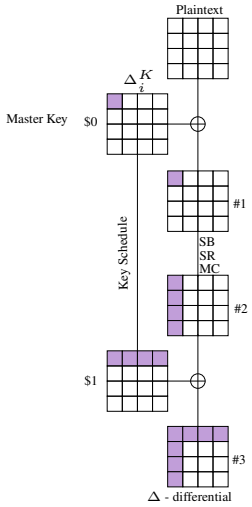


Figure 4.9: An example of the desired trail that will work for attacking DM based hash function. It is to be noted here that the plaintext input is not affected by the differential trail so chosen and is a fixed constant.

The examples discussed above warrant searching of new biclique trails which can be used to launch second preimage attack on AES-128 based hash functions. Moreover, searching these trails manually may not give the best results as demonstrated in [5,37]. Hence, automated search process is required. Coupled with the fact that no prior work on the analysis of block cipher based hash functions based on biclique technique exists, motivated us to apply biclique attack to evaluate the security of AES-128 based hash functions against second preimage attack. In this work, we implemented our restrictions in C programs to enumerate the best biclique trails which guarantee the lowest possible attack complexities.

4.3.1 Our Contributions

- We re-evaluate the offered security of full 10 rounds AES-128 based hash functions against second preimage attack. This improves upon the previous best result on

AES-128 based hash functions by Sasaki at FSE'11 [149] where, the maximum number of rounds attacked is 7.

- Our analysis is applicable to all the 12 PGV modes of the hash function constructions.
- The complexities of the biclique based analysis differ depending upon the PGV construction chosen. For MP and MMO mode it is $2^{126.3}$ whereas for DM mode it is $2^{126.67}$.
- We propose new biclique trails to achieve the above results.
- All the trails have been obtained by implementing C programs which ensure that they yield the best attacks (lowest possible time complexity).

The results of our security evaluation against second preimage attack on all 12 PGV based modes are given in Table 4.5. Though our results do not significantly decrease the attack complexity in comparison to brute force attack but they highlight the actual security margin provided by these constructions against second preimage attack.

Table 4.5: Summary of the second preimage attack results obtained. In this table, we assume the hash function to be instantiated with the block cipher E, h is the chaining variable, m is the message input and $h \oplus m = w$. The brute-force complexity of these attacks is 2^{128}.

S.No.	Hash Function Modes	Second Preimage Complexity
1	$E_h(m) \oplus m$ - MMO	$2^{126.3}$
2	$E_h(m) \oplus w$ - MP	$2^{126.3}$
3	$E_m(h) \oplus h$ - DM	$2^{126.6}$
4	$E_h(w) \oplus w$ - similar to MMO	$2^{126.3}$
5	$E_h(w) \oplus m$ - similar to MMO	$2^{126.3}$
6	$E_m(h) \oplus w$ - similar to DM	$2^{126.6}$
7	$E_m(w) \oplus h$ - similar to DM	$2^{126.6}$
8	$E_m(w) \oplus w$ - similar to DM	$2^{126.6}$
9	$E_w(h) \oplus h$ - similar to DM	$2^{126.6}$
10	$E_w(h) \oplus m$ - similar to DM	$2^{126.6}$
11	$E_w(m) \oplus h$ - similar to MP	$2^{126.3}$
12	$E_w(m) \oplus m$ - similar to MMO	$2^{126.3}$

4.4 Notations

To facilitate better understanding, we use the following notations in the rest of the chapter.

$\mathbf{CV}$	:	Chaining Variable
$\mathbf{IV}$	:	Initialization Vector
$(\mathbf{CV}, \mathbf{message})$	:	Input tuple to hash function/ compression function
$(\mathbf{key}, \mathbf{plaintext})$	:	Input tuple to underlying block cipher
$\mathbf{n}$	:	Input message/key size (in bits)
$\mathbf{A_b}$	:	Base State
$\mathbf{m_b}$	:	Base Plaintext
$\mathbf{K_b}$	:	Base Key
$\mathbf{K}[\mathbf{i}, \mathbf{j}]$	:	Keys generated by Δ_i and ∇_j modifications
$\mathbf{M}[\mathbf{i}, \mathbf{j}]$	:	Messages generated by Δ_i and ∇_j modifications
$\mathbf{Nbr}$	:	Number of AES rounds called
$\mathbf{E_{enc/dec}}$	:	One Round of AES encryption/decryption
$\mathbf{E}(\mathbf{x}, \mathbf{y})$	:	Full AES encryption under y-bit key and x-bit message
$\mathbf{E^{-1}}(\mathbf{x}, \mathbf{y})$	:	Full AES decryption under y-bit key and x-bit message

For the sake of clarity, we will follow the same notation used for description of AES-128 in Section 3.3.1. We address two internal states in each round as follows: #1 is the state before SubBytes in round 1, #2 is the state after MixColumns in round 1, #3 is the state before SubBytes in round 2, ... , #19 is the state before SubBytes in round 10, #20 is the state after ShiftRows in round 10. The key K is expanded to a sequence of keys $K^0, K^1, K^2, \ldots, K^{10}$, which form a 4×44 byte array. Then the 128-bit subkeys $\$0, \$1, \$2, \ldots, \$10$ come out of the sliding window with a 4-column step. We refer the reader to [57] for a detailed description of AES.

4.5 Biclique based Preimage Attack on AES-128 instantiated Compression Function

In this section, we examine how biclique key recovery attack discussed in Section 3.2 can be applied to find preimage for block cipher based compression function. This preimage attack on compression function will then be used to evaluate second preimage resistance of AES-128 based hash functions under different PGV modes as discussed in Section 4.6.

Let us consider an AES-128 based compression function (as shown in Fig. 4.10). To find the preimage for h, the attacker needs to find a valid (CV, message) pair which generates h. In terms of the underlying block cipher E which is instantiated with AES-128, this problem translates to finding a valid (plaintext, key) pair where both the key and the plaintext are of 128-bits size. To guarantee the existence of a preimage for

h (with probability 0.632), the attacker needs to test 2^{128} distinct (key, plaintext) pairs.

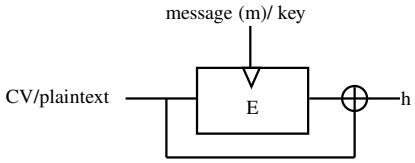


Figure 4.10: AES-128 instantiated compression function in DM mode.

When biclique methodology is applied on AES-128 to recover the secret key [39], full key space, i.e., 2^{128} keys are divided into 2^{112} groups of 2^{16} size each and tested. [2] These 2^{112} groups are generated from 2^{112} base key values where each base value defines one group. However, the same biclique approach when extended to hash functions warrants the need of testing 2^{128} (key, plaintext) pairs. These 2^{128} (key, plaintext) pairs will be generated from 2^{112} (key, plaintext) base states. Hence, under hash function settings, along with the *base key* we introduce the term "*base message*". We denote the base key value as K_b and the base message value as A_b. If we apply the original biclique approach [39] on compression function, then 2^{128} (key, plaintext) pairs are generated from a combination of $2^{112}(K_b, A_b)$ as shown in (Fig. 4.11).

$(K_b^{(1)}, A_b) \longrightarrow 2^{16}$ (key, message) pairs
$(K_b^{(2)}, A_b) \longrightarrow 2^{16}$ (key, message) pairs
$(K_b^{(3)}, A_b) \longrightarrow 2^{16}$ (key, message) pairs
$\vdots$
$(K_b^{(2^{112})}, A_b) \longrightarrow 2^{16}$ (key, message) pairs

Figure 4.11: Generation of groups in the original biclique attack [39]

Algorithm 1 :

Fix a base state A_b
for *each* 2^{112} *base keys* ($K_b's$) *and the fixed chosen* A_b **do**
- Generate 2^{16} (Δ_i^k, ∇_j^k) combinations
- Generate corresponding $2^{16} K[i,j]$
- Construct a biclique structure using these $2^{16} K[i,j]$
- **for** *each* $2^{16} K[i,j]$ **do**
  1. Generate $M[i,j]$ (where $M[i,j]$ = Nbr $E_{enc/dec}(K[i,j], A_b)$)
  2. Perform meet-in-the-middle attack in the rest of the rounds

Figure 4.12: Steps of the original biclique attack in [39] using the base key K_b and the base message A_b.

[2] In this work, bicliques of dimension $d = 8$ are constructed. In our attacks, we also construct bicliques of dimension 8.

In this case, a single A_b is chosen and repeated across all the groups whereas 2^{112} different $K_b's$ are used. The biclique algorithm for the attack is shown in Fig.4.12. In *Algorithm 1*, the specific (i,j) tuple for which a match is found gives us the corresponding $K[i,j]$ and $M[i,j]$ as the desired inputs for compression function. The complexity of this attack when applied for searching preimages in AES-128 instantiated compression function is $2^{125.35}$ [37].

In the procedure described above, it can be seen that the attacker generates a chaining value ($M[i,j]$) of her own along with the preimage ($K[i,j]$). However, as already discussed, the IV value is a public constant in the hash function setting and cannot be altered by the attacker. In the subsequent section, we show how to utilize variants of the above framework for launching second preimage attack on AES-128 based hash functions in different PGV modes with IV being fixed.

4.6 Second Preimage Attack on Hash Functions

In this section, we examine the feasibility of extending the biclique cryptanalysis technique for second preimage attack on AES-128 instantiated hash functions for all 12 PGV modes.

4.6.1 PGV Construction 1 - Matyas-Meyer-Oseas (MMO) Mode: $E_h(\mathbf{m}) \oplus \mathbf{m}$

Consider MMO based hash function (as shown in Fig. 4.13), where the (chaining variable, message block) tuple acts as the (key, plaintext) inputs respectively to block cipher E. In this case, the attacker is given $m = (m_0 \mid\mid m_1 \mid\mid pad)$ and its corresponding hash value h_2. Her aim is to find another different message, m' that will produce the same h_2. To achieve so, the attacker can consider m' as $(m'_0 \mid\mid m_1 \mid\mid pad)$ where the second half of m' is same as message m while for the first half, the attacker has to carry a biclique attack. For the first half, i.e., $h_1 := E_{IV}(m'_0)$, the attacker knows h_1 and IV. Her aim is now to find a preimage m'_0 which produces h_1 under the given IV. The attack steps are as follows:

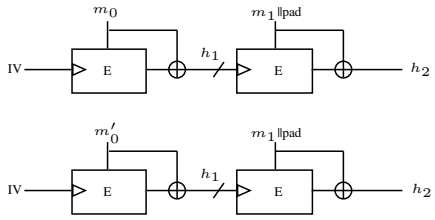
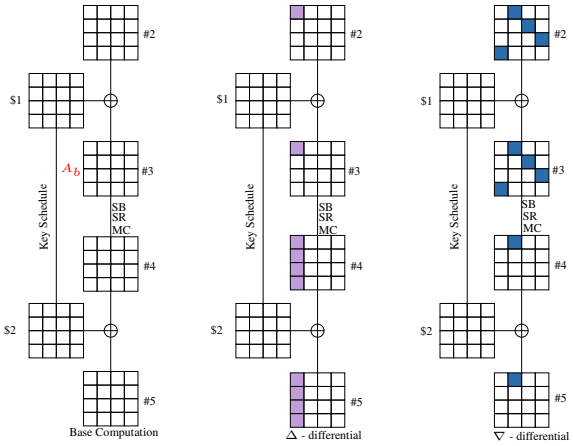


Figure 4.13: Second Preimage attack on MMO based hash function

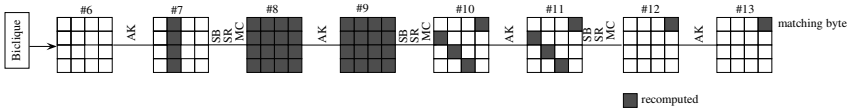
1. The attacker fixes IV as the key input to the block cipher E & chooses a 128-bit base message A_b.

2. ***Choice of biclique structure***. In this case, the key input to the block cipher (i.e., IV) is fixed. The attacker has to choose a biclique structure such that the Δ_i and ∇_j trails only modify the message states and not the key states (since IV cannot change) plus the biclique attack should have lowest search complexity. All the existing biclique trails in literature allow modification in the keys states as well, therefore, we construct new biclique trails to suit our needs.

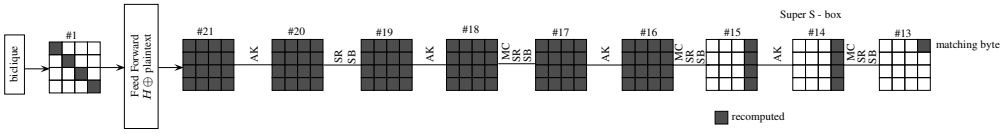
3. We represent the Δ and ∇ trails as Δ_i^m and ∇_j^m respectively. The biclique structure satisfying the above requirements is as shown in Fig. 4.14(a).



(a) Biclique over 1.5 rounds



(b) Forward Recomputations



(c) Backward Recomputations

Figure 4.14: Biclique structure for MMO mode when key/IV is known

4. For the above biclique, she divides the 128-bit message space into 2^{112} groups each having 2^{16} messages with respect to intermediate state #3 as shown in Fig. 4.14(a). The base messages are all 16-byte values with two bytes (i.e., bytes 0 and 4) fixed to 0 whereas the remaining 14-bytes taking all possible values

(shown in Fig. 4.15). The messages in each group ($M[i,j]$) are enumerated with respect to the base message by applying difference as shown in Fig. 4.16. The proof for the claim that this base message (with the corresponding Δ_i and ∇_j differences) uniquely divides the message space into non-overlapping groups is given in Appendix A.1.

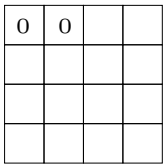


Figure 4.15: Base Message

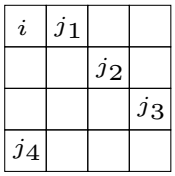


Figure 4.16: Δ_i and ∇_j differences

5. The biclique covers 1.5 rounds (round 2 and round 3 upto *Shift Rows* operation). Δ_i^m trail activates byte 0 whereas ∇_j^m trail activates bytes 3,4,9 and 14 of #3 state.

6. Meet-in-the-middle attack is performed on the rest 8.5 rounds. In the MITM phase, partial matching is done in byte 12 of state #13. In the backward direction, Δ_i^m trail activates 4 bytes in the plaintext i.e., byte 0, 5, 10 and 15 whereas ∇_j^m activates all bytes. As such, during the recomputation phase, the 4 bytes of plaintext affected by both Δ_i^m and ∇_j^m trails need to be recomputed. Similar explanation can be provided for other bytes shown to be recomputed in Figs. 4.14(b) and 4.14(c). In the forward propagation (starting from round 4), 4+16+4 = 24 S-boxes and in the backward propagation (starting from round 1), 4+16+16+4+1= 41 S-boxes are recomputed. Thus, a total of 65 S-boxes are involved in the recomputation process. One full AES encryption requires 200 S-box computations. As each group has 2^{16} messages, $C_{recomp} = 2^{16} \times \frac{65}{200} = 2^{14.3}$. Hence, $C_{full} = 2^{112} \times 2^{14.3} = 2^{126.3}$.

7. For the specific (i,j) value which produces a match in the middle, the corresponding $M[i,j]$ i.e., xoring of #3 states in base computation, Δ_i and ∇_j trails (in Fig. 4.14(a)) yields the plaintext m'_0 for the block cipher E. The biclique algorithm, i.e., *Algorithm 2* is as shown in Fig. 4.17.

Thus, with a time complexity of $\mathbf{2^{126.3}}$, the attacker is able to find a (IV, m'_0) pair which produces hash value h_1 and $m' = (m'_0 \mid\mid m_1 \mid\mid pad)$ forms a valid second preimage.

Algorithm 2:

Fix a base state A_b
for *each* 2^{112} *base messages* ($A_b's$) *and a single base key i.e., IV* **do**
    Generate 2^{16} (Δ_i^m, ∇_j^m) combinations
    Generate corresponding $2^{16} M[i,j]$
    Construct a biclique structure using these $2^{16} M[i,j]$
    for *each* $2^{16} M[i,j]$ **do**
        1. Perform meet-in-the-middle attack in the rest of the rounds

Figure 4.17: Steps of the new biclique attack when key input to the underlying block cipher is fixed and cannot be modified by the attacker for MMO mode.

4.6.2 PGV Construction 2 - Miyaguchi-Preneel Mode (MP) Mode: $\mathbf{E_h(m) \oplus m \oplus h}$

The MP mode is an extended version of MMO mode. The only difference between the two constructions is the fact that output of block cipher is xor'ed both with the plaintext input as well the chaining variable input. However, this does not demand any extra attack requirements and the second preimage attack on MP mode is exactly the same as that described on MMO mode.

4.6.3 PGV Construction 3 - Davies-Meyer (DM) Mode: $E_m(\mathbf{h}) \oplus \mathbf{h}$

In the DM based hash function (as shown in Fig. 4.18), the (chaining variable, message block) tuple acts as the (plaintext, key) inputs respectively to block cipher E.

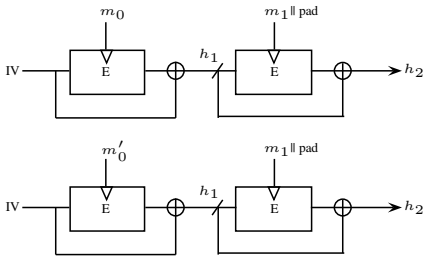
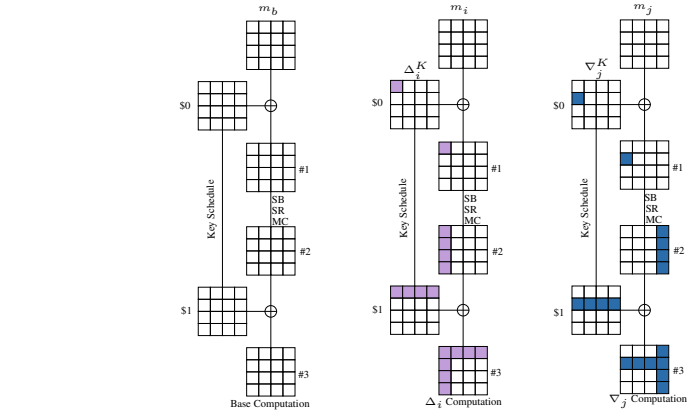


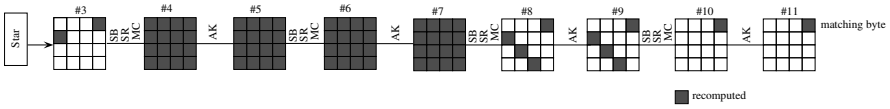
Figure 4.18: Second Preimage attack on DM based hash function

We again inspect a similar scenario as described in 4.6.1, i.e., for a message m = $(m_0 \mid\mid m_1 \mid\mid pad)$, the attacker is given its corresponding hash value h_2. Her aim is to find another different message m' that will produce the same h_2. Consider the hash function as concatenation of two compression functions - $E_{m_0}(IV)$ and $E_{m_1 \mid\mid pad}(h_1)$. To get a valid second preimage, the attacker chooses m' as - $(m'_0 \mid\mid m_1 \mid\mid pad)$ i.e., she focuses on the first compression function and her aim is to find m'_0 such that $E_{m'_0}(IV)$ = h_1 when IV and h_1 are known to the attacker. The attack steps are as follows:

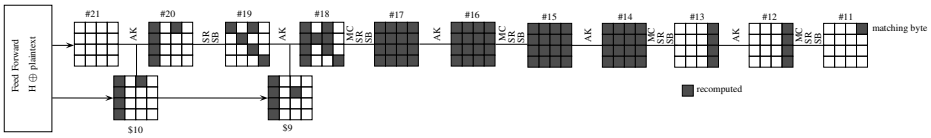
1. The attacker fixes the IV as the plaintext input to the block cipher.
2. ***Choice of biclique structure***. Under the given attack scenario, since the message input, i.e., IV is fixed, the attacker has to choose a biclique structure such that the Δ_i and ∇_j trails do not modify the plaintext state and the biclique attack has the lowest search complexity. The biclique structure satisfying the above requirements is given in Fig. 4.19(a). [3]



(a) Biclique over first round



(b) Forward Recomputations



(c) Backward Recomputations

Figure 4.19: Biclique structure for DM mode when IV/message input is known to the attacker

[3]We utilize star based bicliques here as discussed in Section 3.6

3. For the above biclique, she divides the 128-bit key space into 2^{112} groups, each having 2^{16} keys with respect to subkey \$0 i.e., the master key as the base key as shown in Fig. 4.19(a). The base keys are all 16-byte values with two bytes (i.e., bytes 0 and 1) fixed to 0 whereas the remaining 14-bytes taking all possible values (shown in Fig. 4.20). The keys in each group ($K[i, j]$) are enumerated with respect to the base key by applying difference as shown in Fig. 4.21. It can be easily verified that this base key uniquely divides the key space into non-overlapping groups.

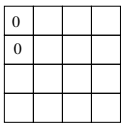


Figure 4.20: Base Message

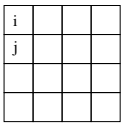


Figure 4.21: Δ_i and ∇_j differences

4. The biclique covers the first round. Δ_i trail activates byte 0 of \$0 subkey whereas ∇_j trail activates byte 1 of \$ 0 subkey.

5. The attacker then performs meet-in-the-middle attack on the rest of the 9 rounds. In the MITM phase, partial matching is done in byte 12 of state #11. In the forward propagation (starting from round 2), 2+16+16+4 = 38 S-boxes and in the backward propagation (starting from round 10), 5+16+16+4+1 = 42 S-boxes need to be recomputed (as shown in Figs. 4.19(b) and 4.19(c)). 2 S-box recomputations in the key schedule are also required. Thus a total of 82 S-boxes are involved in recomputation process. One full AES encryption requires 200 S-box computations. As each group has 2^{16} keys, $C_{recomp} = 2^{16} \times \frac{82}{200} = 2^{14.6}$. Hence, $C_{full} = 2^{112} \times 2^{14.6} = 2^{126.6}$.

6. For the specific (i, j) value which produces a match in the middle, the corresponding $K[i, j]$ forms the key (m_0) for the block cipher E. The biclique algorithm, i.e., *Algorithm 3* is given in Fig. 4.22.

Thus with a time complexity of $\mathbf{2^{126.6}}$, the attacker is able to find a (IV, m'_0) pair which produces hash value h_1 and $m' = (m'_0 \mid\mid m_1 \mid\mid pad)$ forms a valid second preimage.

The attack procedure on two block message for other constructions is similar to those discussed in Section 4.6.1-Section 4.6.3. Their results are given in Table 4.5.

4.7 Second Preimage attack on hash functions extended to messages with message length ≥ 3.

The second preimage attack discussed in the previous sections can be extended to messages of any length > 2 with same complexity as obtained for 2-block messages. To

Algorithm 3 :

for *each* 2^{112} *base keys* ($K_b's$) *and the fixed chosen IV* **do**
  Generate 2^{16} (Δ_i^k, ∇_j^k) combinations
  Generate the corresponding $2^{16}K[i,j]$
  Construct a biclique structure using these $2^{16}K[i,j]$
  for *each* $2^{16}K[i,j]$ **do**
    1. Perform meet-in-the-middle attack in the rest of the rounds

Figure 4.22: Steps of the new biclique attack when message input is fixed and known to the attacker under DM mode

demonstrate this, consider a MMO-based hash function with 3-block message as shown in Fig. 4.23. In this case, the attacker is given a message $m = (m_0 \mid\mid m_1 \mid\mid m_2 \mid\mid pad)$ and its corresponding hash value h_3. Her aim is to find another message m' such that $H(m') = H(m)$. The attacker knows IV and the compression function E. She will choose any m_0 of her own choice, e.g., let $m_0 = 0$, and then calculate $h_1 = E_{IV}(0)$. Once she knows h_1, the setting is reduced to the case discussed in Section 4.6.1, i.e., h_1 and h_2 are known to the attacker and her aim is to find m_1' such that $m' = (0 \mid\mid m_1' \mid\mid m_2 \mid\mid pad)$ forms a valid second preimage. This can be found with a complexity of $2^{126.3}$ which is same as that shown for a 2-block message . Similarly, the attack can be applied on other long messages for all other PGV modes.

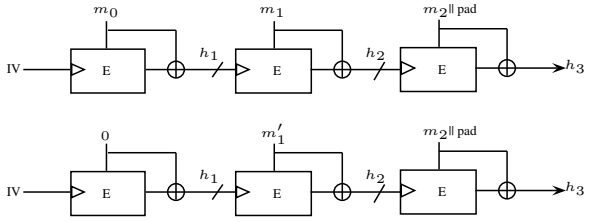


Figure 4.23: MMO base hash function with $\mid m \mid$=3

4.8 Summary

In this chapter, we discussed the application of biclique attack in hash function settings to find preimages. We reviewed a couple of existing biclique based preimage attacks on some dedicated hash functions and then evaluated the security of AES-128 based hash modes against second preimage attack. Specifically, we examined the applicability of biclique attack on all 12 PGV modes when instantiated with AES-128 and showed

that best biclique attack for finding preimages in AES-128 instantiated compression function did not translate to best attack for second preimage search under AES-128 based hash function settings. Similar attacks would work for AES-192 and AES-256 instantiated hash functions as well. A natural research extension to this work would be to apply the ideas discussed in this chapter to hash functions instantiated with other block ciphers. Another research direction can be to extend the methodology to carry out collision attacks on hash functions.

Chapter 5

Sliced Biclique Cryptanalysis of Type-2 Generalized Feistel Networks

In this chapter, we discuss another variant of biclique cryptanalysis termed as *sliced biclique cryptanalysis*. We revisit hash function settings and study the application of biclique cryptanalysis to generate collisions. All the biclique related attacks on block ciphers are carried out under the "unknown key settings" where the key used is unknown to the attacker and the main motive is to recover the secret key. However, this may not always be the case. Particularly, in the case of block cipher based hash modes such as Matyas-Meyer-Oseas (MMO) and Miyuguchi-Preneel (MP), initial vector IV (which acts as the key to the underlying block cipher) is a fixed public constant assumed to be known apriori to the attacker. Such scenarios are called "known key settings" in the attack model. Under such conditions, the aim of the attacker is to find a property which distinguishes known key instantiations of target block cipher from random permutations [109,132]. These settings are considered much stronger from the attacker's point of view since she unwillingly loses some degree of freedom (as in MMO mode: $E_{CV}(M) \oplus M$, if the CV is fixed, then the attacker can only choose the input and cannot manipulate the round injections) reducing chances of carrying out generic attacks such as finding full collisions. Until recently, most of the collision attacks on hash functions under MMO and MP modes were restricted to variants of generic attack such as pseudo-collisions [121] and near collisions [165]. In [99], Khovratovich used biclique technique to mount collisions and preimage attacks on Grøstl and Skein under known key settings. He proposed a variant of classical biclique technique used in [39] to carry out his attack. He termed this variant as *sliced biclique* cryptanalysis. Sliced biclique technique is a translation of the regular biclique technique applied to permutations, i.e., block ciphers with non-modifiable fixed key input. Though the results of this work are quite interesting, yet they have not been studied further. In this chapter, we discuss sliced biclique cryptanalysis and apply it to study Type-2 Generalized Feistel Network (GFN) based constructions under known key settings. Although the security

of GFNs have been studied earlier under known key settings [50, 68, 96, 148, 153], all these previous studies have utilized rebound attack technique [118] for their cryptanalysis. Hence, one of our aims was to investigate how other cryptanalytic techniques could be used to exploit non-ideal properties of generalized Feistel structures.

This chapter is organized as follows: We first discuss sliced biclique and sliced biclique based preimage attack in Section 5.1. We then describe Type-2 Generalized Feistel Network in Section 5.2 along with the current cryptanalytic results existing on it. In Section 5.3 we describe the notations used in this chapter followed by Section 5.4 which explains the important preliminaries. In Section 5.5, we present our distinguishing attack on 8 rounds of 4 branch, Type-2 GFN under fixed key settings. We use this distinguishing attack to show collision attack on 4-branch, Type-2 GFN based compression function in Section 5.6 followed by the extension of this attack to hash functions in Section 5.7. The collision attack on CLEFIA based hash function is discussed in Section 5.8. Finally in Section 5.9, a summary of this chapter is presented. The original contribution of this thesis is from Section 5.2 to Section 5.8.

5.1 Sliced Biclique Cryptanalysis

Sliced biclique cryptanalysis is a variant of biclique attack that works under the *known key settings*. At the core of this attack technique is the construction of *sliced biclique*. As against regular biclique, the term *sliced biclique* not only defines construction of a different biclique structure but also a different matching variable v that will be used in the MITM phase of the sliced biclique attack. In the subsequent subsections, we first define a sliced biclique. This is followed by a description of the biclique structure constructed in the sliced biclique. We then discuss sliced biclique cryptanalysis by showing a preimage attack on block cipher based compression function using sliced biclique.

5.1.1 What is a sliced biclique ?

Let us consider the MMO mode, $H = E_{IV}(M) \oplus M$, where IV is the initial chaining value acting as the key for the block cipher E, M is the message and H is the hash value produced. Since, we assume IV to be public and hence known to the attacker, the cipher E becomes a simple permutation, i.e., $H = E(M) \oplus M$.

Let Q be an internal intermediate state within E whose full state space is partitioned into sets of size 2^{2d} represented as $Q_{i,j}$ where, variables i and j take all d-bit values for some constant d, i.e., $(0 \leq i, j \leq 2^d - 1)$. Let f be a sub-permutation within E which maps $Q_{i,j}$ to another set of intermediate states $P_{i,j}$, i.e.,

$$\forall i, j \quad Q_{i,j} \underset{f}{\rightarrow} P_{i,j}$$

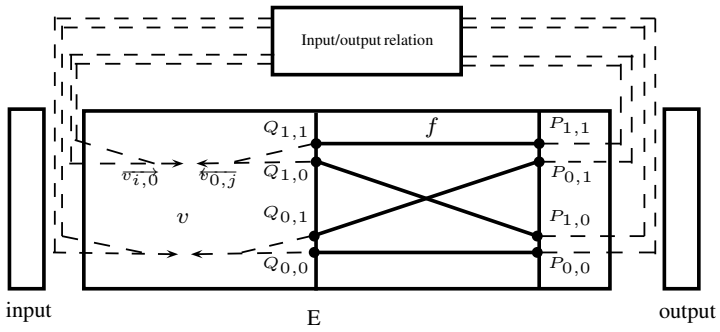


Figure 5.1: Sliced biclique for a permutation E [99].

Let, v be an intermediate state within E that lies outside f, i.e., $v \in \{E \setminus f\}$. Further, let the computation of value of v in the forward direction as a function of P and in the backward direction as a function of Q be denoted as $\overrightarrow{v_{i,j}}$ and $\overleftarrow{v_{i,j}}$ respectively (as shown in Fig. 5.1). A sliced biclique can then be defined as follows:

Definition 1 *Sliced Biclique [99].*

Given a permutation E and intermediate states $Q_{i,j}$, $P_{i,j}$ and v within E as defined above, the states $Q_{i,j}$ and $P_{i,j}$ form a sliced biclique *if, there exists a v such that:*

$$\begin{aligned} \forall i,j: \quad & \overrightarrow{v_{i,j}} = \overrightarrow{v_{0,j}}, \\ \forall i,j: \quad & \overleftarrow{v_{i,j}} = \overleftarrow{v_{i,0}} \end{aligned}$$

i.e., the value of intermediate state v is only influenced by j in the forward direction and by i in the backward direction.

The choice of sub-permutation f and partitioning of Q into $Q_{i,j}$ is described in the next subsection.

5.1.2 Construction of biclique structure in a sliced biclique

The mapping $Q_{i,j} \underset{f}{\rightarrow} P_{i,j}$ in Fig. 5.1 represents the biclique structure in a sliced biclique. The attacker first selects an internal intermediate state Q and partitions the full state space into sets of size 2^{2d} represented as $Q_{i,j}$ where, $0 \leq i,j \leq 2^d - 1$ for some d. Each set is defined by its base state $Q_{0,0}$ which is randomly selected by the attacker. She then choose two sets of differences – Δ_i and ∇_j, and construct a biclique where:

$$Q_{i,j} = Q_{i,0} \oplus \nabla_j \tag{5.1}$$
$$P_{i,j} = P_{0,j} \oplus \Delta_i \tag{5.2}$$

These $Q_{i,j}$ and $P_{i,j}$ are obtained using 2^d Δ_i and ∇_j differentials as follows:

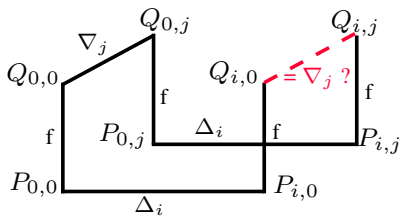


Figure 5.2: Boomerang quartet representation of biclique construction

1. The attacker computes, $Q_{0,0} \underset{f}{\rightarrow} P_{0,0}$ (base computation).
2. She then sets $Q_{0,j} = Q_{0,0} \oplus \nabla_j$
3. She computes $Q_{0,j} \underset{f}{\rightarrow} P_{0,j}$
4. She then sets $P_{i,j} = P_{0,j} \oplus \Delta_i$ ($\implies$ $P_{i,0} = P_{0,0} \oplus \Delta_i$)
5. She computes $Q_{i,j} \underset{f}{\leftarrow} P_{i,j}$

Now, here it can be seen that Eq. 5.2 has already been satisfied by definition. For Eq. 5.1 to satisfy, it is necessary that the states $Q_{0,0}$, $Q_{0,j}$, $Q_{i,0}$ and $Q_{i,j}$ form a boomerang quartet (as shown in Fig. 5.2).

To achieve so, the attacker will choose Δ_i and ∇_j trails such that their propagation through f do not share any active non-linear component between them. Then, the states $Q_{0,0}$, $Q_{0,j}$, $Q_{i,0}$ and $Q_{i,j}$ are guaranteed to form a boomerang. The proof of this property has already been discussed in Section 3.1.2.1. Thus, both Eqs. 5.1, 5.2 will be satisfied and a biclique will be constructed. [1] Each $Q_{0,0}$ defines one biclique structure consisting of 2^{2d} intermediate states where, the parameter d is called the dimension of the biclique.

5.1.3 Preimage attack using sliced biclique

In this section, we describe a preimage attack using sliced biclique technique. The attacker is given hash output $H = E(x) \oplus x$. Her aim is to find the preimage x.

To do so, the attacker considers the block cipher E as a composition of two sub-permutations: $E = f \circ g$ and splits the state space of intermediate state Q into 2^{2d} groups. For each group of 2^{2d} states:

[1] It is not necessary for independent biclique/sliced biclique attack to have Δ and ∇ differentials start from distinct ends of the subcipher. The only requirement that is essential is that both trails should be non-interleaving.

1. The attacker first constructs a biclique structure:

$$\forall i, j \quad Q_{i,j} \underset{f}{\rightarrow} P_{i,j}$$

as described in Section 5.1.2.

2. Once a biclique is constructed, the attacker the chooses an internal state $v \in g$ and computes its value both in the forward direction as a function of P (denoted as $\overrightarrow{v_{i,j}}$) and in the backward direction as a function of Q (denoted as $\overleftarrow{v_{i,j}}$) respectively for every (i, j) pair.

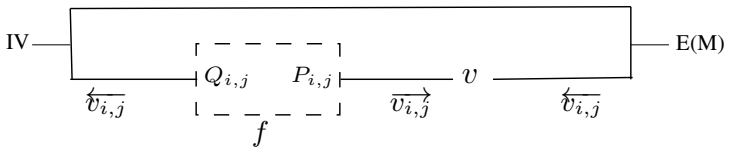


Figure 5.3: Biclique Attack.

To compute $\overleftarrow{v}$ in the backward direction, the value of $E(M)$ is required (as shown in Fig. 5.3) which can be easily calculated by $E(M) = H \oplus M$. As discussed in Section 5.1.1, the attacker tries to choose a state v such that in the forward direction it only depends on j and in the backward direction it only depends on i, i.e.,: [2]

$$\begin{aligned} \forall i, j: \quad & \overrightarrow{v_{i,j}} = \overrightarrow{v_{0,j}}, \\ \forall i, j: \quad & \overleftarrow{v_{i,j}} = \overleftarrow{v_{i,0}}. \end{aligned}$$

3. Let $\overrightarrow{v_{0,j}} = \overrightarrow{v_j}$ and $\overleftarrow{v_{i,0}} = \overleftarrow{v_i}$. Finally, the attacker checks if:

$$\exists i, j: \quad \overrightarrow{v_j} = \overleftarrow{v_i}.$$

If such an (i, j) pair exists, the corresponding $Q_{i,j}$ becomes the preimage candidate. If not, then the attacker picks up another group of states and repeats the whole procedure.

Thus, there are two main differences between sliced biclique cryptanalysis and the regular biclique cryptanalysis:

1. The internal difference in the sliced biclique attack is caused from the state itself instead of the key schedule algorithm in the regular biclique cryptanalysis.

[2] In the traditional biclique key recovery attack in [39], this special restriction on v is not required.

2. The biclique is limited to be sliced in the sliced-biclique cryptanalysis which results in a different partial matching process.

The flip side of this variant is that as compared to regular bicliques, the number of rounds attacked through sliced bicliques is smaller since the diffusion of Δ and ∇ - differences in the intermediate states is much quicker.

Complexity of the attack. The sliced biclique preimage attack comprises of 2 phases - biclique construction phase and MITM phase. Let the block cipher E consist of y rounds and the number of rounds covered in the biclique phase be x. This implies the number of rounds covered in the MITM phase is $y - x = z$. For each set of messages, in the biclique phase, since all $\Delta_i \neq \nabla_j$ and Δ_i trails are independent of ∇_j trails, the construction of biclique is simply reduced to computation of Δ_i and ∇_j trails independently which requires no more than 2.2^d computations of f, i.e.,

$$\text{Complexity of biclique phase} = 2^d \times \frac{x}{y} + 2^d \times \frac{x}{y} = 2^{d+1} \times \frac{x}{y}.$$

Similarly, in the MITM phase, the attacker needs to call each of $\overrightarrow{v_j}$ and $\overleftarrow{v_i}$ for 2^d times, i.e., a total of 2^{d+1} times. Let the number of rounds covered in the forward and backward direction be a and b respectively. Hence,

$$\text{Complexity of MITM phase} = 2^d \times \frac{a}{y} + 2^d \times \frac{b}{y} = 2^d \times \frac{a + b(= z)}{y} = 2^d \times \frac{y - x}{y}.$$

It is now easy to check that the overall complexity of sliced biclique preimage attack for one set of messages does not require more than 2^d full computations of E, i.e.,

$$\text{Total Complexity} = 2^{d+1} \times \frac{x}{y} + 2^d \times \frac{y - x}{y} = 2^d \times (1 + \frac{x}{y}) \approx 2^d \text{ since, } x \ll y.$$

If m bicliques are constructed, then the total cost is $m \times 2^d$.

Khovratovich used sliced biclique based preimage attack to produce collisions and applied it to round-reduced Skein hash function. He showed a 6-round collision attack on Skein-512 with a complexity of 2^{224} and 11-round collision attack with a complexity of 2^{251}. Similar attacks were also demonstrated on various rounds of Skein-256 with the highest number of rounds attacked being 9 with a complexity of 2^{124}. For further reading on sliced bicliques, one can refer to [99].

5.2 Type-2 Generalized Feistel Network

Feistel structure is one of the basic building blocks of block ciphers and block ciphers based constructions. A Feistel network divides the input message into two sub-blocks

(or two branches). Generalized Feistel Networks (GFN) are variants of Feistel networks with more than two branches, i.e., a k-branch GFN partitions the input message into k sub-blocks. They are sometimes favored over traditional Feistel scheme due to their high parallelism, simple design and suitability for low cost implementations. Many types of generalized Feistel schemes have been proposed and studied by researchers, as discussed in Section 2.3. Type-2 GFN in particular has seen wide adoption in well known block ciphers such as RC6 [145], SHAvite3 [27], CLEFIA [160], HIGHT [87] etc. Security analysis of generalized Feistel network [36,85,159,177] has been an active area of research for past many years. In fact, a comprehensive study done by Bogdanov et al. in [42] suggests that Type-2 GFN and its variants are more robust and secure against differential and linear cryptanalysis as compared to Type-1 GFN. Hence, we choose Type-2 GFN (shown in Fig. 5.4) as the basis for our study.

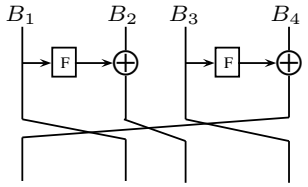


Figure 5.4: 4 branch, Type-2 Generalized Feistel Structure with right cyclic shift.

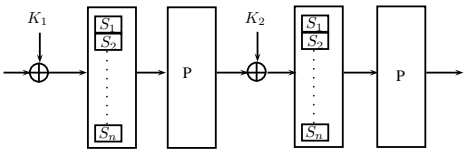


Figure 5.5: Double SP Function.

Type-2 GFN with double SP layer. It is generally desired that the round function F inside a generalized Feistel network should provide good diffusion and confusion properties. This is often realized by implementing F as a *substitution-permutation network* (nonlinear S-box transformation followed by linear permutation) as part of the round function design. There is a general belief that increasing the number of active S-boxes provides higher security margin against certain attacks. In [42], Bogdanov and Shibutani stressed on the importance of double SP (substitution-permutation) layers in the round function of Feistel networks as opposed to the single SP layer in the traditional design. They analyzed several designs such as single SP, double SP, SPS (substitution-permutation-substitution) and multiple SP layers and showed that double SP (shown in Fig. 5.5) layer achieves maximum security with respect to the proportion of active S-boxes in all S-boxes involved against differential and linear cryptanalysis. They especially compared double SP structure with single SP and showed that for Type-1 and Type-2 GFNs, proportion of linearly and differentially active S-boxes in double SP instantiations is 50% and 33% higher respectively as compared to the single SP instantiation. Their research advocated a possibility of designing more efficient and secure block cipher based constructions using double SP layer. In [148], Sasaki presented a 7-round distinguisher attack on 4-branch, type-2 GFN with double SP layer and a 6-round near collision attack on the compression function based on the

same structure. Kumar et al. [50] further improved the distinguishing attack on 4-branch, type-2 GFN with double SP layer by showing an 8-round distinguisher for the same. However, the form of truncated differential trails followed in [50, 148] cannot be used to launch collision attack when the above GFN structure is instantiated in compression function modes under known key settings. In our work, we construct an 8-round distinguisher which can be then used to generate collisions in 4-branch, type-2 GFN with double SP instantiated compression function with a complexity lesser than brute force (2^{64}) on 128-bit block input. We achieve so with the help of sliced biclique cryptanalysis technique.

5.2.1 Our Contributions

The main contributions of this chapter are as follows:

1. We apply sliced biclique technique to construct an 8-round distinguisher on 4-branch, Type-2 Generalized Feistel Network.

2. We use the distinguisher so constructed to demonstrate an 8-round collision attack on 4-branch, Type-2 GFN based compression functions (in MMO and MP mode) under known key settings with a complexity of 2^{56} (on 128-bit hash output). The attack can be directly translated to collision attacks on Matyas-Meyer-Oseas (MMO) and (Miyaguchi-Preneel) MP mode based hash functions and pseudo-collision attacks on Davies-Meyer (DM) mode based hash functions.

3. When the round function F is instantiated with double SP layer, we demonstrate the first 8-round collision attack on 4-branch, Type-2 GFN with double SP layer. This improves upon the 6 round near collision attack on the same structure by Sasaki in [148].

4. We investigate CLEFIA which is a real world-implementation of 4-branch, Type-2 GFN and demonstrate an 8-round collision attack on CLEFIA based hash function with a complexity of 2^{56}.

5.3 Notation

We consider 4-branch, type-2 generalized Feistel network for our attack. Following notation is followed in the rest of the sections.

$\mathbf{N}$	:	Input message size (in bits)
$\mathbf{n}$	:	Message word size (in bits) which is input to each branch, i.e., $n = N/4$
$\$\mathbf{R}$	:	Round R
$\$\mathbf{R_p}$	:	p^{th} word in round R. Each round has 4 words corresponding to 4 partitions of 4-branch GFN, i.e., $1 \leq p \leq 4$
$\$\mathbf{R_p^l}$	:	l^{th} block of word p in round R

5.4 Preliminaries

In this section, we give a brief overview of the key concepts used in our cryptanalysis technique to facilitate better understanding.

5.4.1 Type-2 Generalized Feistel Network (GFN) instantiated with double SP layer

One round of Type-2 GFN is shown in Fig. 5.6. A GFN with 4 branches divides the input B into four equal parts $[B_1, B_2, B_3, B_4]$. A round of Type-2 GFN with left cyclic shift outputs $[F(B_1) \oplus B_2, B_3, F(B_3) \oplus B_4, B_1]$ for some keyed nonlinear function F [42]. On the other hand, a round of Type-2 GFN with right cyclic shift outputs $[F(B_3) \oplus B_4, B_1, F(B_1) \oplus B_2, B_3]$ (as shown in Fig. 5.4) for round function F.

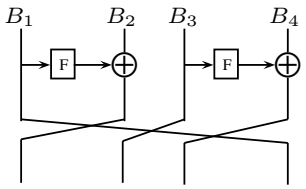


Figure 5.6: 4-branch, Type-2 Generalized Feistel Network with left cyclic shift.

The round transformation function F when defined by non-linear S-box layer followed by a permutation layer P exhibits substitution permutation structure. The permutation P is generally implemented using standard MDS matrix. If this SP structure is applied twice one after another then it is called double SP, as shown in Fig. 5.5. Few reasons favoring double SP over single SP function are as follows [42]:

- The second S-box in double SP provides larger number of active S boxes when differential and linear attacks are applied.
- The second permutation layer in double SP structure limits the differential effect, i.e., number of differential trails resulting in same differential is smaller as compared to round function having single permutation layer.

5.4.2 t-bit Partial Target Preimage Attack

Let the output of a hash function H with initial chaining value IV and message M be denoted by h, i.e., $h = \mathrm{H}(IV, M)$. In this attack, when the attacker is given t-bits of h, his aim is to find a message M' such that the hash output $h' = \mathrm{H}(IV, M')$ matches these t-bits of h and at the same positions. The other bits of hash output $\mathrm{H}(IV, M')$ are generated randomly.

5.5 Distinguishing Attack on 4-branch, Type-2 GFN based Permutation

In this section, we present an 8-round distinguisher on permutation E_k (where k is the key) which is an 8-round, 4-branch, Type-2 Generalized Feistel Network using sliced biclique cryptanalysis. We assume that the S-box layer has good differential property and the P-layer implements standard MDS matrix. [3] We also assume that the key k (that is IV in the overlying hash function construction) is a fixed constant. The distinguishing property used by the distinguisher is as follows:

Distinguishing Property. Let E_k be a block cipher with message size $N = 128$-bits. The aim of the adversary is to collect 2^{16} (plaintext, ciphertext) pairs such that the XOR of the lower 16 bits of the third word in the plaintext and the lower 16 bits of the third word in the ciphertext (where each word is of size 32-bits) is always a 16-bit constant value chosen by the attacker, i.e.,

$$(\text{plaintext})_3^2 \oplus (\text{ciphertext})_3^2 = \text{constant} \tag{5.3}$$

where, |constant| = 16-bits. [4]

In case of random permutation. When E_k is a random permutation, the probability that any (plaintext, ciphertext) pair satisfies the desired property (as mentioned in Equation 5.3) is 2^{-16}. This means that the expected time complexity to generate one such (plaintext, ciphertext) pair is 2^{16}. Hence, expected time complexity to generate 2^{16} such (plaintext, ciphertext) pairs is 2^{32}.

In case of E instantiated with 4-branch, Type-2 GFN. For the illustration of our attack, we consider N =128-bit and n = 32-bit each. The attacker first chooses a random base value $Q_{0,0}$ (as discussed in Section 5.1). Let $\Delta_i = (\bar{0}\bar{0} \mid i\bar{0} \mid \bar{0}\bar{0} \mid \bar{0}\bar{0})$ and $\nabla_j = (\bar{0}\bar{0} \mid \bar{0}j \mid \bar{0}\bar{0} \mid \bar{0}\bar{0})$ where $(0 \leq i, j \leq 2^{16} - 1)$ be the Δ and ∇ differences injected in Round 4. Here, each $\bar{0}$ represents 0^{16}. The propagation of Δ_i trail (marked as '|' in green) and ∇_j trail (marked as '-' in red) is shown in Fig. 5.7 and Fig. 5.8 respectively. In these figures, the four words shown in each round are the corresponding inputs to four branches at each round. In ∇_j trail, the attacker first injects the given j difference in $\$4_2^2$ word only. As the ∇_j trail propagates as shown in Fig. 5.8, $\$4_1$ and $\$4_4$ words are subsequently affected. The dimension of this biclique is d=16.

It is easy to check that Δ_i and ∇_j trails are independent and do not share any non-linear components (shown in Fig. 5.9) between them in rounds 4 and 5. Thus, a

[3] In this line of work, implementation of P-layer as a standard MDS matrix having optimal branch number is believed to be a good design choice [42, 96, 148, 153]

[4] Here $(plaintext)_3^2$ denotes second block of third word of plaintext as described in Section 5.3. The term $(ciphertext)_3^2$ can be understood similarly.

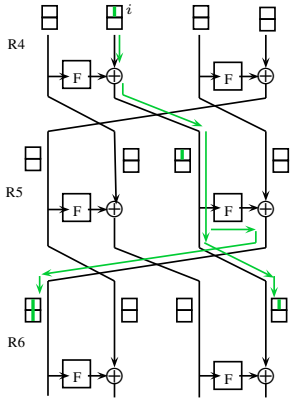


Figure 5.7: Δ_i difference injection in Round 4 and its propagation.

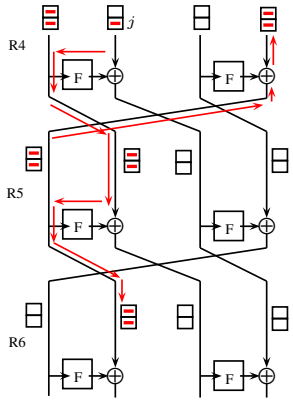


Figure 5.8: ∇_j difference injection in Round 4 and its propagation.

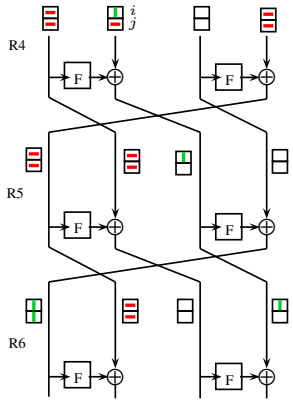


Figure 5.9: 2-round biclique placed in Round 4 - 5.

2-round biclique (consisting of $2^{2d} = 2^{32}$ messages) is formed where the biclique covers rounds \$4 and \$5. Now the aim of the attacker is to find a matching variable v which only depends on Δ_i trail in one direction and ∇_j trail in the other direction (as discussed in Section 5.1). Hence, from round 6 only ∇_j trail is propagated in the forward

direction and from round 3 only Δ_i trail is propagated in the backward direction (as shown in Fig. 5.10). At the end of 8^{th} round it can be seen that $\$1_3^2$ (marked in yellow in Fig. 5.10) in the backward direction is not affected by the Δ_i trail (i.e., will be affected by ∇_j trail only) and $\$8_3^2$ (marked in yellow in Fig. 5.10) in the forward direction remains unaffected by the ∇_j trail (i.e., will be affected by Δ_i trail only). Through feed forward operation, 16 bits of $\$1_3^2$ can then be matched with 16 bits of $\$8_3^2$. Hence, in this attack we choose $\$8_3^2$ to be our matching variable v and $\mid v \mid = 16$ which is denoted by t.

Once the matching variable v is obtained, as mentioned above, through our biclique attack, $2^{2d} = 2^{32}$ (plaintext, ciphertext) pairs are generated in a set. Out of these 2^{2d} (plaintext, ciphertext) pairs, there exists $2^{2d-t} = 2^{16}$ (plaintext, ciphertext) pairs which match on matching variable v. In other words, if we XOR the lower 16 bits of the third word in the plaintext and the lower 16 bits of the third word in the ciphertext (i.e., at positions $\$1_3^2$ and $\$8_3^2$ respectively), Equation 1 will always be satisfied. These 2^{16} (plaintext, ciphertext) pairs will be generated with a computational complexity of $2^d = 2^{16}$ (as discussed in Section 5.1) which is lower than the computational complexity of 2^{32} in case of random permutation. Hence, a valid distinguisher for E when instantiated with 4-branch, Type-2 GFN is constructed.

Similarly, our attack can be applied to messages of other sizes as well. In Table 5.1, we report the complexity values for our distinguisher attack on message inputs of different size.

Table 5.1: Complexity of our distinguishing attack on message inputs of different size. N represents the input message size in bits and #(P-C) pairs represent the number of plaintext-ciphertext pairs needed for our attack. The number of plaintext-ciphertext pairs depends on the size of matching variable v.

N	n	#(P-C) pairs	Complexity of our attack	Complexity of random permutation
64	16	2^8	2^8	2^{16}
256	64	2^{32}	2^{32}	2^{64}
512	128	2^{64}	2^{64}	2^{128}

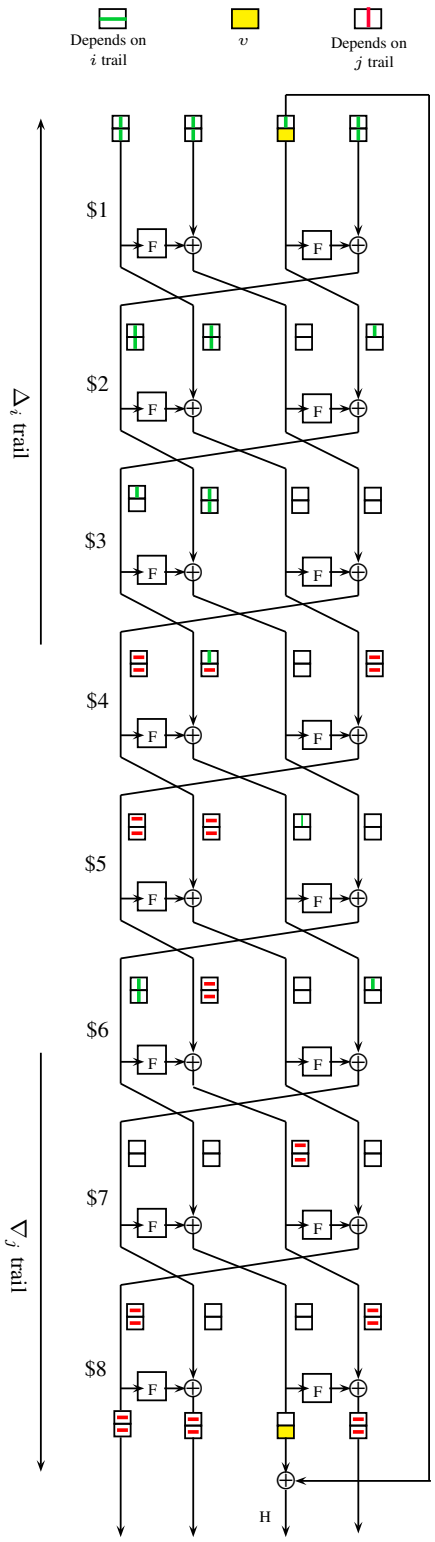


Figure 5.10: Matching in 8 rounds of 4-branch Type-2 GFN with right cyclic shift.

5.6 Collision Attack on 4-branch, Type-2 GFN based compression function

The distinguisher constructed in the previous section can be used to launch collision attack on 4-branch, Type-2 GFN based compression function as described below. We assume the compression function to be in MMO mode and the output is assumed to be of $N = 128$-bits.

- The attacker first chooses a t-bit constant of his choice.
- In the above attack, the attacker then finds a matching variable v, where $\mid v \mid \leq t$. In our attack, $\mid v \mid = t = 16$ bits.
- There are $2^{2d} = 2^{32}$ messages in a biclique set. Out of these 2^{2d} messages, only 2^{2d-t} messages will match on v. This means that out of 2^{32} messages only 2^{16} messages will survive the MITM phase.
- In other words, it can be said that the attacker has generated 2^{16} t-bit partial target preimages with these t-bits equal to an arbitrarily chosen constant selected in first step.
- These 2^{16} t-bit partial target preimages collide on $t = 16$ bits. Hence, if the attacker generates $2^{(N-t)/2}$ such preimages which collide on t-bits, there exists a colliding pair with high probability which collide on the remaining $N - t$ bits as well. Thus, the attacker will generate $2^{(128-16)/2} = 2^{56}$ such t-bit partial target preimages to obtain a collision on complete hash output H with high probability.
- Now, one sliced biclique generates 2^{16} t-bit partial target preimages. Hence, to generate 2^{56} such preimages, the attacker needs to construct $2^{56-16} = 2^{40}$ sliced bicliques (or, $2^{(N-t)/2-(2d-t)}$ bicliques where, $2^{(N-t)/2} = 2^{56}$ and $2^{(2d-t)} = 2^{16}$).

Complexity of the collision attack. Since the computational complexity of performing sliced biclique attack once is $2^d = 2^{16}$ (as discussed in Section 5.1), hence computational complexity of running sliced biclique attack 2^{40} times is $2^{40} \times 2^{16} = 2^{56}$. Therefore, given IV, the complexity to find a pair of messages (M, M') such that CF(IV, M) = CF(IV, M'), when CF (i.e., compression function) is instantiated with 8-rounds of 4-branch type-2 GFN is 2^{56} ($< 2^{64}$ brute-force attack). The compression function output is of 128-bits size. In general, the complexity of the attack is given by the following formula:

$$\text{Complexity} = 2^{\frac{(N-t)}{2}-(2d-t)} \times 2^d.$$

For the purpose of illustration, we show the cost of our attack for various other message sizes in Table 5.2.

Table 5.2: Complexity of our 8-round collision attack on message inputs of different size. N represents the input message size in bits, n represents the branch word size in bits and t represents the size of matching variable v in bits. In our attack $d = t$ always.

N	n	t	Complexity of our attack	Brute force complexity
64	16	8	2^{28}	2^{32}
128	32	16	2^{56}	2^{64}
256	64	32	2^{112}	2^{128}
512	128	64	2^{224}	2^{256}

Since we need to store all the partial preimages to find a colliding pair, memory required is of the order of 2^{56} (for 128-bit output). However, it is mentioned in [99] that memoryless equivalents of these attacks do exist.

Collision Attack on 4-branch Type-2 GFN with Double SP layer. The above attack technique is generic and independent of the internal F-function structure. Hence, if we instantiate the round function F with double SP-layer, the above attack can be directly translated to 8-round collision attack on 4-branch, Type-2 GFN with double SP layer based compression function with a complexity of 2^{56}. This improves the 6-round near collision attack on the same structure shown by Sasaki in [148]. In Table 5.3 we compare our result with the previous cryptanalysis results on 4-branch, Type-2 GFN with double SP layer.

Table 5.3: Comparison of our results with previous cryptanalytic results on 4-branch, Type-2 GFN with double SP layer.

Rounds	Attack Type	Reference
6	Near Collisions	[148]
7	Distinguishing	[148]
8	Distinguishing	[50]
8	Distinguishing	Section 5.5
8	Full Collisions	Section 5.6

As discussed above, since the attack technique is generic, presence of multiple SP layers in the round function F does not provide any extra resistance against sliced biclique attack as compared to double SP layer. In fact, in our collision attack neither

the attack complexity nor the the number of rounds attacked change if double SP layer is replaced by multiple SP layers. This is in contrast to attacks such as rebound attacks [118], where the number of SP layers inside the round function F influence the number of rounds attacked [50, 68, 96, 148, 153] in Generalized Feistel Networks.

5.7 Collision Attack on Hash Functions

In this attack, given the IV, the aim of the attacker is to find a pair of messages (M, M') such that $H(M) = H(M')$. To do so, the attacker first finds two messages M_1 and M_1' which collide to same hash value h_1 using collision attack technique described in Section 5.6 with a complexity of 2^{56}. Now he concatenates any message M_2 with M_1 and M_1' (as shown in Fig. 5.11) such that $H(M_1 \| M_2) = H(M_1' \| M_2)$. Message M_2 can also be chosen such that it satisfies padding restrictions (where length of input message is appended at the end) if required. In this way, collision attack can be carried out on 4-branch, Type-2 GFN with double SP layer based hash function with a complexity of 2^{56}. Since we assume known key settings (i.e., key part to the underlying block cipher is known to the attacker), hence this attack can be used to generate collisions in MP and MMO based hash functions but pseudo collisions in DM based hash functions.

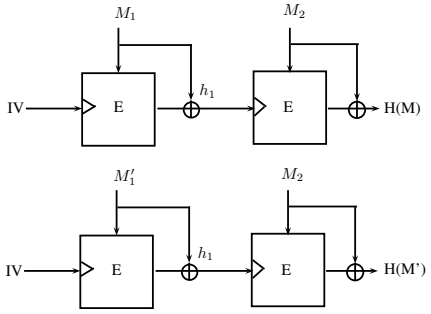


Figure 5.11: Collision Attack.

5.8 8-Round Collision Attack on CLEFIA based Compression Function

In this section, we investigate CLEFIA which is a real world-implementation of 4-branch, Type-2 GFN. In the attacks discussed in Section 5.5 and Section 5.6, we considered 4-branch, Type-2 GFN with double SP layer where right cyclic shift is applied on the message sub-blocks at the end of each round. This was done to facilitate direct comparison with previous results [50, 148] on the same structure. However in [180], Type-2 GFN's have been defined with left cyclic shift and is followed in all the practical implementations of Type-2 GFN structure - e.g., RC6 [145], CLEFIA [160],

HIGHT [87] etc. Since, left and right cyclic shifts are equivalent hence, similar attack procedure (as discussed in Section 5.6) can be applied on CLEFIA as well but with different Δ_i and ∇_j trails. CLEFIA is a 128-bit block cipher and supports three key lengths - 128-bit, 192-bit and 256-bit. The number of rounds correspondingly are 18, 22 and 26. In this section, we examine CLEFIA with 128-bit key size. [5] WK_0 and WK_1 represent the whitening keys at the start of the cipher. Each round has two 32-bit round keys RK_{2i-2} and RK_{2i-1} (where, $1 \leq i \leq 18$).

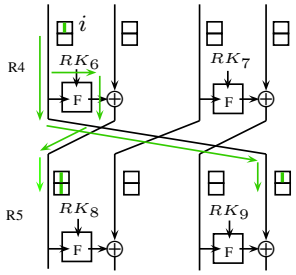


Figure 5.12: Δ_i difference injection in Round 4 and its propagation

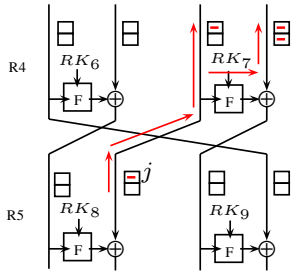


Figure 5.13: ∇_j difference injection in Round 5 and its propagation

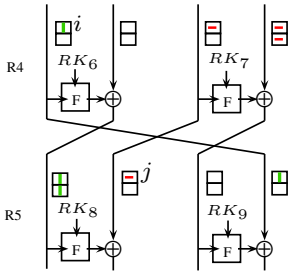


Figure 5.14: 1-round biclique placed in Round 4

In this attack, let $\Delta_i = (i\bar{0} \mid \bar{0}\bar{0} \mid \bar{0}\bar{0} \mid \bar{0}\bar{0})$ be the Δ difference injected in Round 4 and $\nabla_j = (\bar{0}\bar{0} \mid j\bar{0} \mid \bar{0}\bar{0} \mid \bar{0}\bar{0})$ be the ∇ difference injected in Round 5, where the variables i and j can take all 16-bit values, i.e., $0 \leq i, j \leq 2^{16} - 1$. Here, each $\bar{0}$ represents 0^{16}. The attacker first chooses a random base value $Q_{0,0}$ and then injects the Δ_i and ∇_j differences accordingly. The propagation of Δ_i trail (marked as '|' in green) and ∇_j trail (marked as '-' in red) is shown in Fig. 5.12 and Fig. 5.13 respectively. The dimension of this biclique is d=16. It is easy to check that Δ_i and ∇_j trails are independent and do not share any non-linear components (shown in Fig. 5.14) between them in round 4. Thus a 1-round biclique (consisting of $2^{2d} = 2^{32}$ messages) is formed in \$4 round.

From round 5, only ∇_j trail is propagated in the forward direction and from round 3 only Δ_i trail is propagated in the backward direction (as shown in Fig. 5.15). At the end of 8^{th} round it can be seen that $\$1_3^2$ (marked in yellow in Fig. 5.15) in the backward direction is not affected by Δ_i trail and $\$8_3^2$ (marked in yellow in Fig. 5.15) in the forward direction remains unaffected by ∇_j trail. Through feed forward operation, 16 bits of $\$1_3^2$ can then be matched with 16 bits of $\$8_3^2$. Hence, in this attack we choose $\$8_3^2$ to be our matching variable v. The steps of collision attack for CLEFIA are exactly the same as discussed in Section 5.6 and Section 5.7. Therefore, we can generate collisions in 8-rounds of CLEFIA based hash function with a complexity of 2^{56}.

[5] The attack works on other key sizes as well since key is constant under known key settings.

5.9 Conclusions

In this chapter, we discussed another variant of biclique cryptanalysis termed as sliced biclique cryptanalysis technique We applied the sliced biclique technique to show collision attack on 8-rounds of 4-branch, type-2 GFN. When it is instantiated with double SP layer, we presented the first 8-round collision on 4-branch, type-2 GFN with double SP layer. It would be interesting to apply sliced biclique technique to attack other potential targets. One possible extension can be to apply this attack technique on 2-branch, Type-2 GFN such as Shavite-3 etc.

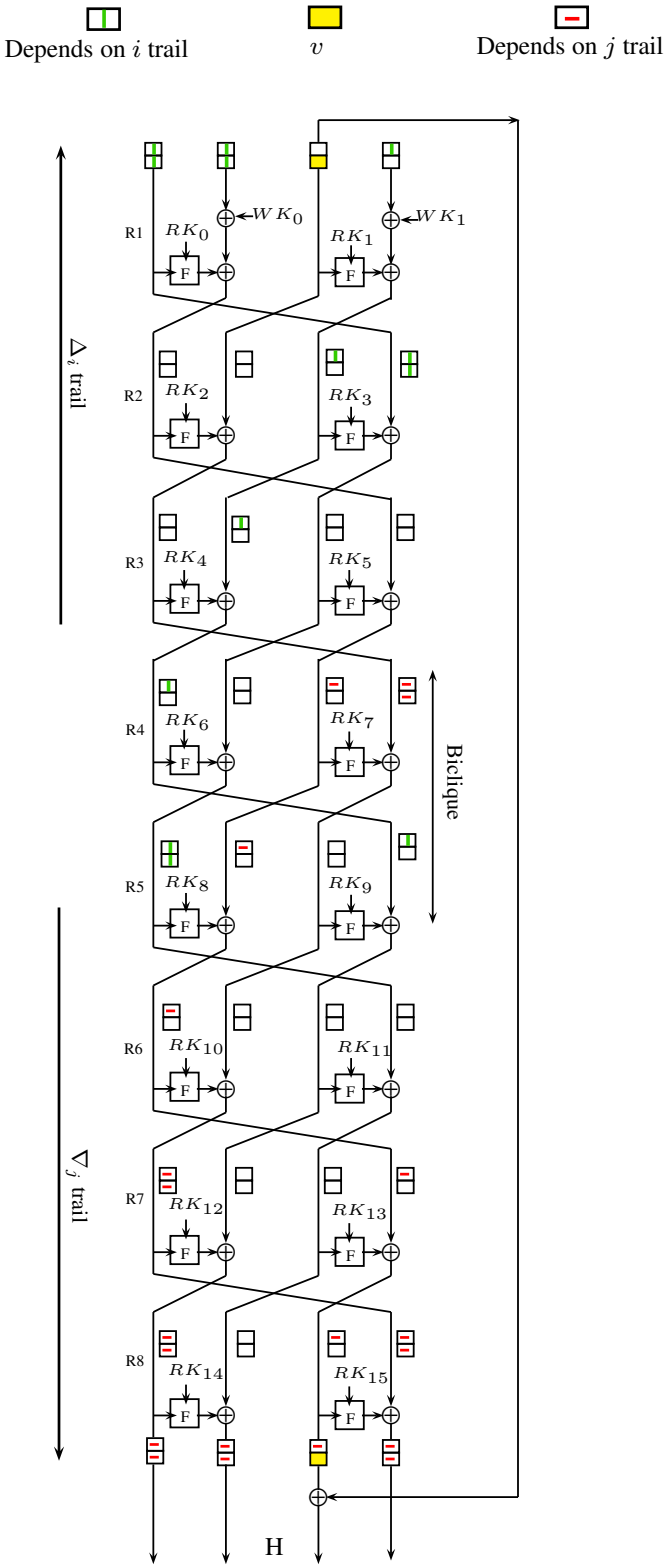


Figure 5.15: Matching in 8 rounds of CLEFIA

Chapter 6

Multiset based Meet-in-the-Middle Attack on ARIA-192 and ARIA-256

In this chapter, we again switch back to block cipher cryptanalysis and discuss yet another variant of meet-in-the-middle attacks termed as *multiset attacks*. For the attacks under single key model, the best attack for AES in terms of highest number of rounds cryptanalyzed, is attributed to biclique based key recovery attacks proposed by Bogdanov et al. [39]. However, the flip side of this attack is its very high time complexity and just marginal gain over brute-force. After the biclique attacks, the next best attack on AES in the single key model (in terms of number of rounds cryptanalyzed) is the meet-in-the middle based multiset attack first proposed by Dunkelman et al. in [71].

For AES block cipher, the meet-in-the-middle attack model was first introduced by Demirci et al. in FSE'08 [60], where they improved the collision attack proposed by Gilbert and Minier [79]. Their attack involved constructing a set of functions which mapped one active byte in the first round to another active byte after 4-rounds of AES. This set of functions were dependent on 25 parameters only and could be described using a table of 2^{25} ordered 256-byte sequence of entries. This table was precomputed and stored, thus allowing building a 4-round distinguisher and attacking upto 8 rounds of AES. Combined with data/time/memory tradeoff, this attack was applied to analyze 7-round AES-192 and 8-round AES-256. Demirci et al.'s attack on AES was improved by Dunkelman et al. in [71] who proposed multiset attack which replaced the idea of storing 256 ordered byte sequences with 256 unordered byte sequences (with multiplicity). This reduced both memory and time complexity of MITM attack on AES by reducing the number of parameters to 23. They also introduced the novel idea of *differential enumeration technique* which employed a truncated differential characteristic to significantly lower the number of parameters required to construct the multiset from 23 to just 16, thus further decreasing the attack complexities on AES. Their cryptanalysis resulted in a 7-round attack on AES-128 and 8-round attack on AES-192 and AES-256. Derbez at al. in [61] improved Dunkelman et al.'s attack by refining the differential

Table 6.1: Comparison of meet-in-the-middle based cryptanalytic attacks on AES.[1]

Target	Rounds	Time	Data	Memory	Reference
AES-128	7	2^{116}	2^{116}	2^{116}	[71]
	7	2^{99}	2^{99}	2^{96}	[61]
AES-192	8	2^{208}	2^{32}	2^{206}	[60]
	8	2^{172}	2^{113}	2^{129}	[71]
	8	2^{172}	2^{107}	2^{96}	[61]
AES-256	8	2^{208}	2^{32}	2^{206}	[60]
	8	2^{196}	2^{113}	2^{129}	[71]
	8	2^{196}	2^{107}	2^{96}	[61]
	9	2^{203}	2^{120}	2^{203}	[61]

enumeration technique. By using rebound-like techniques [118], they showed that the number of reachable multisets are much lower than those counted in Dunkelman et al.'s attack. The number of possible mutisets now depended on just 10 parameters giving the best attack on AES-128. This improvement allowed mounting of comparatively more efficient attacks on AES and also enabled extension of number of rounds attacked for AES-256. Table 6.1, summarizes the cryptanalytic results of these attacks.

Though the results of this line of work are quite interesting, yet they have not been explored further. In this chapter, we investigate the Korean encryption standard ARIA and study the effectiveness of multiset attacks on it. The rest of the chapter is organized as follows: In Section 6.1, we describe ARIA and present the current cryptanalytic results existing on it. We also discuss some of the challenges existing in these results. This is followed by Section 6.2 where we mention the notation followed throughout the chapter. In Section 6.3, we give details of our distinguisher on 4-rounds of ARIA. In Section 6.4, we present our 7-round attack followed by Section 6.5, where we demonstrate our 8-round attack on ARIA and show the recovery of the secret key. Finally in Section 6.6, we summarize and conclude our chapter. The original contribution of this thesis is from Section 6.2 to Section 6.5.

[1]The results mentioned here for AES-192 and AES-256 are not the best. The best attack results on these AES variants will be reported in the next chapter.

6.1 Block Cipher ARIA

The block cipher ARIA, proposed by Kwon et al. [114], is a 128-bit block cipher that adopts substitution-permutation network (SPN) structure similar to AES [57] and supports three key sizes: 128-bit, 192-bit and 256-bit. The first version of ARIA (version 0.8) had 10/12/14 rounds for key sizes of 128/192/256 respectively and only two kinds of S-boxes were employed in its substitution layer [46, 178]. Later ARIA version 0.9 was announced at ICISC 2003 [114] in which four kinds of S-boxes were used. This was later upgraded to ARIA version 1.0 [77], the current version, which was standardized by Korean Agency for Technology and Standards (KATS) - the government standards organization of South Korea as the 128-bit block encryption algorithm (KS X 1213) in December, 2004. In this version, the number of rounds were increased to 12/14/16 and some modifications in the key scheduling algorithm were made. ARIA has also been adopted by several standard protocols such as IETF (RFC 5794 [113]), SSL/TLS (RFC 6209 [104]) and PKCS #11 [115] thereafter.

For block cipher ARIA, the 128-bit internal state and the key state are treated as a byte matrix of 4×4 size, where the bytes are numbered from 0 to 15 column wise (as shown in Fig. 6.1). Each round consists of 3 basic operations (as shown in Fig. 6.2):

0	4	8	12
1	5	9	13
2	6	10	14
3	7	11	15

Figure 6.1: Byte numbering in a state of ARIA

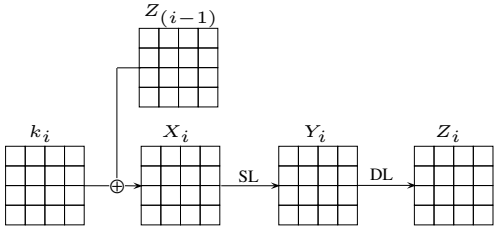


Figure 6.2: i^{th} round of ARIA.

1. *Add Round Key (ARK)* - This step involves an exclusive-or operation with the round subkey. The key schedule of ARIA consists of two phases:
   - A nonlinear expansion phase, in which the 128-bit, 192-bit or 256-bit master key is expanded into four 128-bit words W_0, W_1, W_2, W_3 by using a 3-round 256-bit Feistel cipher.
   - A linear key schedule phase in which the subkeys are generated via simple XORs and rotation of W_0, W_1, W_2, W_3 each.
2. *Substitution Layer (SL)* - It uses four types of 8-bit S-boxes S_1, S_2 and their inverses S_1^{-1} and S_2^{-1}. Each S-Box is defined to be an affine transformation of the inversion function over GF(2^8). The S_1 S-box is the same as that used in AES. ARIA has two types of substitution layers for even and odd rounds respectively.

In each odd round, the substitution layer is (LS, LS, LS, LS) where LS = (S_1, S_2, S_1^{-1}, S_2^{-1}) operates one column and in each even round, the substitution layer is (LS^{-1}, LS^{-1}, LS^{-1}, LS^{-1}) where LS^{-1} = (S_1^{-1}, S_2^{-1}, S_1, S_2) operates on one column as well.

3. *Diffusion Layer (DL)* - This layer consists of a 16 × 16 involutional binary matrix with branch number 8. Given an input state y and output state z, the diffusion layer is defined as:

$$
\begin{aligned}
z[0] &= y[3] \oplus y[4] \oplus y[6] \oplus y[8] \oplus y[9] \oplus y[13] \oplus y[14] \\
z[1] &= y[2] \oplus y[5] \oplus y[7] \oplus y[8] \oplus y[9] \oplus y[12] \oplus y[15] \\
z[2] &= y[1] \oplus y[4] \oplus y[6] \oplus y[10] \oplus y[11] \oplus y[12] \oplus y[15] \\
z[3] &= y[0] \oplus y[5] \oplus y[7] \oplus y[10] \oplus y[11] \oplus y[13] \oplus y[14] \\
z[4] &= y[0] \oplus y[2] \oplus y[5] \oplus y[8] \oplus y[11] \oplus y[14] \oplus y[15] \\
z[5] &= y[1] \oplus y[3] \oplus y[4] \oplus y[9] \oplus y[10] \oplus y[14] \oplus y[15] \\
z[6] &= y[0] \oplus y[2] \oplus y[7] \oplus y[9] \oplus y[10] \oplus y[12] \oplus y[13] \\
z[7] &= y[1] \oplus y[3] \oplus y[6] \oplus y[8] \oplus y[11] \oplus y[12] \oplus y[13] \\
z[8] &= y[0] \oplus y[1] \oplus y[4] \oplus y[7] \oplus y[10] \oplus y[13] \oplus y[15] \\
z[9] &= y[0] \oplus y[1] \oplus y[5] \oplus y[6] \oplus y[11] \oplus y[12] \oplus y[14] \\
z[10] &= y[2] \oplus y[3] \oplus y[5] \oplus y[6] \oplus y[8] \oplus y[13] \oplus y[15] \\
z[11] &= y[2] \oplus y[3] \oplus y[4] \oplus y[7] \oplus y[9] \oplus y[12] \oplus y[14] \\
z[12] &= y[1] \oplus y[2] \oplus y[6] \oplus y[7] \oplus y[9] \oplus y[11] \oplus y[12] \\
z[13] &= y[0] \oplus y[3] \oplus y[6] \oplus y[7] \oplus y[8] \oplus y[10] \oplus y[13] \\
z[14] &= y[0] \oplus y[3] \oplus y[4] \oplus y[5] \oplus y[9] \oplus y[11] \oplus y[14] \\
z[15] &= y[1] \oplus y[2] \oplus y[4] \oplus y[5] \oplus y[8] \oplus y[10] \oplus y[15]
\end{aligned}
$$

In the last round, diffusion layer is replaced by key xoring to generate the ciphertext.

Key Schedule Algorithm of ARIA. The key schedule algorithm of ARIA [113] is divided into two phases - *Initialization phase* and *Round Key Generation phase*. In the initialization phase, first the master key K is expanded into a 256-bit value as follows:

$$KL \mid\mid KR = K \mid\mid 0...0$$

where, $\mid KL \mid = \mid KR \mid$ = 128-bits and number of zeroes padded to K equals 128, 64 and 0 for $\mid K \mid$ equal to 128, 192 and 256-bits respectively.

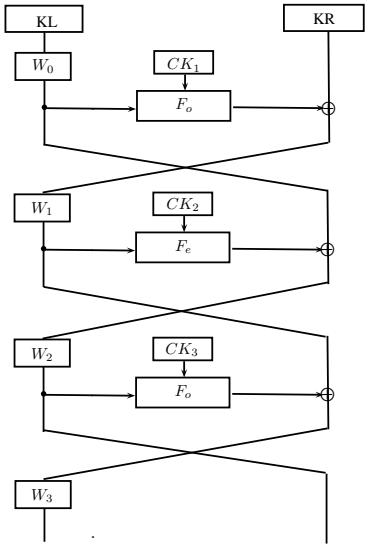


Figure 6.3: Generation of four 128-bit words, W_0, W_1, W_2 and W_3 through a 3-round Feistel.

Then, four 128-bit values W_0, W_1, W_2 and W_3 are set as (shown in Fig. 6.3):

$$W_0 = KL \tag{6.1}$$
$$W_1 = F_o(W_0, CK_1) \oplus KR \tag{6.2}$$
$$W_2 = F_e(W_1, CK_2) \oplus W_0 \tag{6.3}$$
$$W_3 = F_o(W_2, CK_3) \oplus W_1 \tag{6.4}$$

where, F_o and F_e are ARIA odd and even round functions and CK_1, CK_2 and CK_3 are pre-defined constants. In the round key generation phase, the following round subkeys are generated as follows:

$$K_1 = W_0 \oplus (W_1 >>> 19) \tag{6.5}$$
$$K_2 = W_1 \oplus (W_2 >>> 19) \tag{6.6}$$
$$K_3 = W_2 \oplus (W_3 >>> 31) \tag{6.7}$$
$$K_4 = (W_0 >>> 19) \oplus W_3 \tag{6.8}$$
$$K_5 = W_0 \oplus (W_1 >>> 31) \tag{6.9}$$
$$K_6 = W_1 \oplus (W_2 >>> 31) \tag{6.10}$$
$$K_7 = W_2 \oplus (W_3 >>> 31) \tag{6.11}$$
$$K_8 = (W_0 >>> 31) \oplus W_3 \tag{6.12}$$
$$K_9 = W_0 \oplus (W1 <<< 61) \tag{6.13}$$

The key schedule algorithm of ARIA has been designed such that it does not allow the recovery of the actual secret key given any full round key. For further details, one can refer [113].

Previous cryptanalytic results on ARIA. ARIA block cipher has been subjected to reasonable cryptanalysis in the past 12 years since its advent. In [30], Biryukov et al. analyzed the first version (version 0.8) of ARIA and presented several attacks such as truncated differential cryptanalysis, dedicated linear attack, square attack etc. against reduced round variants of ARIA. In the official specification document of ARIA [114], the ARIA developers analyzed the security of ARIA against many classical cryptanalyses such as differential and linear cryptanalysis, impossible and higher order differential cryptanalysis, slide attack, interpolation attack etc. and claimed that ARIA has better resistance against these attacks as compared to AES. In [176], Wu et al. presented a 6-round impossible differential attack against ARIA which was improved in terms of attack complexities by Li et al. in [123]. In [124], Li et al. presented a 6-round integral attack on ARIA followed by Fleischmann et al. [76] who demonstrated boomerang attacks on 5 and 6 rounds of ARIA. Du et al. in [69], extended the number of rounds by one and demonstrated a 7-round impossible differential attack on ARIA-256. In [168], Tang et al., applied meet-in-the-middle (MITM) attack to break 7 and 8-rounds of ARIA-192/256 and is the best attack on ARIA so far. In Table 6.2, we summarize all the existing attacks on ARIA version 1.0.

The security of ARIA has not been analyzed after Fleischmann et al.'s attack in Indocrypt 2010 [76]. This motivated us to investigate the effectiveness of multiset attack on ARIA. In this work, we improve the attack complexities of the 7 and 8-rounds of ARIA-192/256. We construct a new 4-round distinguisher for ARIA. Using this distinguisher, our subkey recovery attacks significantly reduce the data/time/memory complexities of 7-round attack on ARIA-192/256 and time and memory complexities of 8-round attack on ARIA-256 as compared to the previous best attack complexities reported in [168]. The key schedule algorithm of ARIA does not allow recovery of master key from a subkey unlike AES [57]. This is likely the reason why none of the previous attacks have shown the secret key retrieval on any ARIA variant. However, depending upon the key expansion of ARIA, recovery of specific subkeys allows extracting the secret key. In our 7 and 8-round attack on ARIA-192/256, we exploit this key scheduling property to demonstrate the secret key recovery in ARIA. To the best of our knowledge, we are the first to demonstrate the actual secret key recovery attack on ARIA.

6.1.1 Our Contribution.

The main contributions of this chapter are as follows:

- We present the best 7-round key recovery attack on ARIA 192/256 and 8-round attack on ARIA-256.
- We apply multiset attack to construct a new 4-round distinguisher on ARIA-192 and ARIA-256.

Table 6.2: Comparison of cryptanalytic attacks on ARIA version 1.0. The entries are arranged in terms of decreasing time complexities for each category of attacked rounds.

Rounds attacked	Attack type	Time complexity	Data complexity	Memory complexity	Reference
5	Boomerang Attack	2^{110}	2^{109}	2^{57}	[76]
	Integral Attack	$2^{76.7}$	$2^{27.5}$	$2^{27.5}$	[124]
	Impossible Differential	$2^{71.6}$	$2^{71.3}$	2^{72}	[123]
	Meet-in-the-middle	$2^{65.4}$	2^{25}	$2^{122.5}$	[168]
6	Integral Attack	$2^{172.4}$	$2^{124.4}$	$2^{124.4}$	[124]
	Meet-in-the-middle	$2^{121.5}$	2^{56}	$2^{122.5}$	[168]
	Impossible Differential	2^{112}	2^{121}	2^{121}	[176]
	Boomerang Attack	2^{108}	2^{128}	2^{56}	[76]
	Impossible Differential	$2^{104.5}$	$2^{120.5}$	2^{121}	[123]
7	Impossible Differential	2^{238}	2^{125}	2^{125}	[69]
	Boomerang Attack	2^{236}	2^{128}	2^{184}	[76]
	Meet-in-the-middle	$2^{185.3}$	2^{120}	2^{187}	[168]
	Meet-in-the-middle (ARIA-192)	$2^{135.1}$	2^{113}	2^{130}	Sec. 6.4
	Meet-in-the-middle (ARIA-256)	$2^{136.1}$	2^{115}	2^{130}	Sec. 6.4
8	Meet-in-the-middle (ARIA-256)	$2^{251.6}$	2^{56}	2^{252}	[168]
	Meet-in-the-middle (ARIA-256)	$2^{245.9}$	2^{113}	2^{138}	Sec. 6.5

- Our 7-round attack on ARIA-192 has data/time/memory complexity of 2^{113}, $2^{135.1}$ and 2^{130} respectively.
- Our 7-round attack on ARIA-256 has data/time/memory complexity of 2^{115}, 2^{136} and 2^{130} respectively.
- Our 8-round attack on ARIA-192/256 has data/time/memory complexity of 2^{113}, $2^{245.6}$ and 2^{138} respectively.
- We demonstrate the first master key recovery on our attacks on ARIA-192/256.

Our results are summarized in Table 6.2.

6.2 Preliminaries

In this section, we mention the key notations and definitions used in our cryptanalysis technique to facilitate better understanding.

6.2.1 Notations and Definitions

The following notations are followed in this chapter.

$\mathbf{P}$	:	Plaintext
$\mathbf{C}$	:	Ciphertext
$\mathbf{k_i}$	:	Subkey of round i
$\mathbf{k_i^*}$	:	$DL^{-1}(k_i)$, where, DL^{-1} is the inverse diffusion layer
$\mathbf{X_i}$	:	State obtained after ARK in round i
$\mathbf{Y_i}$	:	State obtained after SL in round i
$\mathbf{Z_i}$	:	State obtained after DL in round i
$\mathbf{\Delta_x}$	:	Difference in a state s
$\mathbf{s_i[m]}$	:	m^{th} byte of a state s in round i, where $0 \leq m \leq 15$
$\mathbf{s_i[p,\ldots,r]}$	:	p^{th} byte, $\ldots$, r^{th} byte of state s in round i, where $0 \leq p, r \leq 15$

In our attacks, rounds are numbered from 1 to R, where R = 7 or 8. A *full* round consists of all the three round operations, i.e., ARK, SL and DL whereas a *half* round denotes a round in which the DL operation is omitted.

We utilize the following definitions for our attacks.

Definition 1 (δ-list). We define the δ-list as an ordered list of 256 16-byte distinct elements that are equal in 15 bytes. Each of the 15 equal bytes is called as passive byte whereas the one byte that takes all possible 256 values is called the active byte [57]. We denote the δ-list as (x^0, x^1, x^2, $\ldots$, x^{255}) where x^j indicates the j^{th} 128-bit member of the δ-list. As mentioned in the notations section, x_i^j [m] represents the m^{th} byte of x^j in round i.

Definition 2 (Multiset). A multiset is a set of elements in which multiple instances of the same element can appear. A multiset of 256 bytes, where each byte can take any one of the 256 possible values, can have $\binom{2^8+2^8-1}{2^8} \approx 2^{506.17}$ different values.

Two crucial properties that will be used in our attacks are as follows:

Property 1. For a given input-output difference (denoted as (ΔY, ΔZ)) state over a diffusion layer operation (as shown in Fig. 6.4), if the 7-bytes of ΔY [3, 4, 6, 8, 9, 13, 14] have equal differences, say y, then it will lead to non-zero difference only at byte 0 of ΔZ (instead of full state diffusion) after the diffusion layer operation. Rest all bytes

of ΔZ will be passive. Thus, under the given constraints, probability of the differential trail ΔY → Δ Z is 1.

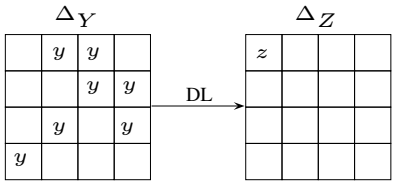


Figure 6.4: Differential property of diffusion layer

Proof. As per the diffusion layer specification of ARIA, each output byte of state Z is a xored sum of 7 input bytes of state Y. The same property is preserved in case of differences as well, i.e., each output byte difference of Z is a xor'ed sum of 7 input byte difference of Y. In lieu of this, for each output byte, if even number of corresponding input bytes (i.e., 2, 4 or 6) have equal differences, then they cancel out each other. In the above trail, 7 bytes of Y, i.e., Y [3, 4, 6, 8, 9, 13, 14] have equal differences 'y', whereas the rest of the bytes have zero differences. Hence, all output bytes except ΔZ [0] have zero differences since their xored sum have either 2 or 4 equal input byte difference. E.g.,

$$\begin{aligned}
\Delta Z[0] &= \Delta Y[3] \oplus \Delta Y[4] \oplus \Delta Y[6] \oplus \Delta Y[8] \oplus \Delta Y[9] \oplus \Delta Y[13] \oplus \Delta Y[14] \\
&= y \oplus y \oplus y \oplus y \oplus y \oplus y \oplus y = y \\
\Delta Z[1] &= \Delta Y[2] \oplus \Delta Y[5] \oplus \Delta Y[7] \oplus \Delta Y[8] \oplus \Delta Y[9] \oplus \Delta Y[12] \oplus \Delta Y[15] \\
&= 0 \oplus 0 \oplus 0 \oplus y \oplus y \oplus 0 \oplus 0 = 0 \\
\Delta Z[11] &= \Delta Y[2] \oplus \Delta Y[3] \oplus \Delta Y[4] \oplus \Delta Y[7] \oplus \Delta Y[9] \oplus \Delta Y[12] \oplus \Delta Y[14] \\
&= 0 \oplus y \oplus y \oplus 0 \oplus y \oplus 0 \oplus y = 0
\end{aligned}$$

Similar equations can be constructed for other output bytes of Z as well. Thus, property 1 holds true.

Property 2. For a given ARIA S-box, say S_1 and any non-zero input - output difference pair, say (Δ_i - Δ_o) in F_{256}, there exists one solution in average, say y, for which the equation, $S_1(y) \oplus S_1(y \oplus \Delta_i) = \Delta_o$, holds true (since ARIA uses AES S-box as S_1 [61]) This property is also applicable to other ARIA S-boxes, i.e., S_2, S_1^{-1} and S_2^{-2}.

The time complexity of the attack is measured in terms of number of full round (7 or 8) ARIA encryptions required. The memory complexity is measured in units of 128-bit ARIA blocks required.

6.3 Distinguishing Property of 4-round ARIA

Given a list of 256 distinct bytes (M^0, M^1, ..., M^{255}), a function $f : \{0,1\}^{128} \mapsto \{0,1\}^{128}$ and a 120-bit constant U, we define a multiset v as follows:

$$\begin{aligned} C^i &= f(M^i \,||\, U), \text{where } (0 \leq i \leq 255) \\ v &= \{C^0[0] \oplus C^0[0], C^1[0] \oplus C^0[0], \ldots, C^{255}[0] \oplus C^0[0]\} \end{aligned}$$

Note that, ($M^0 \,||\, U$, $M^1 \,||\, U$, ... , $M^{255} \,||\, U$) forms a δ-list and atleast one element of the multiset is always zero.

Distinguishing Property. Let us consider $\mathcal{F}$ to be a family of permutations on 128-bit. Then, given any list of 256 distinct bytes (M^0, M^1, ..., M^{255}), the aim is to find how many multisets v are possible when, $f \xleftarrow{\$} \mathcal{F}$ and $U \xleftarrow{\$} \{0,1\}^{120}$.

In case, when $\mathcal{F}$ = family of all permutations on 128-bit and $f \xleftarrow{\$} \mathcal{F}$. Under such setting, since in the multiset v, we have 255 values that are chosen uniformly and independently from the set $\{0, 1, \ldots, 255\}$ (as one element, say $C^0[0] \oplus C^0[0]$, is always 0), the total possible multisets v are $\binom{2^8-1+2^8-1}{2^8-1} \approx 2^{505.17}$.

In case, when $\mathcal{F}$ = 4-full rounds of ARIA and $f \xleftarrow{\$} \mathcal{F}$. Here, $f \xleftarrow{\$} \mathcal{F} \Leftrightarrow K \xleftarrow{\$} \{0,1\}^k$ and $f = E_K$, where, $k = 128$ (for ARIA-128), 192 (for ARIA-192) or 256 (for ARIA-256). Let us consider, 4-full rounds of ARIA as shown in Fig. 6.5 where, multiset v is defined as $v = \{Z_4^0[0] \oplus Z_4^0[0], Z_4^1[0] \oplus Z_4^0[0], \ldots, Z_4^{255}[0] \oplus Z_4^0[0]\}$. Then, we state the following *Observation 1.*

Observation 1. The multiset v is determined by the following 30 single byte parameters only :

- $X_2^0[3, 4, 6, 8, 9, 13, 14]$ (7-bytes)
- $X_3^0[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15]$ (full 16-byte state)
- $X_4^0[3, 4, 6, 8, 9, 13, 14]$ (7-bytes)

Thus, the total number of multisets possible is $2^{30\times 8} = 2^{240}$ since, each 30-bytes defines one multiset.

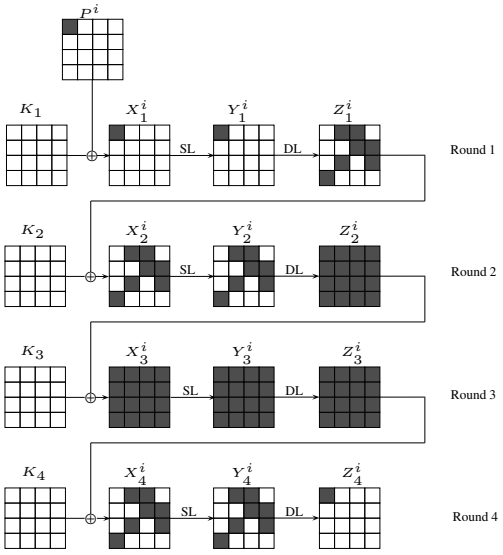


Figure 6.5: 4-Round distinguisher in ARIA . P^i denotes (M^i || U) and X_j^i, Y_j^i, Z_j^i denote intermediate states corresponding to P^i in round j. The round subkeys K_i, where, $1 \leq i \leq 4$ are generated from the master key K.

Proof. In round 1, the set of differences $\{X_1^0[0] \oplus X_1^0[0],\ X_1^1[0] \oplus X_1^0[0]\ ,\ \ldots\ldots,\ X_1^{255}[0] \oplus X_1^0[0]\}$ (or, equivalently, set of differences at $X_1[0]$) are known as there are exactly 256 differences possible. Since S-box S_1 is injective, exactly 256 values exist in the set $\{Y_1^0[0] \oplus Y_1^0[0],\ Y_1^1[0] \oplus Y_1^0[0],\ \ldots\ldots,\ Y_1^{255}[0] \oplus Y_1^0[0]\}$ as well. Due to DL and ARK operations being linear, the set of differences at X_2 [3, 4, 6, 8, 9, 13, 14] are known (according to diffusion layer (DL) definition discussed in Section 6.2).

Owing to the non-linearity of the substitution layer, the set of differences at Y_2[3, 4, 6, 8, 9, 13, 14] cannot be known and one cannot move forward. To alleviate this problem, it is sufficient to know X_2^0 [3, 4, 6, 8, 9, 13, 14], i.e., values of the active bytes of the first state (out of 256 states) at X_2 as it enables calculating the active bytes of the other X_2^i states (where, $1 \leq$ i ≤ 255) and cross SL in round 2. Again, since DL and ARK operations are linear, the set of differences $\{X_3^0 \oplus X_3^0,\ X_3^1 \oplus X_3^0,\ \ldots\ ,\ X_3^{255} \oplus X_3^0\}$ is known. In order to know the set of values $\{X_3^0,\ X_3^1,\ \ldots\ ,\ X_3^{255}\}$ for crossing the SL in round 3, it is sufficient to know the value of the full state X_3^0 which is given as a parameter.

By similar logic, as explained above, the set of differences $\{X_4^0 \oplus X_4^0,\ X_4^1 \oplus X_4^0,\ \ldots\ ,\ X_4^{255} \oplus X_4^0\}$ are known. Now, at this stage, if only X_4^0[3, 4, 6, 8, 9, 13, 14] bytes are known, the SL layer in round 4 can be crossed and the set of 256 values $\{Z_4^0[0],\ Z_4^1[0],\ \ldots,\ Z_4^{255}[0]\}$ at Z_4 can be computed. Then the value of multiset $v = \{\ Z_4^0[0] \oplus Z_4^0[0],\ Z_4^1[0] \oplus Z_4^0[0],\ \ldots\ ,\ Z_4^{255}[0] \oplus Z_4^0[0]\}$ can be determined easily as well. This shows that the multiset v depends on 30 parameters and can take 2^{240} possible values. □

Since, there are 2^{240} possible multisets at $Z_4[0]$, if we precompute and store these values in a hash table, then the precomputation complexity goes higher than brute force for ARIA-192. In order to reduce the number of multisets, we apply the Differential Enumeration technique suggested by Dunkelman et al. in [71] and improved by Derbez et al. in [61]. We call the improved version proposed in [61] as *Refined Differential Enumeration*.

Refined Differential Enumeration. The basic idea behind this technique is to choose a list of 256 distinct bytes (M^0, M^1, ..., M^{255}) such that several of the parameters that are required to construct the multiset equal some pre-determined constants.

To achieve so, we construct a truncated differential for four full rounds of ARIA, in which the input and output differences are non-zero at byte 0 only (as shown in Fig. 6.6.)

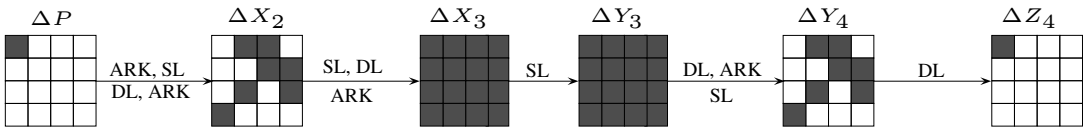


Figure 6.6: 4-Round truncated differential in ARIA

The probability of this trail is 2^{-120} as follows: the one byte difference at ΔP[0] propagates to 7-byte difference in ΔX_2 and 16-byte difference in ΔY_3 with probability 1. Next, the probability that full state difference in ΔY_3 leads to 7-byte difference in ΔY_4 is 2^{-72} (since 9 bytes of ΔY_4, i.e., ΔY_4[0, 1, 2, 5, 7, 10, 11, 12, 15]) have zero difference). Further, the probability that random differences in ΔY_3 yield equal differences in the active bytes of ΔY_4 i.e., ΔY_4[3, 4, 6, 8, 9, 12, 13] is 2^{-48}. [2] Therefore, the total probability of $\Delta Y_3 \rightarrow \Delta Y_4$ is $2^{-(72+48)} = 2^{-120}$. Then, by the virtue of *Property 1* (mentioned in Section 6.2), 7-byte difference in ΔY_4 yields a single byte difference in ΔZ_4[0] with probability 1. Thus, the overall probability of the differential from $\Delta P \rightarrow \Delta Z_4$ is 2^{-120}.

In other words, we require 2^{120} plaintext pairs to get a right pair. Once, we get a right pair, say (P^0, P^1), we state the following *Observation 2*:

Observation 2. Given a right pair (P^0, P^1) that follows the truncated differential trail shown in Fig. 6.6, then the 30 parameters corresponding to P^0 mentioned in *Observation 1* can take one of atmost 2^{128} fixed 30-byte values (out of the total 2^{240}

[2]The differences in 16-bytes of ΔY_3 yield differences in the 7 active bytes of ΔX_4 which in turn lead to 7-bytes difference in ΔY_4. The probability that these differences in the 7-bytes of ΔY_4 are equal is 2^{-48}.

possible values) where, each of these 2^{128} 30-byte values are defined by each of the 2^{128} values of the 16 following parameters:

- $\Delta Y_1[0]$
- $X_2^0[3, 4, 6, 8, 9, 13, 14]$
- $Y_4^0[3, 4, 6, 8, 9, 13, 14]$
- $\Delta Z_4[0]$

Proof. Given a right pair (P^0, P^1), the knowledge of these 16 new parameters allows us to compute all the differences shown in Fig. 6.5. This is so because, knowledge of $\Delta Y_1[0]$ allows computation of $\Delta Z_1[3, 4, 6, 8, 9, 13, 14]$ and $\Delta X_2[3, 4, 6, 8, 9, 13, 14]$. Then, if the values of X_2^0 [3, 4, 6, 8, 9, 13, 14] are known, one can compute the corresponding X_2^1 [3, 4, 6, 8, 9, 13, 14], cross the SL layer in round 2 and calculate the full state difference ΔX_3. Similarly, from the bottom side, knowledge of $\Delta Z_4[0]$ allows computation of $\Delta Y_4[3, 4, 6, 8, 9, 13, 14]$. Then, if the values of $Y_4^0[3, 4, 6, 8, 9, 13, 14]$ are known, one can easily determine $Y_4^1[3, 4, 6, 8, 9, 13, 14]$, compute the corresponding $X_4^0[3, 4, 6, 8, 9, 13, 14]$ and $X_4^1[3, 4, 6, 8, 9, 13, 14]$ respectively and subsequently full state ΔY_3. Then, using the differential property of ARIA S-boxes (*property 2* mentioned in Section 6.2), the possible values of X_3^0 and X_3^1 can be computed. □

Thus, the knowledge of these 16 bytes given in *Observation 2* allows computation of the corresponding 30 parameters described in *Observation 1*. Hence, total possible values of these 30 single byte parameters are atmost $2^{16\times 8} = 2^{128}$. Moreover, since these computations do not require the knowledge of key bytes, they can be easily pre-computed.

Using *Observation 1* and *Observation 2*, we state the following third *Observation 3*:

Observation 3. Given (M^0, M^1, ..., M^{255}) and $f \xleftarrow{\$} \mathcal{F}$ and $U \xleftarrow{\$} \{0,1\}^{120}$, such that $M^0 \,||\, U$ and $M^j \,||\, U$, (where, $j \in \{ 0, 1, \ldots, 255 \}$) is a right pair that follows differential trail shown in Fig. 7.8, then atmost 2^{128} multisets v are possible at $Z_4[0]$.

Proof. From *Observation 1*, we know that each 30-byte parameter defines one multiset and *Observation 2* restricts the possible values of these 30-byte parameters to 2^{128}. Thus, atmost 2^{128} multisets are only possible for ARIA. □

As the number of multisets in case of 128-bit random permutation ($= 2^{505.17}$) is much higher than 4-round ARIA ($= 2^{128}$), a valid distinguisher is constructed.

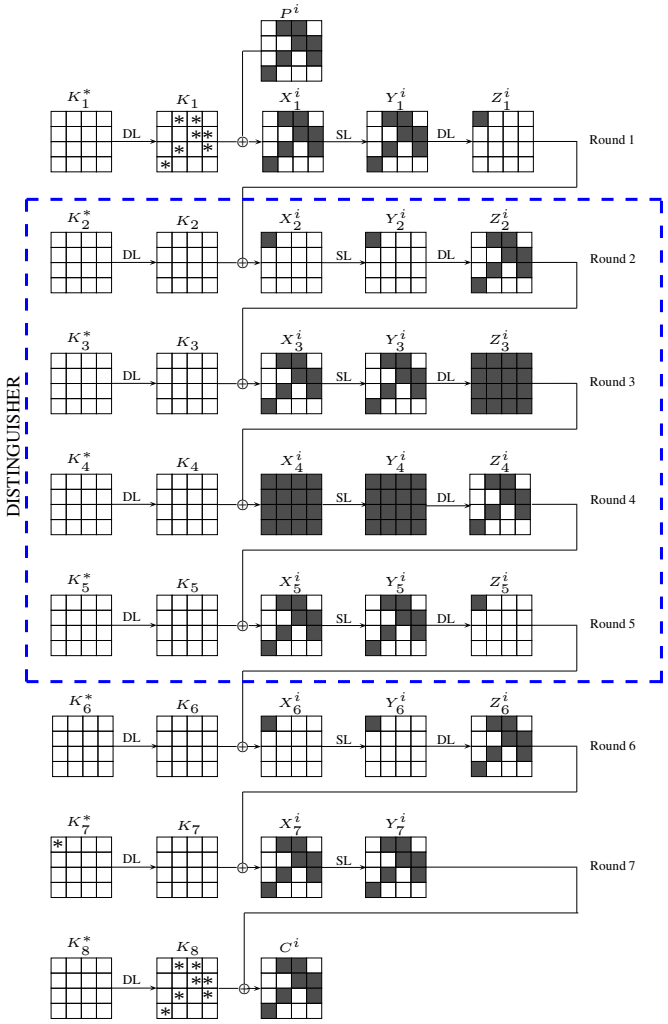


Figure 6.7: 7-round attack on ARIA-192/256. The subkey bytes derived are star marked.

6.4 Key Recovery Attack on 7-round ARIA-192/256

In this section, we use our *Observation 3* to launch a meet-in-the-middle attack on 7-round ARIA-192/256 to recover the key. The distinguisher is placed from round 2 to round 5, i.e., δ-list is constructed in state X_2 with byte 0 being the active byte and multiset is checked in $Z_5[0]$ (as shown in Fig. 6.7). One round at the top and two rounds at the bottom are added to the 4-round distinguisher. The attack consists of the following two phases:

Precomputation Phase. In this phase, we compute and store the 2^{128} possible multisets at $\Delta Z_5[0]$ in a hash table based on *Observation 2*.

Online Phase. If we extend the differential trail (shown in Fig. 6.6) by one round backwards, such that 7-bytes (3, 4, 6, 8, 9, 13 and 14) are active in the plaintext, then with a probability of 2^{-48}, these 7 active bytes will induce a non-zero difference of one byte in $X_2[0]$. Thus, we require $2^{120+48} = 2^{168}$ plaintext pairs to start our online phase. For each of these pairs, we will guess the subkey candidates for which the pair becomes a right pair and construct the corresponding δ-list. The steps of the online phase are:

1. We encrypt 2^{57} structures of 2^{56} plaintexts each, where bytes 3, 4, 6, 8, 9, 13 and 14 take all possible values and rest of the bytes are constants. [3].

2. For each structure, we store the ciphertexts in a hash table and look for pairs in which the difference in bytes 0, 1, 2, 5, 7, 10, 11, 12, 15 of the ciphertext is zero. Out of the total 2^{168} pairs, only 2^{96} pairs are expected to remain.

3. For each of the remaining 2^{96} plaintext pairs, we do the following:

   (a) We guess 7 bytes of $K_8[3, 4, 6, 8, 9, 13, 14]$ and check whether ΔY_6 has non zero difference only in byte 0 or not. Out of the 2^{56} possible values for K_8, only 2^8 key guesses are expected to remain (since with probability 2^{-48}, each will yield equal differences in the active bytes of ΔZ_6). Since we are only interested in checking the difference at $\Delta Y_6[0]$, $K_7[0]$ is not required to be guessed at this stage.

   (b) We then guess 7 bytes of $K_1[3, 4, 6, 8, 9, 13, 14]$ and check whether ΔZ_1 has non zero difference only in byte 0 or not. Out of the 2^{56} possible values for K_1, only 2^8 key guesses are expected to remain.

   (c) For each of the $2^8 \times 2^8 = 2^{16}$ remaining guesses of 14 active bytes of K_1 and K_8:

   - We take one of the members of the pair and find its δ-list at $Z_1[0]$ using the knowledge of 7 active bytes of K_1. [4]
   - We obtain the corresponding ciphertexts of the resulting plaintext set of the δ-list from the hash table. We guess byte $K_7^*[0] = DL^{-1}(K_7[0])$ $= K_7[3] \oplus K_7[4] \oplus K_7[6] \oplus K_7[8] \oplus K_7[9] \oplus K_7[13] \oplus K_7[14]$ and using the knowledge of $K_8[3, 4, 6, 8, 9, 13, 14]$, partially decrypt the ciphertexts of the δ-list to obtain the multiset at $\Delta Z_5[0]$ (which is same as that constructed in $\Delta X_6[0]$).
   - We then check whether this multiset exists in the precomputed table or not. If not, we discard the corresponding key guess.

[3] One structure has $2^{56} \times 2^{55} = 2^{111}$ plaintext pairs. Therefore, 2^{57} structures have $2^{57+111} = 2^{168}$ plaintext pairs.

[4] Encrypt the chosen right pair message to one full round using $k_1[3, 4, 6, 8, 9, 13, 14]$ and compute $Z_1[0]$. Xor other $Z_1[0]$ byte with 255 other values and decrypt them back to obtain the other plaintexts.

The probability for a wrong guess to pass the test is $2^{128} \times 2^{-467.6} = 2^{-339.6}$. [5] Since we try only $2^{96+16+8} = 2^{120}$ multisets, only the right subkey should verify the test with a probability close to 1.

Complexities. The time complexity of the precomputation phase is $2^{128} \times 2^8 \times 2^{-1.9} = 2^{134.1}$ ARIA encryptions. [6] The time complexity of the online phase is dominated by step 3(c) which is $2^{96} \times 2^{16} \times 2^8 \times 2^8 \times 2^{-2.9} = 2^{125.1}$ ARIA encryptions. Clearly the time complexity of this attack is dominated by the precomputation phase. It was shown in [61] that each 256-byte multiset requires 512-bit space. Hence, the memory complexity of the attack is $2^{128} \times 2^2 = 2^{130}$ 128-bit ARIA Blocks. The data complexity of the attack is 2^{113} plaintexts.

6.4.1 Recovering the master key for 7-round ARIA-192

In the above attack, 7-bytes of subkeys K_1 and 7-bytes of K_8 as well as 1 byte of K_7^* were recovered. In order to recover the master key, we do the following:

1. Guess 16-bytes of W_0.
   (a) Using the guessed value of W_0 and 7-bytes of K_1 recovered in the attack, we can deduce 56-bit of W_1 from Eq. 6.5. It is observed that 16-bit of this 56-bit of W_1 deduced, are part of 11^{th}, 12^{th} and 13^{th} bytes and rest 40-bits are part of first 8 bytes.
   (b) We then calculate $F_o(W_0, CK_1)$. We already know that for ARIA-192, $KR[8, 9, \ldots, 15] = 0$. Thus, $W_1[8, 9, \ldots, 15]$ equals the corresponding bytes of $F_o(W_0, CK_1)$ following Eq. 6.2.
   (c) Next, we discard the guesses of W_0 for which the common 16-bit of W_1 computed in (a) and (b) do not match. 2^{112} guesses of W_0 are expected to remain.
2. For each of the remaining guesses of W_0, we guess 24-bits of $W_1[0, 1, \ldots 7]$ other than the 40-bits deduced in 1(a) to know the 2^{24} possible values of W_1 corresponding to each of W_0.
3. For each remaining guesses of W_0 and corresponding guesses of W_1, we deduce W_2 and W_3 from Eqs. 6.3 and 6.4.

[5] Note that the probability of randomly having a match is $2^{-467.6}$ and not $2^{-505.17}$ since the number of ordered sequences associated with a multiset is not constant [71].

[6] The normalization factor of $2^{-1.9}$ is calculated by calculating the ratio of number of S-Box operations required in the precomputation phase to the total number of S-Box operations performed in 7-Round ARIA encryption. Similarly all other normalization factors have been calculated.

(a) Following Eq. 6.12, we deduce K_8 and compare its bytes 3, 4, 6, 8, 9, 13 and 14 with the values of the same 7-bytes of K_8 recovered from the attack. We discard the guesses of W_0 and W_1 in case of mismatch of these 7-bytes of K_8. The same process is repeated for 1-byte of K_7^*. This is an 8-byte and 64-bit filtering. Out of 2^{136}, 2^{72} guesses of W_0 and W_1 are expected to remain which can be tested by brute force to obtain the correct master key.

The time complexity of the recovering process of step 3 is maximum. It is equal to $2^{136} \times (2/7) = 2^{134.2}$ 7-round ARIA encryptions as we need to compute 2 rounds of ARIA to deduce W_2 and W_3 and all other operations have negligible complexity as they are simple linear operations.

Therefore, the final time complexity of the attack is $2^{134.2} + 2^{134} = 2^{135.1}$. Other complexities remain the same.

6.4.2 Recovering the master key for 7-round ARIA-256

In the above attack, 7-byte of subkey K_1 and 7-byte of subkey K_8 as well as 1 byte of K_7^* were recovered. As shown in Fig. 6.7, we have obtained a trail such that 1^{st} byte is active at X_2. In order to recover all 16-bytes of subkey K_1, we can repeat the attack 4 times by modifying the trail such that we get a different byte active at X_2:

- bytes 3,4,6,8,9,13,14 to obtain byte 0 active at X_2
- bytes 2,5,7,8,9,12,15 to obtain byte 1 active at X_2
- bytes 1,4,6,10,11,12,15 to obtain byte 2 active at X_2
- bytes 0,5,7,10,11,13,14 to obtain byte 3 active at X_2

The time and data complexity of the attack will become 4 times larger than time and data complexities mentioned in the 7-round attack in Section 6.4 respectively. Then, we do the following to recover the master key:

1. We guess 16-bytes of W_0
2. For each guess of W_0, using the value of K_1 recovered from the attack, we obtain W_1 from Eq. 6.2. Then we follow Step 3 as mentioned in Section 6.4.1

The time complexity of recovering the master key is $2^{128} \times (2/7) = 2^{126.2}$ 7-round ARIA encryptions.

Therefore, the final time complexity of the attack is $(4 \times 2^{134}) + 2^{126.2} = 2^{136}$. The data complexity of the attack becomes 2^{115} while the memory complexity remains same.

6.5 Key Recovery Attack on 8-round ARIA-256

In this section, we describe our meet-in-the-middle attack on 8-round ARIA-256.

6.5.1 Construction of 4.5-round distinguisher

For the 8-round attack, the distinguisher constructed in Fig. 6.5 is extended by half round forwards upto Y_5 (DL operation is omitted). The 4.5 round distinguisher for 8-round attack is shown in Fig. 6.8. Similar to *Observation 1*, we state the following *Observation 4*:

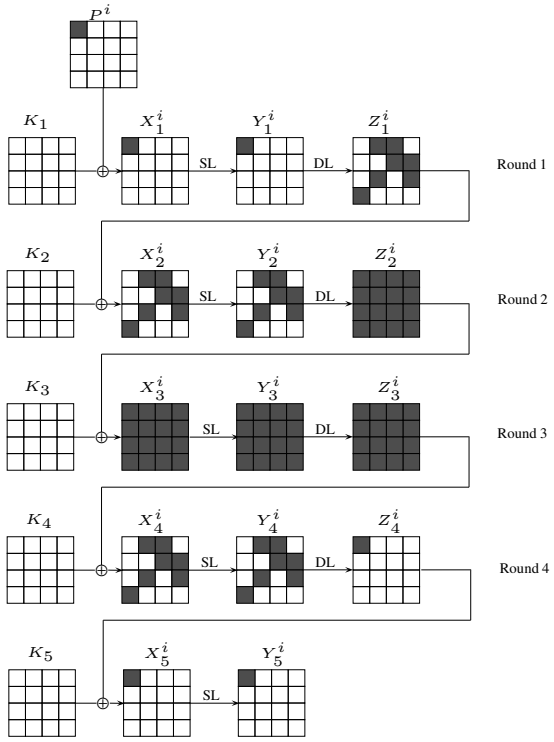


Figure 6.8: 4.5-Round distinguisher in ARIA

Observation 4. Given (M^0, M^1, ..., M^{255}) and $f \xleftarrow{\$} \mathcal{F}$ and $U \xleftarrow{\$} \{0,1\}^{120}$, where, f represents 4.5 rounds of ARIA, the multiset $v = \{Y_5^0[0] \oplus Y_5^0[0],\ Y_5^0[0] \oplus Y_5^1[0], \ldots\ldots,\ Y_5^0[0] \oplus Y_5^{255}[0]\}$ is determined by the following 31 1-byte parameters:

- $X_2^0[3, 4, 6, 8, 9, 13, 14]$
- $X_3^0[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15]$ (full 16-byte state)
- $X_4^0[3, 4, 6, 8, 9, 13, 14]$

- $X_5^0[0]$

The number of possible multisets is $2^{31\times8} = 2^{248}$. The proof for this is similar to that described for *Observation 1* in Section 6.3.

Number of admissible multisets. The differential trail shown in Fig. 7.8 can be extended 0.5 round forwards to ΔY_5 in which only byte 0 is active with probability 1, i.e., the probability of differential trail: $\Delta P \rightarrow \Delta Y_5$ remains 2^{-120}. Then, similar to *Observation 2*, we state the following *Observation 5*.

Observation 5. Given a right pair (P^0, P^1) that follows the truncated differential trail $(\Delta P \rightarrow \Delta Y_5)$, then the 31 parameters corresponding to P^0 mentioned in *Observation 4* can take one of at most 2^{136} fixed 31-byte values (out of the total 2^{248} possible values) where, each of these 2^{136} 31-byte values are defined by each of the 2^{136} values of the 17 following parameters:

- $\Delta Y_1[0]$
- $X_2^0[3, 4, 6, 8, 9, 13, 14]$
- $Y_4^0[3, 4, 6, 8, 9, 13, 14]$
- $\Delta Z_4[0]$
- $X_5^0[0]$

The proof of this *Observation* is similar to the proof of *Observation 2* described in Section 6.3. From, *Observation 4* and *Observation 5*, we can say that the total number of admissible multisets is $2^{17\times8} = \mathbf{2^{136}}$.

6.5.2 Key Recovery Attack

In this section, we discuss our 8-round attack. The distinguisher is placed from round 2 to round 5.5, i.e, δ-list is constructed in state X_2 with byte 0 being the active byte and multiset is checked in $Y_6[0]$ (as shown in Fig. 6.9). One round at the top and three rounds at the bottom are added to the 4.5-round distinguisher. The attack consists of the following two phases:

Precomputation Phase. Compute and store the 2^{136} possible multisets at $\Delta Y_6[0]$ in a hash table based on *Observation 5*.

Online Phase. The steps of the online phase are:

1. Encrypt 2^{57} structures of 2^{56} plaintexts each, where bytes 3, 4, 6, 8, 9, 13 and 14 take all possible values and rest of the bytes are constants. Store the ciphertexts in a hash table.

2. For each of the 2^{168} plaintext pairs do the following:

   (a) For each 2^8 guesses of ΔZ_1 [0], resolve input-output differences at SL layer of round 1 (using *Property 2*) and deduce the corresponding value of K_1[3, 4, 6, 8, 9, 13, 14].

   (b) For each $2^8 \times 2^{56} = 2^{64}$ guesses of ΔY_6[0] and ΔY_7 [3, 4, 6, 8, 9, 13, 14], resolve input-output differences at SL layers in round 7 and round 8 respectively and deduce corresponding K_8^* [3, 4, 6, 8, 9, 13, 14] and full subkey K_9.

   (c) For each of the $2^{64+8} = 2^{72}$ guesses of 30 bytes of K_1, K_8^* and K_9:

   - Take one of the members of the pair and find its δ-list using the knowledge of 7 active bytes of K_1.
   - Get the corresponding ciphertexts of the resulting plaintext set of the δ-list from the hash table. Using the knowledge of K_9 and K_8^* [3, 4, 6, 8, 9, 13, 14], partially decrypt the ciphertexts of the δ-list to compute the multiset at ΔY_6[0].
   - Check whether this multiset exists in the precomputed table or not. If not, then discard the corresponding key guess.

The probability for a wrong guess to pass the test is $2^{136} \times 2^{-467.6} = 2^{-331.6}$. Since, we try only $2^{168+72} = 2^{240}$ multisets, only the right subkey should verify the test with a probability close to 1.

Complexities. The time complexity of the precomputation phase is $2^{136} \times 2^8 \times 2^{-2} = 2^{142}$ ARIA encryptions. The time complexity of the online phase is dominated by step 2(c) which is $2^{168} \times 2^{72} \times 2^8 \times 2^{-2.1} = 2^{245.9}$ ARIA encryptions. Clearly the time complexity of this attack is dominated by the online phase. The memory complexity of the attack is $2^{136} \times 2^2 = 2^{138}$ 128-bit ARIA Blocks. The data complexity of the attack is 2^{113} plaintexts.

6.5.3 Recovering the actual master key

In the above attack, 7-bytes of subkeys k_1 and k_8 as well as full subkey k_9 were recovered. Once these bytes are known, the remaining bytes in k_1 and k_8 can be found by exhaustive search without affecting the overall complexity of the 8-round attack. When full subkeys k_1 and k_9 are known, then the master key K can be recovered as follows. Since, Eq. 6.5 and Eq. 6.6 are two equations in two variables, they can be solved through standard matrix method by constructing a (256×256) binary matrix. We found the rank of this matrix to be 240 suggesting 2^{16} solutions for the tuple (W_0 and W_1). Once the values of W_0 and W_1 are known, KL and KR can be obtained through Eq. 6.1 & Eq. 6.2 respectively. Thus, we get 2^{16} solutions for the master key K. The master key can then be easily recovered through brute force.

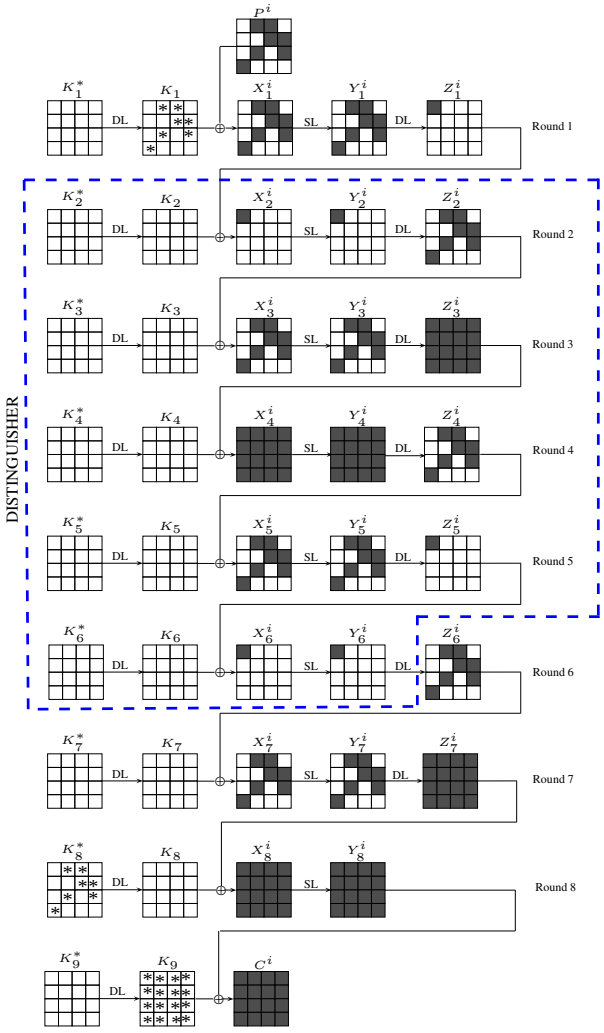


Figure 6.9: 8-round attack on ARIA-256. The subkey bytes derived are star marked.

6.6 Conclusions

In this chapter, we discuss multiset attacks and its results on AES. We then explore the space of multiset attacks as applied to key recovery attack on ARIA-192 and ARIA-256. We improve the previous 7-round and 8-round attacks on these structures and show the best attacks on them. We achieve these results by constructing a new 4-round distinguisher on ARIA and applying MITM attacks on the rest of the rounds. We also show recovery of the actual master key through our 7 and 8-round attacks on ARIA-192 and ARIA-256. To the best of our knowledge, this is the first attempt in this direction. Currently, the number of attacked rounds remains 8 and it would be an interesting problem to try applying multiset attacks to break more rounds of ARIA.

Chapter 7

Multiset based Meet-in-the-Middle Attack on Kalyna-128/256 and Kalyna-256/512

In this chapter, we continue with multiset attacks and discuss further advancements in this technique. In the previous chapter, we had reviewed the multiset attacks on AES variants that had delivered the best results on AES-128 till now. [1] However, for AES-192 and AES-256, further improvements were reported in literature. The multiset attack on AES-192/256 was improved by Li et al. in [122]. They extended the number of rounds attacked to 9 by introducing the concept of *key sieving*, where dependencies between the AES round keys were exploited to filter out the wrong states and reduce the number of possible admissible multisets. Recently, in [147], Li et al. demonstrated the most efficient multiset attack on AES-256. By exploiting some more key sieving properties and clever MixColumns properties, they extended the number of rounds attacked to 10. Table 7.1 summarizes the best multiset attacks currently existing on AES-192 and AES-256.

The Kalyna block cipher has recently been established as the Ukranian encryption standard in June, 2015 [138]. It was selected in a Ukrainian National Public Cryptographic Competition running from 2007 to 2010. Kalyna supports block sizes and key lengths of 128, 256 and 512 bits. Denoting the variants of Kalyna as Kalyna-b/k, where b denotes the block size and k denotes the keylength, the design specifies $k \in \{b, 2b\}$ and defines 5 variants of Kalyna: Kalyna - 128/128, Kalyna - 128/256, Kalyna - 256/256, Kalyna - 256/512 and Kalyna - 512/512. The number of rounds of these variants are - 10, 14, 14, 18 and 18 respectively. Structurally, Kalyna is quite similar to AES [57] but has an increased MDS matrix size, a new set of four different S-boxes, pre-and post-whitening modular 2^{64} key addition and a new key scheduling algorithm. Since, this block cipher has been introduced very recently, therefore, it has not yet received

[1] Apart from biclique based key recovery attacks

Table 7.1: Summary of multiset attacks existing on AES-192 and AES-256

Target	Rounds	Time	Data	Memory	Reference
AES-192	8	2^{208}	2^{32}	2^{206}	[60]
	8	2^{172}	2^{113}	2^{129}	[71]
	8	2^{172}	2^{107}	2^{96}	[61]
	9	$2^{186.5}$	2^{117}	$2^{177.5}$	[122]
AES-256	8	2^{208}	2^{32}	2^{206}	[60]
	8	2^{196}	2^{113}	2^{129}	[71]
	8	2^{196}	2^{107}	2^{96}	[61]
	9	2^{203}	2^{120}	2^{203}	[61]
	9	$2^{203.5}$	2^{121}	$2^{169.9}$	[122]
	10	2^{253}	2^{111}	$2^{211.2}$	[147]

significant attention of the cryptanalysis community. As multiset class of attacks has produced the best results on AES, hence, we investigate the effectiveness of improved multiset attack on Kalyna in this chapter.

The roadmap for this chapter is as follows: In Section 7.1, we describe Kalyna and present the current cryptanalytic results existing on it. This is followed by Section 7.2 where we mention some definitions and notations used in this chapter. In Section 7.3, we give details of our 6-round distinguisher for Kalyna 128/256 followed by Section 7.4 where we present our 9-round attack on the same. In Section 7.5, we describe our 6-round distinguisher for Kalyna-256/512 and in Section 7.6 we discuss our 9-round attack on the same. Finally in Section 7.7, we summarize and conclude this chapter. The original contribution of this work is from Section 7.2 to Section 7.6.

7.1 Description of Kalyna

The block cipher Kalyna proposed by Oliynykov et al. [138] has been recently selected as the Ukranian encryption standard in 2015. The aim of standardizing a new block cipher was to replace the previous standard – block cipher GOST owing to its slower speed in software implementations as compared to AES and presence of some effective theoretical attacks on it. In the first stage, 5 submissions were accepted namely –

Kalyna, *Mukhomor*, *Labirynth*, *ADE* and *RSB*. Out of this 3 block ciphers - *Kalyna*, *Mukhomor* and *Labirynth* went on to the second stage from which *Kalyna* was finally declared as the winner.

As discussed earlier, the block cipher Kalyna-b/k has five variants namely: Kalyna-128/128, Kalyna-128/256, Kalyna-256/256, Kalyna-256/512 and Kalyna-512/512 where, b is the block size and k is the key size. The 128-bit, 256-bit and 512-bit internal states are treated as a byte matrix of 8×2 size, 8×4 size and 8×8 size respectively where, the bytes are numbered column-wise (as shown in Fig. 7.1). Each internal round consists of 4 basic operations (as shown in Fig. 7.2):

0	8
1	9
2	10
3	11
4	12
5	13
6	14
7	15

(a)

0	8	16	24
1	9	17	25
2	10	18	26
3	11	19	27
4	12	20	28
5	13	21	29
6	14	22	30
7	15	23	31

(b)

0	8	16	24	32	40	48	56
1	9	17	25	33	41	49	57
2	10	18	26	34	42	50	58
3	11	19	27	35	43	51	59
4	12	20	28	36	44	52	60
5	13	21	29	37	45	53	61
6	14	22	30	38	46	54	62
7	15	23	31	39	47	55	63

(c)

Figure 7.1: (a)Byte numbering in a state of Kalyna-128. (b) Byte numbering in a state of Kalyna-256. (c) Byte numbering in a state of Kalyna-512

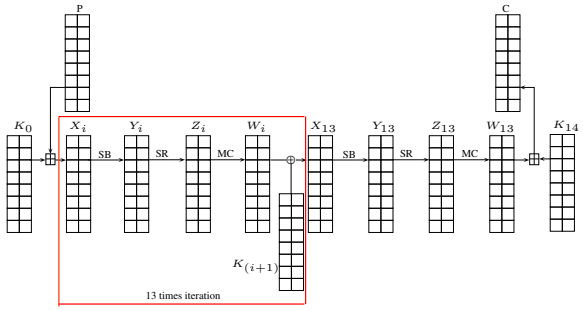
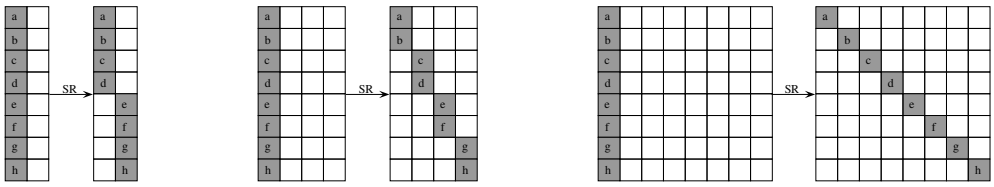


Figure 7.2: One full encryption in Kalyna-128/256. (Refer to Section 7.2 for notations)

1. *SubBytes (SB)* - The nonlinear substitution layer uses four types of 8-bit S-boxes: S_0, S_1, S_2 and S_3. Each S-Box is defined to be an affine transformation of the inversion function over GF(2^8). Each byte x of row j (where $0 \leq j \leq 7$) in an intermediate state s is substituted by $S_{j \ mod \ 4}\ (x)$.

2. *Shift Rows (SR)* - The linear shift rows operation performs circular right shift on each row of an internal state (as shown in Fig. 7.3). The number of shifts (δ_j) in each row j (where $0 \leq j \leq 7$) is calculated by $\delta_j = \lfloor \frac{j \cdot b}{512} \rfloor$, where b denotes the block size. E.g., for row $j = 6$ and block size $b = 128$, the number of shifts (δ_j) = 1. The inverse shift rows operation circularly left shifts the elements by δ_j in each row.

3. *MixColumns (MC)* - This linear transformation pre-multiplies each column of the state matrix by a 8×8 MDS matrix over GF(2^8). The vector `(0x01, 0x01, 0x05, 0x01, 0x08, 0x06, 0x07, 0x04)` forms the circulant MDS matrix for the MixColumns operation (shown below) whereas the vector `(0xAD, 0x95, 0x76, 0xA8, 0x2F, 0x49, 0xD7, 0xCA)` forms the circulant MDS matrix for the inverse MixColumns operation. The branch factor of this MDS matrix is 9.



(a) Shift Rows operation in Kalyna-128

(b) ShiftRows operation in Kalyna-256

(c) ShiftRows operation in Kalyna-512

Figure 7.3: Illustration of *ShiftRows* operation in Kalyna variants

The polynomial $x^8 + x^4 + x^3 + x^2 + 1$ (represented as `0x11D` in short) is used as the irreducible polynomial for Galois field multiplication. It is to be noted that unlike AES, in the last round of Kalyna, MixColumns operation is not omitted.

$$\begin{bmatrix} 1 & 1 & 5 & 1 & 8 & 6 & 7 & 4 \\ 4 & 1 & 1 & 5 & 1 & 8 & 6 & 7 \\ 7 & 4 & 1 & 1 & 5 & 1 & 8 & 6 \\ 6 & 7 & 4 & 1 & 1 & 5 & 1 & 8 \\ 8 & 6 & 7 & 4 & 1 & 1 & 5 & 1 \\ 1 & 8 & 6 & 7 & 4 & 1 & 1 & 5 \\ 5 & 1 & 8 & 6 & 7 & 4 & 1 & 1 \\ 1 & 5 & 1 & 8 & 6 & 7 & 4 & 1 \end{bmatrix}$$

4. *Add Round Key (ARK)* - This step involves an exclusive-or operation with the round subkey. However, for the pre-whitening and post-whitening keys, key addition operation involves addition modulo 2^{64}. This was done to increase the resistance of Kalyna against differential and linear attacks. The round subkeys are of the same size as the intermediate state size.

Key Scheduling Algorithm. The key scheduling algorithm of Kalyna first involves splitting of the master key K into two parts - K_α and K_ω. If the block size and key size are equal, i.e., $(k = b)$, then $K_\alpha = K_\omega = K$, otherwise if $(k = 2b)$, then $K_\omega \mid\mid K_\alpha = K$, i.e., K_α is set as $b/2$ least significant bits of K and K_ω is set as $b/2$ most significant bits of K. Using these two parameters, an intermediate key K_t is generated which is then used to independently generate even indexed round keys (as shown in Fig. 7.4). Complete details of the key schedule algorithm are not relevant to the attacks described in this work and hence are omitted here. One may refer to [138] for the same.

Two properties which are important for us are as follows:

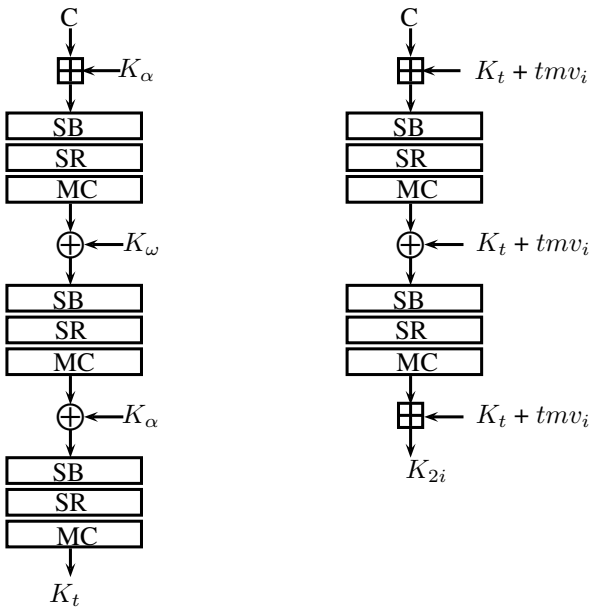


Figure 7.4: Key Schedule Algorithm of Kalyna. K denotes the master key, C is a predefined constant and $tmv_i = K \ggg 32.i$ where i denotes the round index.

1. Recovery of a subkey does not allow recovery of master key better than brute force.

2. The keys for round i where i is an odd number can be linearly computed from the key used in round $(i-1)$ and vice- versa as follows:

$$K_{i+1} = K_i \ggg (b/4 + 24) \tag{7.1}$$

where, $\ggg$ denotes circular right shift operation.

Previous cryptanalytic results on Kalyna. The official version of Kalyna specification (in English) available publicly does not include any security analysis of the design. A preliminary study in [1], before this cipher was standardized, reports attack complexities for Kalyna -128/128 against various attacks such as differential, linear, integral, impossible differential, boomerang etc. and shows that upto 5 rounds of this variant can be broken. Similar results are claimed for other Kalyna variants as well. The designers of Kalyna thus claim brute force security against Kalyna for rounds ≥ 6. In [13], AlTawy et al. presented the first detailed key recovery attack against standardized Kalyna-128/256 and Kalyna-256/512. They applied meet-in-the-middle (MITM) attack [61, 71] to break 7-rounds of both Kalyna variants and demonstrated the best attack on Kalyna so far. These results are summarized in Table 7.2.

Since, no justification of the cryptanalytic results obtained in [1] is given and the very fact that the claim of the Kalyna designers has been nullified by Altawy et al. motivated us to investigate the effectiveness of improved multiset attacks on Kalyna. In our attacks, we examine Kalyna-128/256 and Kalyna-256/512. We construct new 6-round distinguishers for both the variants and use it to extend our attacks up to 9 rounds. For Kalyna-256/512, we significantly reduce the data and time complexities of the previous best 7-round attack on the same [13]. The key schedule algorithm of Kalyna does not allow recovery of all subkeys or the master key from one subkey only unlike AES [57]. However, it allows recovery of odd-round keys from even-round keys and vice-versa. This property will be used by us in our attacks to reduce the attack complexities. To the best of our knowledge, our attacks are the first attacks on 9-round Kalyna-128/256 and Kalyna-256/512 respectively.

Table 7.2: Comparison of cryptanalytic attacks on round reduced variants of Kalyna. The blank entries were not reported in [1]. (The memory complexity header represents the number of 128-bit blocks for Kalyna-128 and 256-bit blocks for Kalyna-256 required to be stored in memory.)

Algorithm	Rounds attacked	Attack Type	Time	Data	Memory	Reference
Kalyna-128/128	2	Interpolation	–	-	-	[1]
	3	Linear Attack	$2^{52.8}$	-	-	[1]
	4	Differential	2^{55}	-	-	[1]
	4	Boomerang	2^{120}	-	-	[1]
	5	Impossible Differential	2^{62}	-	2^{66}	[1]
	5	Integral	2^{97}	-	2^{33+4}	[1]
Kalyna-128/256	7	Meet-in-the-Middle	$2^{230.2}$	2^{89}	$2^{202.64}$	[13]
	9	Meet-in-the-Middle	$2^{245.83}$	2^{105}	$2^{226.86}$	Sec. 7.4
Kalyna-256/512	7	Meet-in-the-Middle	$2^{502.2}$	2^{233}	2^{170}	[13]
	9	Meet-in-the-middle	$2^{477.83}$	2^{217}	$2^{443.45}$	Sec. 7.5

7.1.1 Our Contribution.

The main contributions of this chapter are as follows:

- We present the first 9-round key recovery attack on Kalyna-128/256 and Kalyna-256/512.

- We apply multiset attack to construct new 6-round distinguishers on each of the above mentioned Kalyna variants.
- Our 9-round attack on Kalyna-128/256 has data/time/memory complexity of 2^{105}, $2^{245.83}$ and $2^{226.86}$ respectively.
- Our 9-round attack on Kalyna-256/512 has data/time/memory complexity of 2^{217}, $2^{477.83}$ and $2^{443.45}$ respectively. This improves upon the previous best attack [13] in terms of time and data complexity as well.

Our results are summarized in Table 7.2.

7.2 Definitions and Notations

We utilize the following definitions for our attacks.

Definition 1 (δ-list). We define the δ-list as an ordered list of 256 16-byte (or 32-byte) distinct elements that are equal in 15 (or 31) bytes for Kalyna-128 (or Kalyna-256). Each of the equal bytes are called as passive bytes whereas the one byte that takes all possible 256 values is called the active byte. We denote the δ-list as (x^0, x^1, $x^2, \ldots, x^{255}$) where x^j indicates the j^{th} 128-bit (or 256-bit) member of the δ-list for Kalyna-128 (or Kalyna-256). As mentioned in the notations, $x_i^j[m]$ represents the m^{th} byte of x^j in round i.

Definition 2 (Multiset). A multiset is a set of elements in which multiple instances of the same element can appear. A multiset of 256 bytes, where each byte can take any one of the 256 possible values, can have $\binom{2^8+2^8-1}{2^8} \approx 2^{506.17}$ different values. [2]

Definition 3 (Super S-Box). The Kalyna Super S-box (denoted as SSB) can be defined similar to AES Super S-box [58]. For each 8-byte key, it produces a mapping between an 8-byte input array to an 8-byte output array. Formally, a two round Kalyna can be written as:

$$SB \rightarrow SR \rightarrow MC \rightarrow ARK \rightarrow SB \rightarrow SR \rightarrow MC \rightarrow ARK$$

or,

$$SR \rightarrow \underbrace{SB \rightarrow MC \rightarrow ARK \rightarrow SB} \rightarrow SR \rightarrow MC \rightarrow ARK \text{ [3]} \tag{7.2}$$

Since, MixColumns operation operates on a column of the state, the above map ($SB \rightarrow MC \rightarrow AK \rightarrow SB$) in Eq. 7.2 can be described as d parallel instances of

[2] Count of multisets of cardinality r with elements from a set with cardinality $n = \binom{n+r-1}{r}$

[3] Note that Sub Bytes and Shift Row operations in the first round have been interchanged as these functions commute with each other

SSB, where $d = 2$, 4 and 8 for Kalyna-128, Kalyna-256 and Kalyna-512 respectively.

Two important properties that will be used in our attacks are as follows:

Property 1a. (Kalyna S-box) For any given Kalyna S-box, say S_i (where, $i = 0$, 1, 2 or 3) and any non-zero input - output difference pair, say (Δ_{in}, Δ_{out}) in $F_{256} \times F_{256}$, there exists one solution in average, say y, for which the equation, $S_i(y) \oplus S_i(y \oplus \Delta_{in}) = \Delta_{out}$, holds true.

Property 1b. (Kalyna Super S-box) For any given Kalyna Super S-box, say SSB and any non-zero input - output difference pair, say (Δ_{in}, Δ_{out}) in $F_{2^{64}} \times F_{2^{64}}$, the equation, $SSB(z) \oplus SSB(z \oplus \Delta_{in}) = \Delta_{out}$ has one solution in average.

Property 2. (Kalyna MixColumns) If the values (or the differences) in any eight out of its sixteen input/output bytes of the Kalyna MixColumns operation are known, then the values (or the differences) in the other eight bytes are uniquely determined and can be computed efficiently. This is similar to AES MixColumns property stated in [147].

Proof. The Kalyna MixColumns works on a column of 8 bytes. Thus, the inputs and outputs of Kalyna MixColumns operation can be related through 8 equations. Therefore, out of 16 variables (8 input and 8 output), if 8 variables are known then the other 8 variables can be uniquely determined through the 8 equations. This is because as mentioned in [82], any sub-matrix of a MDS matrix is invertible which guarantees existence of an unique solution.

The time complexity of the attack is measured in terms of 9-round Kalyna encryptions required. The memory complexity is measured in units of b-bit Kalyna (where, $b = 128$ or 256) blocks required.

The following notations are followed in this chapter:

$\mathbf{P}$	:	Plaintext
$\mathbf{C}$	:	Ciphertext
i	:	Round number i, where, $0 \leq i \leq 8$
Kalyna-b	:	Kalyna with state size of b-bits
Kalyna-b/k	:	Kalyna with state size of b-bits and key size of k-bits
$\mathbf{K_i}$	:	Subkey of round i
$\mathbf{U_i}$	:	$MC^{-1}(\mathrm{K}_i)$, where, MC^{-1} is the inverse MixColumns operation
$\mathbf{X_i}$	:	State before SB in round i
$\mathbf{Y_i}$	:	State before SR in round i
$\mathbf{Z_i}$	:	State before MC in round i
$\mathbf{W_i}$	:	State after MC in round i
$\mathbf{\Delta s}$	:	Difference in a state s
$\mathbf{s_i[m]}$	:	m^{th} byte of state s in round i, where, $0 \leq m \leq l$ and $l = 15$ for Kalyna-128/256 and $l = 31$ for Kalyna-256/512
$\mathbf{s_i[p-r]}$	:	p^{th} byte to r^{th} byte (both inclusive) of state s in round i, where, $0 \leq p < r \leq l$ and $l = 15$ for Kalyna-128/256 and $l = 31$ for Kalyna-256/512

In some cases we are interested in interchanging the order of the *MixColumns* and *Add Round Key* operations. As these operations are linear, they can be swapped, by first xoring the intermediate state with an equivalent key and then applying the *MixColummn* operation (as shown in Figs. 7.5, 7.6). This is exactly similar to what one can do in AES [61]. As mentioned above, we denote the equivalent round key by $U_i = MC^{-1}(K_i)$.

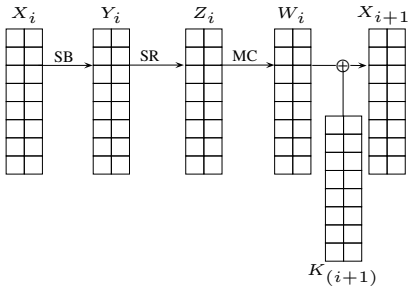


Figure 7.5: Normal one round of Kalyna-128/128.

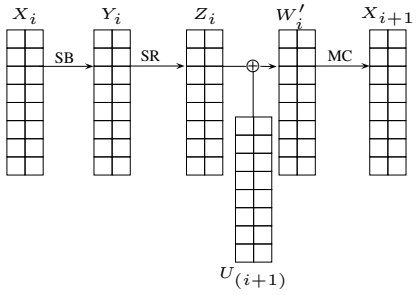


Figure 7.6: One round of Kalyna-128/128 with swapped MC and ARK operation. Here, $W'_i = Z_i \oplus U_{i+1}$.

7.3 Construction of distinguisher for 6-round Kalyna-128/256

In this section, we construct a distinguisher on the 6-inner rounds of Kalyna-128/256. Before, we proceed further, we first establish the following relation for Kalyna-128/256. According to *Property 2*, we can form an equation using any 11 out of 16 input-output bytes in the Kalyna MixColumns operation. For any round j, where, $0 \leq j \leq 8$:

$$\begin{aligned} \texttt{0xCA} \cdot Z_j[12] \oplus \texttt{0xAD} \cdot Z_j[13] &= \texttt{0x94} \cdot W_j[8] \oplus \texttt{0xB4} \cdot W_j[9] \oplus \\ \oplus\ \texttt{0x49} \cdot Z_j[14] \oplus \texttt{0xD7} \cdot Z_j[15] &\quad \texttt{0x4E} \cdot W_j[10] \oplus \texttt{0x7E} \cdot W_j[11] \oplus \\ &\quad \texttt{0xC0} \cdot W_j[13] \oplus \texttt{0xDA} \cdot W_j[14] \oplus \\ &\quad \texttt{0xC5} \cdot W_j[15] \qquad (7.3) \end{aligned}$$

$$or,$$

$$\begin{aligned} \texttt{0xCA} \cdot Z_j[12] \oplus \texttt{0xAD} \cdot Z_j[13] &= \texttt{0x94} \cdot (K_j[8] \oplus X_{j+1}[8]) \oplus \\ \oplus\ \texttt{0x49} \cdot Z_j[14] \oplus \texttt{0xD7} \cdot Z_j[15] &\quad \texttt{0xB4} \cdot (K_j[9] \oplus X_{j+1}[9]) \oplus \\ &\quad \texttt{0x4E} \cdot (K_j[10] \oplus X_{j+1}[10]) \oplus \\ &\quad \texttt{0x7E} \cdot (K_j[11] \oplus X_{j+1}[11]) \oplus \\ &\quad \texttt{0xC0} \cdot (K_j[13] \oplus X_{j+1}[13]) \oplus \\ &\quad \texttt{0xDA} \cdot (K_j[14] \oplus X_{j+1}[14]) \oplus \\ &\quad \texttt{0xC5} \cdot (K_j[15] \oplus X_{j+1}[15]) \qquad (7.4) \end{aligned}$$

where, $W_j = K_j \oplus X_{j+1}$. Derivation of Eq. 7.3 is shown in Appendix B. Let,

$$P_j = \texttt{0xCA} \cdot Z_j[12] \oplus \texttt{0xAD} \cdot Z_j[13] \oplus \texttt{0x49} \cdot Z_j[14] \oplus \texttt{0xD7} \cdot Z_j[15] \qquad (7.5)$$

$$\begin{aligned} Q_j &= \texttt{0x94} \cdot X_{j+1}[8] \oplus \texttt{0xB4} \cdot X_{j+1}[9] \oplus \texttt{0x4E} \cdot X_{j+1}[10] \oplus \texttt{0x7E} \cdot X_{j+1}[11] \\ &\quad \oplus\ \texttt{0xC0} \cdot X_{j+1}[13] \oplus \texttt{0xDA} \cdot X_{j+1}[14] \oplus \texttt{0xC5} \cdot X_{j+1}[15] \qquad (7.6) \end{aligned}$$

$$\begin{aligned} Const &= \texttt{0x94} \cdot K_j[8] \oplus \texttt{0xB4} \cdot K_j[9] \oplus \texttt{0x4E} \cdot K_j[10] \oplus \texttt{0x7E} \cdot K_j[11] \oplus \\ &\quad \texttt{0xC0} \cdot K_j[13] \oplus \texttt{0xDA} \cdot K_j[14] \oplus \texttt{0xC5} \cdot K_j[15] \qquad (7.7) \end{aligned}$$

then, Eq. 7.4 can be rewritten as,

$$P_j = Q_j \oplus Const \qquad (7.8)$$

Eq. 7.8 will be used to establish the distinguishing property as shown next.

7.3.1 Distinguishing Property for Kalyna-128/256

Given, a list of 256 distinct bytes (M^0, M^1, ... , M^{255}), a function $f : \{0,1\}^{128} \mapsto \{0,1\}^{128}$ and a 120-bit constant T, we define a multiset v as follows :

$$
\begin{aligned}
C^i &= f(T \,||\, M^i), \text{where } (0 \leq i \leq 255) && (7.9)\\
u^i &= \texttt{0x94} \cdot C^i[8] \oplus \texttt{0xB4} \cdot C^i[9] \oplus \texttt{0x4E} \cdot C^i[10] \oplus \texttt{0x7E} \cdot C^i[11] \oplus \\
&\quad \texttt{0xC0} \cdot C^i[13] \oplus \texttt{0xDA} \cdot C^i[14] \oplus \texttt{0xC5} \cdot C^i[15] && (7.10)\\
v &= \{u^0 \oplus u^0, u^1 \oplus u^0, \ldots, u^{255} \oplus u^0\} && (7.11)
\end{aligned}
$$

Note that, ($T \,||\, M^0$, $T \,||\, M^1$, ... , $T \,||\, M^{255}$) forms a δ-list and atleast one element of v (i.e., $u^0 \oplus u^0$) is always zero.

Distinguishing Property. Let us consider $\mathcal{F}$ to be a family of permutations on 128-bit. Then, given any list of 256 distinct bytes (M^0, M^1, ... , M^{255}), the aim is to find how many multisets v (as defined above) are possible when, $f \xleftarrow{\$} \mathcal{F}$ and $T \xleftarrow{\$} \{0,1\}^{120}$.

In case, when $\mathcal{F}$ = family of all permutations on 128-bit and $f \xleftarrow{\$} \mathcal{F}$. Under such setting, since in the multiset v, we have 255 values (one element is always 0) that are chosen uniformly and independently from the set {0, 1, ..., 255 }, the total number of possible multisets v are atmost $\binom{2^8-1+2^8-1}{2^8-1} \approx 2^{505.17}$.

In case, when $\mathcal{F}$ = 6-full rounds of Kalyna-128/256 and $f \xleftarrow{\$} \mathcal{F}$. Here, $f \xleftarrow{\$} \mathcal{F}$ $\Leftrightarrow K \xleftarrow{\$} \{0,1\}^{256}$ and $f = E_K$. Let us consider the 6 inner rounds of Kalyna-128/256 as shown in Fig. 7.7. Here, C in Eq. 7.9 is represented by X_6 and Eq. 7.10 is defined as :

$$
\begin{aligned}
u^i &= \texttt{0x94} \cdot X_6^i[8] \oplus \texttt{0xB4} \cdot X_6^i[9] \oplus \texttt{0x4E} \cdot X_6^i[10] \oplus \texttt{0x7E} \cdot X_6^i[11] \oplus \\
&\quad \texttt{0xC0} \cdot X_6^i[13] \oplus \texttt{0xDA} \cdot X_6^i[14] \oplus \texttt{0xC5} \cdot X_6^i[15] && (7.12)
\end{aligned}
$$

It is to be noted that under this setting, for each i where $(0 \leq i \leq 255)$, Eq. 7.12 is same as Eq. 7.6 computed at round 5, i.e., $u^i = Q_5^i$. Now, we state the following *Observation 1.*

Observation 1. The multiset v is determined by the following 52 single byte parameters only :

- X_1^0[0 - 7] (8-bytes)
- X_2^0[0 - 15] (16-bytes)
- X_3^0[0 - 15] (16-bytes)
- X_4^0[0 - 3, 12 - 15] (8-bytes)

- $X_5^0[4 - 7]$ (4-bytes)

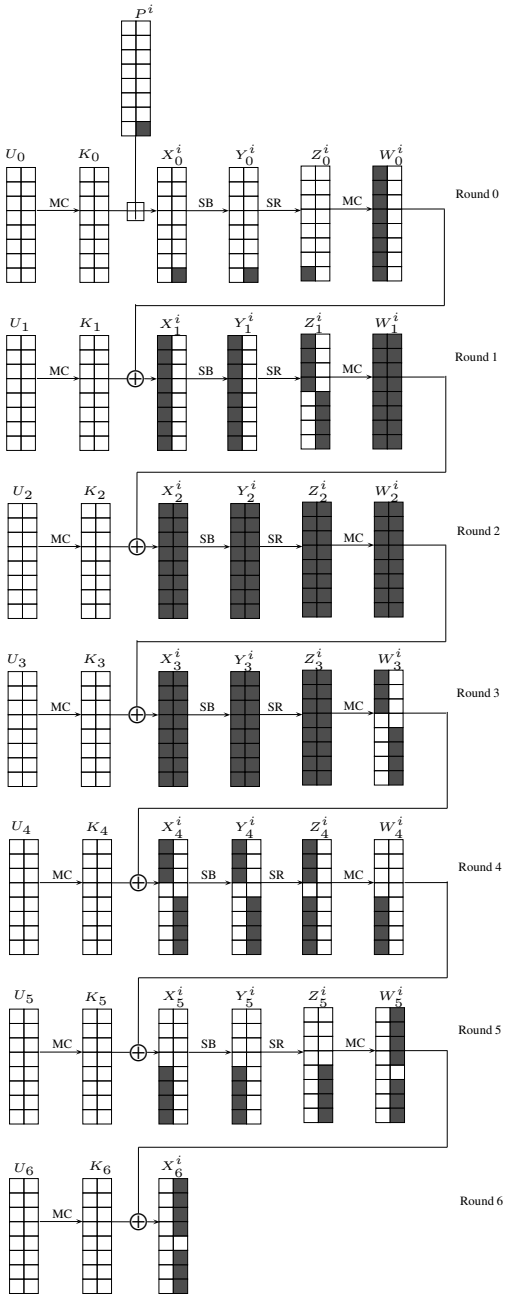


Figure 7.7: 6-Round distinguisher for Kalyna-128/256. P^i denotes $(T \mid\mid M^i)$ and X_j^i, Y_j^i, Z_j^i, W_j^i denote intermediate states corresponding to P^i in round j. The round subkeys K_j, where, $0 \leq j \leq 6$ are generated from the master key K.

Thus, the total number of possible multisets is $2^{52\times 8} = 2^{416}$ since each 52-byte value defines one sequence.

Proof. In round 0 (in Fig. 7.7), the set of differences $\{X_0^0[15] \oplus X_0^0[15], X_0^1[15] \oplus X_0^0[15], \ldots, X_0^{255}[15] \oplus X_0^0[15]\}$ (or, equivalently the set of differences at $X_0[15]$) is known to the attacker as there are exactly 256 differences possible. This is so, because in the plaintext we make the most significant byte as the active byte. Hence, when the pre-whitening key is added (columnwise), the carry-bit in the most significant bit is ignored limiting the possible values (and the differences) at $X_0[15]$ to 256 only. Since S-box is injective, exactly 256 values exist in the set $\{Y_0^0[15] \oplus Y_0^0[15], Y_0^1[15] \oplus Y_0^0[15], \ldots, Y_0^{255}[15] \oplus Y_0^0[15]\}$. As *Shift Row* (SR), *MixColumns* (MC) and *Add Round Key* (ARK) are linear operations, the set of differences at $X_1[0-7]$ will be known to the attacker.

Owing to the non-linearity of the S-box operation, the set of differences at Y_1 [0-7] cannot be computed to move forward. To alleviate this problem, it is sufficient to guess X_1^0[0-7], i.e., values of the active bytes of the first state (out of 256 states) at X_1 as it allows calculating the other X_1^i[0-7] states (where, $1 \leq \mathrm{i} \leq 255$) and cross SB layer in round 1. Since, SR, MC and ARK operations are linear, the set of differences at $X_2[0-15]$ is known. Continuing in a similar manner as discussed above, if the attacker guesses full states X_2^0 [0-15] and X_3^0 [0-15], then the set of differences at Z_3, i.e., $\{Z_3^0 \oplus Z_3^0, Z_3^1 \oplus Z_3^0, \ldots, Z_3^{255} \oplus Z_3^0\}$ can be easily computed. Now at this stage, she can easily calculate the set of differences at W_3 [0, 1, 2, 3, 12, 13, 14, 15] which is equal to the set of differences at X_4 [0, 1, 2, 3, 12, 13, 14, 15]. [4]. By guessing X_4^0 [0, 1, 2, 3, 12, 13, 14, 15], the attacker can cross the SB layer in round 4 and calculate the set of differences at W_4 [4, 5, 6, 7]. By guessing X_5^0 [4, 5, 6, 7], the attacker can obtain the set of values $\{Z_5^0[12-15], Z_5^1[12-15], \ldots, Z_5^{255}[12-15]\}$. Using these, she can compute P_5^i at Z_5^i as $P_5^i = CA_x \cdot Z_5^i[12] \oplus AD_x \cdot Z_5^i[13] \oplus 49_x \cdot Z_5^i[14] \oplus D7_x \cdot Z_5^i[15]$ (according to Eq. 7.5) and thus the set $\{P_5^0 \oplus P_5^0, P_5^0 \oplus P_5^1, \ldots, P_5^{255} \oplus P_5^0\}$. Since, according to Eq. 7.8, $P_j^i \oplus P_j^0 = (Q_j^i \oplus Const) \oplus (Q_j^0 \oplus Const) = Q_j^i \oplus Q_j^0$ and $u^i = Q_5^i$ (mentioned above), the attacker can easily calculate the multiset $v = \{Q_5^0 \oplus Q_5^0, Q_5^1 \oplus Q_5^0, \ldots, Q_5^{255} \oplus Q_5^0\}$. This shows that the multiset v depends on 52 parameters and can take 2^{416} possible values. □

Since, there are 2^{416} possible multisets, if we precompute and store these values in a hash table then the precomputation complexity goes higher than brute force for Kalyna-128/256. In order to reduce the number of multisets, we apply the Differential Enumeration technique suggested by Dunkelman et al. in [71] and improved by Derbez et al. in [61]. We call the improved version proposed in [61] as *Refined Differential Enumeration*.

Refined Differential Enumeration. The basic idea behind this technique is to choose a δ-set such that several of the parameters mentioned in *Observation 1* equal

[4] In Fig. 7.7, byte 3 in states W_3, X_4, Y_4 and Z_4 have not been colored grey for a purpose which will be cleared when we reach *Observation 2*

some pre-determined constants. To achieve so, we first construct a 6-round truncated differential trail in round 0 - round 5 (as shown in Fig. 7.8) where, the input difference is non-zero at one byte and output difference is non zero in 7 bytes.

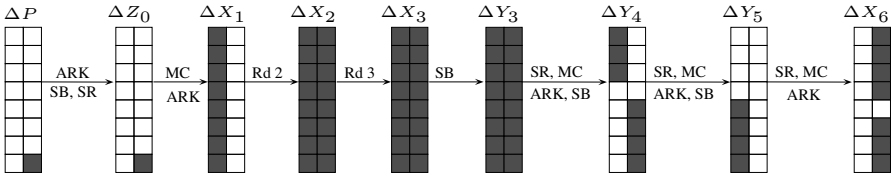


Figure 7.8: 6-Round Truncated Differential in Kalyna-128/256

The probability of such a trail is 2^{-112} as follows: the one byte difference at $\Delta P[15]$ and correspondingly at $\Delta X_0[15]$ propagates to 8-byte difference in $\Delta X_1[0-7]$ and 16-byte difference in $\Delta X_2[0-15]$ and further till $\Delta Z_3[0-15]$ with probability 1. Next, the probability that 16-byte difference in $\Delta Z_3[0-15]$ propagates to 7-byte difference in $\Delta W_3[0-2, 12-15]$ $(= \Delta X_4[0-2, 12-15])$ is 2^{-72}. This 7-byte difference in ΔX_4 propagates to 4-byte difference in $\Delta W_4[4-7]$ followed by 7-byte difference in $\Delta W_5[8-11, 13-15]$ with a probability of 2^{-32} and 2^{-8} respectively. Thus, the overall probability of the differential from ΔP to ΔZ_5 is $2^{-(72+32+8)} = 2^{-112}$.

In other words, we require 2^{112} plaintext pairs to get a right pair. Once we get a right pair, say (P^0, P^1), we state the following *Observation 2* using this right pair.

Observation 2. Given a right pair (P^0, P^1) that follows the truncated differential trail shown in Fig. 7.8, the 52 parameters corresponding to P^0, mentioned in Observation 1 can take one of atmost 2^{224} fixed 52-byte values (out of the total 2^{416} possible values), where each of these 2^{224} 52-byte values are defined by each of the 2^{224} values of the following 39 parameters:

- $\Delta Z_0[7]$ (1-byte)
- $X_1^0[0-7]$ (8-bytes)
- $Y_3^0[0-15]$ (16-bytes)
- $Y_4^0[0-3, 12-15]$ (8-bytes)
- $Y_5^0[5-7]$ (3-bytes)
- $\Delta Z_5[12-14]$ (3-bytes)

Proof. Given a right pair (P^0, P^1), the knowledge of these 39 new parameters allows us to compute all the differences shown in Fig. 7.8. This is so because the knowledge of $\Delta Z_0[7]$ allows us to compute $\Delta X_1[0-7]$. Then, if the values of $X_1^0[0-7]$ are known, one can compute the corresponding $X_1^1[0-7]$ and cross the S-box layer in round 1 to get ΔX_2.

From the bottom side, it can be seen that $\Delta W_5[12] = \Delta Z_5[8] = \Delta Z_5[9] = \Delta Z_5[10] = \Delta Z_5[11] = 0$. Thus, if $\Delta Z_5[12, 13, 14]$ are known, then using *Property 2* (as 8 bytes are known), we can deduce $\Delta Z_5[15]$ (and ΔW_5 [8-11, 13-15]). Knowledge of $\Delta Z_5[8-15]$ allows us to to compute $\Delta Y_5[4-7]$. Then, by guessing $Y_5^0[5-7]$, we can determine the corresponding $Y_5^1[5-7]$ and compute $\Delta X_5[5-7]$ (and $\Delta W_4[5-7]$). Now again, we know that $\Delta W_4[0] = \Delta W_4[1] = \Delta W_4[2] = \Delta W_4[3] = \Delta Z_4[3] = 0$. Using *Property 2* (as 8 bytes are known), we can deduce $\Delta W_4[4]$ (and $\Delta Z_4[0-2, 4-7]$). This allows us to compute $\Delta X_5[4]$ as well. Since we already know $\Delta Y_5[4]$ (from $\Delta Z_5[12]$ guessed previously), using *Property 1a.*, the possible values of $X_5[4]$ and $Y_5[4]$ can be computed.

Now, knowledge of $\Delta Z_4[0-7]$ allows us to compute $\Delta Y_4[0-3, 12-15]$. By guessing $Y_4^0[0-3, 12-15]$, we can obtain $\Delta Y_3[0-15]$. Using the value of $Y_3^0[0-15]$, we can compute ΔY_2. Then using *Property 1a.*, the possible values of X_2^0 and Y_2^0 can be computed. At this stage, the total possible values of these 39 parameters are $2^{39\times 8} = 2^{312}$.

Key Sieving. However, for each value of this 39-byte parameter, the following key bytes - $U_2[0-3, 12-15]$, K_3, $K_4[0-3, 12-15]$ and $K_5[4-7]$ can be deduced as follows:

1. Knowledge of $X_1^0[0-7]$ allows us to compute the corresponding $Z_1^0[0-3, 12-15]$. Xoring these values with $X_2^0[0-3, 12-15]$ helps us in deducing $U_2[0-3, 12-15]$.

2. Knowledge of X_2^0 allows us to compute the corresponding W_2^0. Xoring W_2^0 with X_3^0 helps us in deducing K_3.

3. Knowledge of X_3^0 and $X_4^0[0-3, 12-15]$ (from $Y_4^0[0-3, 12-15]$) can be used to deduce $K_4[0-3, 12-15]$.

4. Knowledge of $X_4^0[0-3, 12-15]$ and $X_5^0[4-7]$ (from $Y_5^0[4-7]$) helps us in deducing $K_5[4-7]$.

Now, according to the key schedule algorithm of Kalyna-128/256, from K_3, we can compute K_2 (according to Eq. 7.1) which allows us to compute the corresponding U_2. Thus, by comparing the computed $U_2[0-3, 12-15]$ with the deduced $U_2[0-3, 12-15]$, a sieve of 8-bytes (since matching probability is 2^{-64}) can be applied to eliminate the wrong guesses. Similarly, again from Eq. 7.1, knowledge of $K_5[4-7]$ allows us to compute K_4 [12], K_4 [13] and K_4 [14] as $K_4[12] = K_5[5]$, $K_4[13] = K_5[6]$ and $K_4[14] = K_5[7]$. This allows us a filtering of further 3-bytes. Thus by key sieving, the total possible

guesses of 39-byte parameter reduces from $2^{39\times8}$ to $2^{(39-(8+3))\times8} = 2^{28\times8} = 2^{224}$. □

Using *Observation 1* and *Observation 2*, we state the following third *Observation 3*:

Observation 3. Given (M^0, M^1, ... , M^{255}) and $f \xleftarrow{\$} \mathcal{F}$ and $T \xleftarrow{\$} \{0,1\}^{120}$, such that $T \mid\mid M^0$ and $T \mid\mid M^j$, (where, $j \in \{ 1, \ldots , 255 \}$) is a right pair that follows the differential trail shown in Fig. 7.8, atmost 2^{224} multisets v are possible.

Proof. From *Observation 1*, we know that each 52-byte parameter defines one multiset and *Observation 2* restricts the possible values of these 52-byte parameters to 2^{224}. Thus, atmost 2^{224} multisets are only possible for Kalyna-128/256. □

As the number of multisets in case of 128-bit random permutation ($= 2^{505.17}$) is much higher than 6-round Kalyna-128/256 ($= 2^{224}$), a valid distinguisher is constructed.

7.4 Key Recovery Attack on 9 Round Kalyna-128/256

In this section, we use our Observation 3 to launch meet-in-the-middle attack on 9-round Kalyna-128/256 to recover the key. The distinguisher is placed in round 0 to round 5, i.e, the set of plaintexts is considered as the δ-list with byte 15 being the active byte and the multiset sequence being checked at X_6 (as shown in Fig. 7.9). Three rounds are added at the bottom of the 6-round distinguisher. The attack consists of the following three phases:

7.4.1 Precomputation Phase

In this phase, we build a lookup table T to store 2^{224} sequences to be used for comparison in the online phase. The construction of this table requires us to create two more hash tables (T_0 and T_1) in the intermediate steps. The entire procedure is as follows:

1. For each K_3

   - We guess $\Delta Z_1[0-3, 12-15]||\Delta X_4[0-2, 12-15]$ to compute the difference ΔX_2 and ΔY_3 respectively. We resolve (ΔX_2 - ΔY_3) using *Property 1b* to compute the corresponding $X_2||X_3$. We then deduce K_2 from K_3 and compute the corresponding value of $Z_1[0-3, 12-15]$. Using the guessed value of $\Delta Z_1[0-3, 12-15]$ and the computed value of $Z_1[0-3, 12-15]$, we compute $\Delta Z_0[0-7]$. If $\Delta Z_0[0-6] = 0$ (which happens with a probability of 2^{-56}), we store the corresponding $X_1[0-7]||\Delta Z_1[0-3, 12-15]||X_2||X_3||W_3[12-$

$14]||\Delta X_4[0-2, 12-15]$ at index K_3 in table T_0. There are about 2^{64} entries for each index.

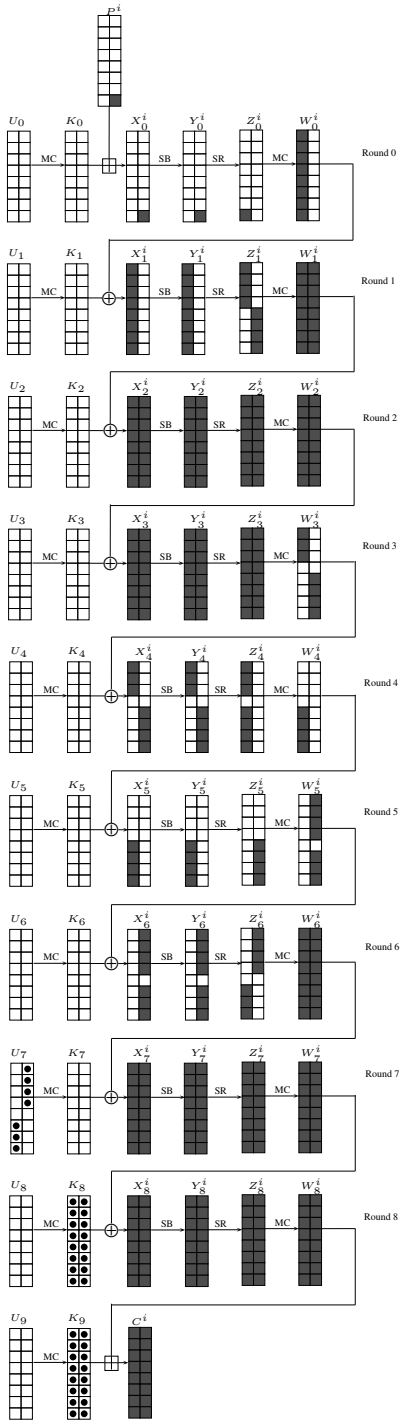


Figure 7.9: 9-round attack on Kalyna-128/256. The subkey bytes guessed are shown dotted.

2. For each guess of $\Delta Z_5[12-14]$

   - We compute $\Delta Z_5[15]$ using *Property 2*.
   - We guess $Y_5[5-7]$, compute $X_5[5-7]$ and $\Delta X_5[0-3, 5-7]$ where, $\Delta X_5[0-3] = 0$. Since, $\Delta X_5[0-3, 5-7] = \Delta W_4[0-3, 5-7]$ and we know that $\Delta Z_4[3] = 0$, thus we can compute $\Delta X_5[4]$ $(= \Delta W_4[4])$ and $\Delta Z_4[0-2, 4-7]$ again using *Property 2*. Since $\Delta Y_5[4]$ is known from $\Delta Z_5[12]$, we can resolve ($\Delta X_5[4]$-$\Delta Y_5[4]$) to get $X_5[4]$.
   - We guess $Y_4[0-3, 12-15]$ and compute corresponding $X_4[0-3, 12-15]$ in the backward direction and $W_4[4-7]$ in the forward direction. This allows us to calculate $K_5[4-7]$ and deduce the corresponding $K_4[12-14]$. We use this to compute $W_3[12-14]$.
   - We store $X_4[0-3, 12-15]||X_5[4-7]$ at index value $W_3[12-14]||\Delta X_4[0-2, 12-15]$ in table T_1. There are about 2^{32} entries for each index.

3. For each of the 2^{128} index of K_3 in table T_0, we have 2^{64} entries of $W_3[12-14]||\Delta X_4[0-2, 12-15]$ and corresponding to each of these we have 2^{32} entries of $X_4[0-3, 12-15]||X_5[4-7]$ in table T_1. So in all, after merging T_0 and T_1, we get $2^{128+64+32} = 2^{224}$ unique set of 39-byte parameters, that are required to construct the multiset v.

4. For each of these 2^{224} 39-byte parameters, we calculate the corresponding 52-byte parameters for all the elements of the δ-list and compute the multiset $v = \{u^0 \oplus u^0, u^1 \oplus u^0, \ldots, u^{255} \oplus u^0\}$. We store the multiset along with the 52-byte parameters in the table T.

The time complexity to construct [5] $T_0 = 2^{(16+8+7)\times 8} \times 2^{-2.17} = 2^{245.83}$. The time complexity to construct $T_1 = 2^{(3+3+8)\times 8} \times 2^{-2.17} = 2^{109.83}$. The time complexity to merge T_0 and $T_1 = 2^{128+64+32} = 2^{224}$. Finally, the time complexity to construct $T = 2^{224} \times 2^8 \times 2^{-0.58} = 2^{231.41}$.

7.4.2 Online Phase

In this phase, we extend the distinguisher described in Section 7.3, by adding 3 more rounds at the bottom (as shown in Fig. 7.9). The steps of the online phase are as follows:

1. We encrypt 2^{97} structures of 2^8 plaintexts each where byte 15 takes all possible values and rest of the bytes are constants. We store the corresponding ciphertexts in the hash table.

[5] The normalization factor $2^{-2.17}$ is calculated by finding the ratio number of rounds encrypted/decrypted to 9 (i.e., the number of rounds of Kalyna considered in this work). Similarly all other normalization factors have been calculated.

2. For each of the 2^{112} (P_0, P_0') plaintext pairs, do the following:

   - We guess 2^{128} values of K_9 and deduce the corresponding values of K_8 from K_9. We decrypt each of the ciphertext pairs through 2 rounds, to get X_7 and ΔX_7. Then, we deduce the corresponding ΔW_6 and ΔZ_6.
   - We filter out the keys, which do not give zero difference at $\Delta Z_6[0-4, 12-15]$. 2^{56} key guesses are expected to remain.
   - We pick one member of the pair, say P_0, create the δ-list by constructing the rest of the 255 plaintexts as $P_i = P_0 \oplus i$, where, $1 \leq i \leq 255$ and get their corresponding ciphertexts.
   - For each remaining 2^{56} key guesses of K_8 and K_9, we guess $U_7[5-11]$, compute the corresponding $Z_6[5-11]$ and $Y_6[8-11, 13-15]$ and then obtain the multiset $\{ u^0 \oplus u^0, u^1 \oplus u^0, \ldots, u^{255} \oplus u^0\}$.
   - We check whether this multiset exists in the precomputation table T or not. If not, then we discard the corresponding guesses.

The probability for a wrong guess to pass the test is $2^{224} \times 2^{-467.6} = 2^{-243.6}$. [6] Since we try only $2^{112+56} = 2^{168}$ multisets, only the right subkey should verify the test.

7.4.3 Recovering the remaining Subkey bytes

The key schedule algorithm of Kalyna does not allow recovery of master key from any subkey better than brute-force [138]. However, knowledge of all round keys enables encryption/decryption. We follow a similar approach as described in [13] to recover all the round subkeys. When a match with a multiset is found using a given plaintext-ciphertext pair, we choose one of the ciphertexts and perform the following steps:

1. We already know the corresponding K_8 and K_9 and U_7[5-11].
2. We guess the remaining 9 bytes of U_7, and deduce the corresponding 2^{72} values of K_7 and K_6.
3. For each 2^{72} guesses of (K_7, K_6), from X_7 we compute X_5. We discard the key guesses for which $X_5[4-7]$ does not match with the values of $X_5[4-7]$ obtained from the corresponding matched multiset in the pre-computation table.
4. For the remaining $2^{72-32} = 2^{40}$ guesses of (K_9, K_8, K_7, K_6), we guess 2^{128} values of K_5. We deduce X_4 and discard the key guesses for which X_4[0-2, 12-15] does not match with the values obtained corresponding to the correct multiset sequence from the precomputation table. From a total of $2^{128+40} = 2^{168}$ key guesses, 2^{112} key guesses are expected to remain.

[6] Note that the probability of randomly having a match is $2^{-467.6}$ and not $2^{-505.17}$ since the number of ordered sequences associated to a multiset is not constant [71].

5. We deduce K_4 from K_5 for the remaining key guesses and compute X_3. We compare this to the value obtained from the precomputation table corresponding to the correct multiset sequence and discard those that do not match. Only one value of (K_9, K_8, K_7, K_6, K_5, K_4) is expected to remain.

6. One value of K_3 and K_2 corresponding to the matching sequence is already known from the pre-computation table. We deduce X_1 for the remaining one value of (K_9, K_8, K_7, K_6, K_5, K_4, K_3, K_2).

7. We guess 2^{128} values of K_1, deduce K_0 and compute the plaintext. We compare this to the plaintext corresponding to ciphertext being decrypted. We are left with only one value of (K_9, K_8, K_7, K_6, K_5, K_4, K_3, K_2, K_1, K_0).

Complexities. The time complexity of the precomputation phase is dominated by step 1 and is $2^{248} \times 2^{-2.17} = 2^{245.83}$ Kalyna-128/256 encryptions. The time complexity of the online phase is dominated by step 2 (part 1) and is $2^{112} \times 2^{128} \times 2^{-2.17} = 2^{233.83}$. The time complexity of the Subkey recovery phase is dominated by step 4 which is $2^{168} \times 2^{-3.17} = 2^{164.83}$. Clearly the time complexity of the whole attack is dominated by the time complexity of the precomputation phase, i.e., $2^{245.83}$. It was shown in [61] that each 256-byte multiset requires 512-bits space. Hence, to store each entry in table T, we require 512-bits to store the multiset and $52 \times 8 = 416$-bits to store the 52-byte parameters, i.e., a total of 928-bits ($= 2^{9.86}$). Therefore, the memory complexity of this attack is $2^{224} \times 2^{9.86-7} = 2^{226.86}$ Kalyna 128-bit blocks. The data complexity of this attack is 2^{105} plaintexts.

7.5 Construction of distinguisher for 6-Round Kalyna-256/512

In this section, we construct a distinguisher for the 6-inner rounds of Kalyna-256/512. The distinguisher construction details are similar to Kalyna-128/256 (discussed in Sec. 7.3) except the fact that here instead of counting multisets, we count 256-byte ordered sequences. The reason for opting ordered sequences would be discussed in Sec. 7.6.

We first establish the following relation for Kalyna-256/512. According to *Property 2*, it is possible to construct an equation using any 12 out of 16 input-output bytes in the Kalyna MixColumns operation. For any round j, where, $0 \leq j \leq 8$:

$$\begin{aligned} Z_j[8] \oplus Z_j[9] \oplus Z_j[12] \oplus Z_j[13] \;=\; & EA_x \cdot W_j[8] \oplus 54_x \cdot W_j[9] \oplus 7D_x \cdot W_j[10] \oplus \\ & C3_x \cdot W_j[11] \oplus E0_x \cdot W_j[12] \oplus 5E_x \cdot W_j[13] \oplus \\ & 7D_x \cdot W_j[14] \oplus C3_x \cdot W_j[15] \end{aligned} \tag{7.13}$$

Derivation of Eq. 7.13 is shown in Appendix C. Similar to as shown in Section 7.3, since, $W_j = K_j \oplus X_{j+1}$, if

$$\begin{aligned} P_j &= Z_j[8] \oplus Z_j[9] \oplus Z_j[12] \oplus Z_j[13] && (7.14) \\ Q_j &= EA_x \cdot X_{j+1}[8] \oplus 54_x \cdot X_{j+1}[9] \oplus 7D_x \cdot X_{j+1}[10] \oplus \\ &\quad C3_x \cdot X_{j+1}[11] \oplus E0_x \cdot X_{j+1}[12] \oplus 5E_x \cdot X_{j+1}[13] \oplus \\ &\quad 7D_x \cdot X_{j+1}[14] \oplus C3_x \cdot X_{j+1}[15] && (7.15) \\ Const &= EA_x \cdot K_j[8] \oplus 54_x \cdot K_j[9] \oplus 7D_x \cdot K_j[10] \oplus C3_x \cdot K_j[11] \oplus \\ &\quad E0_x \cdot K_j[12] \oplus 5E_x \cdot K_j[13] \oplus 7D_x \cdot K_j[14] \oplus C3_x \cdot K_j[15] && (7.16) \end{aligned}$$

then, Eq. 7.13 can be rewritten as,

$$P_j = Q_j \oplus Const \quad (7.17)$$

7.5.1 Construction of 6-round distinguisher for Kalyna-256/512

Given a list of 256 distinct bytes (M^0, M^1, ... , M^{255}), a function $f : \{0,1\}^{256} \mapsto \{0,1\}^{256}$ and a 248-bit constant T, we define an ordered sequence ov as follows:

$$\begin{aligned} C^i &= f(T \,||\, M^i), \text{where } (0 \leq i \leq 255) && (7.18) \\ ou^i &= EA_x \cdot C^i[8] \oplus 54_x \cdot C^i[9] \oplus 7D_x \cdot C^i[10] \oplus C3_x \cdot C^i[11] \oplus \\ &\quad E0_x \cdot C^i[12] \oplus 5E_x \cdot C^i[13] \oplus 7D_x \cdot C^i[14] \oplus C3_x \cdot C^i[15] && (7.19) \\ ov &= \{ou^0 \oplus ou^0, ou^1 \oplus ou^0, \ldots, ou^{255} \oplus ou^0\} && (7.20) \end{aligned}$$

Note that, ($T \,||\, M^0$, $T \,||\, M^1$, ... , $T \,||\, M^{255}$) forms a δ-list and the first element of ov (i.e., $ou^0 \oplus ou^0$) is always zero.

Distinguishing Property. Let us consider $\mathcal{F}$ to be a family of permutations on 256-bit. Then, given any list of 256 distinct bytes (M^0, M^1, ... , M^{255}), the aim is to find how many ordered sequences ov (as defined above) are possible when, $f \xleftarrow{\$} \mathcal{F}$ and $T \xleftarrow{\$} \{0,1\}^{248}$.

In case, when $\mathcal{F}$ = family of all permutations on 256-bit and $f \xleftarrow{\$} \mathcal{F}$. Under such setting, since, ov is a 256-byte ordered sequence in which the first byte is always zero and the rest 255 bytes are chosen uniformly and independently from the set {0, 1, ... , 255}, the total possible values of ov are $(256)^{255} = 2^{2040}$.

In case, when $\mathcal{F}$ = 6-full rounds of Kalyna-256/512 and $f \xleftarrow{\$} \mathcal{F}$. Here, $f \xleftarrow{\$} \mathcal{F} \Leftrightarrow K \xleftarrow{\$} \{0,1\}^{512}$ and $f = E_K$. Let us consider the first 6 inner rounds of Kalyna-256/512 as shown in Fig. 7.10. Here, C in Eq. 7.18 is represented by X_6 and Eq. 7.19 is defined as :

$$ou^i = EA_x \cdot X_6^i[8] \oplus 54_x \cdot X_6^i[9] \oplus 7D_x \cdot X_6^i[10] \oplus C3_x \cdot X_6^i[11] \oplus E0_x \cdot X_6^i[12] \oplus 5E_x \cdot X_6^i[13] \oplus 7D_x \cdot X_6^i[14] \oplus C3_x \cdot X_6^i[15] \quad (7.21)$$

It is to be noted that here, for each i where, $(0 \leq i \leq 255)$, Eq. 7.21 is same as Eq. 7.15 computed at round 5, i.e., $ou^i = Q_5^i$. Now, we state the following *Observation 4*.

Observation 4. The ordered sequence ov is determined by the following 93 single byte parameters only :

- $X_0^0[31]$ (1-byte)
- $X_1^0[16-23]$ (8-bytes)
- $X_2^0[0-31]$ (32-bytes)
- $X_3^0[0-31]$ (32-bytes)
- $X_4^0[2,3,6,7,8,9,12,13,18,19,22,23,24,25,28,29]$ (16-bytes)
- $X_5^0[8,9,28,29]$ (4-bytes)

Thus, the total number of ordered sequences is $2^{93\times 8} = 2^{744}$ since each 93-byte value defines one sequence.

Proof. In round 0 (in Fig. 7.10), the ordered list of differences at $\{X_0^0[31] \oplus X_0^0[31], X_0^1[31] \oplus X_0^0[31], \ldots, X_0^{255}[31] \oplus X_0^0[31]\}$ (or, equivalently the list of differences at $X_0[31]$) is known to the attacker as the list [7] of differences at $X_0[31]$ = list of differences at $P[31]$, i.e., $P^i[31] \oplus P^0[31] = X_0^i[31] \oplus X_0^0[31]$ for $(1 \leq i \leq 255)$. This is so, because in the plaintext, we make the most significant byte as the active byte. Hence, when the pre-whitening key is added (columnwise), the carry-bit in the most significant bit is ignored, thus converting the addition operation to xor operation. Since the value of $X_0^0[31]$ is known, the attacker can compute the other $X_0^i[31]$. This allows her to cross the *SB* and *SR* layer in round 0. Since, *MixColumns* (MC) and *Add Round Key* (ARK) are linear operations, the list of differences at $X_1[16-23]$ can be computed by the attacker.

Owing to the non-linearity of the S-box operation, the list of differences at $Y_1[16-23]$ cannot be computed to move forward. To allievate this problem, it is sufficient to guess $X_1^0[16-23]$ as it allows calculating other $X_1^i[16-23]$ states and cross SB layer in round 1. Since, SR, MC and ARK operations are linear, the list of differences at $X_2[0-31]$ is known. Continuing in a similar manner as discussed above, if the attacker guesses

[7] From now onwards, list denotes an ordered list

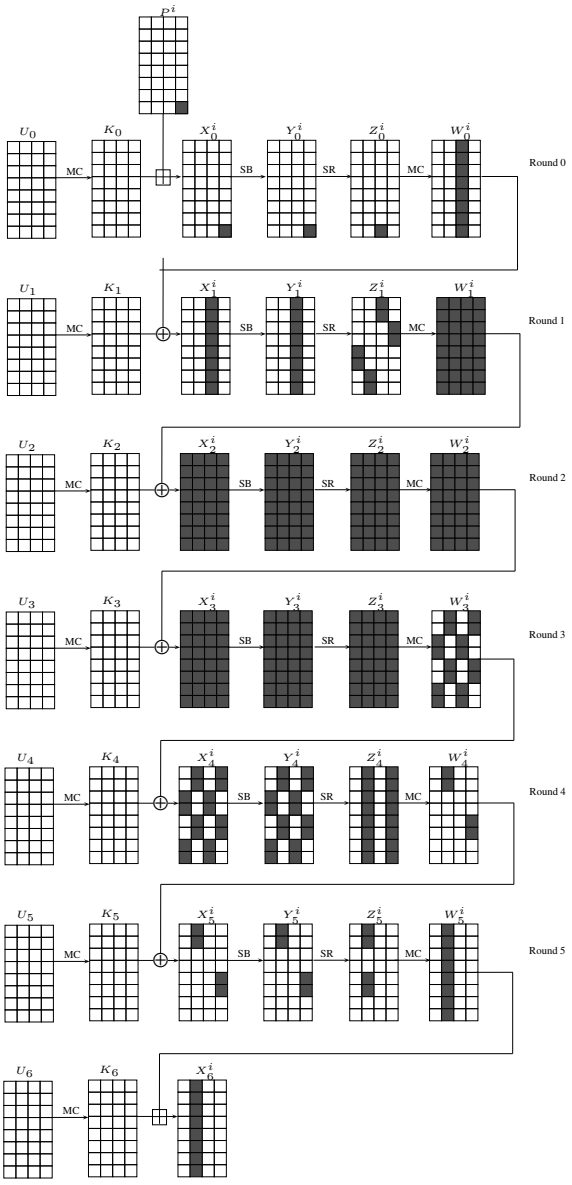


Figure 7.10: 6-Round distinguisher in Kalyna-256/512 . P^i denotes $(T \mid\mid M^i)$ and X_j^i, Y_j^i, Z_j^i, W_j^i denote intermediate states corresponding to P^i in round j. The round subkeys K_j, where, $0 \leq j \leq 6$ are generated from the master key K.

full states $X_2^0[0-31]$ and $X_3^0[0-31]$, then the list of differences at Z_3, i.e., $\{Z_3^0 \oplus Z_3^0$, $Z_3^1 \oplus Z_3^0$, ..., $Z_3^{255} \oplus Z_3^0\}$ can be easily computed.

This also allows her to calculate the list of differences at X_4 [2, 3, 6, 7, 8, 9, 12, 13, 18, 19, 22, 23, 24, 25, 28, 29]. By guessing X_4^0 [2, 3, 6, 7, 8, 9, 12, 13, 18, 19, 22, 23, 24, 25, 28, 29], the attacker can cross the SB layer in round 4 and calculate the list of differences at $X_5[8, 9, 28, 29]$. By guessing X_5^0 [8, 9, 28, 29], the attacker can obtain

the list of values $\{Z_5^0[8,9,12,13],\ Z_5^1[8,9,12,13],\ \ldots,\ Z_5^{255}[8,9,12,13]\}$. Using these, she can compute P_5^i at Z_5^i using Eq. 7.14 and thus the list $\{P_5^0 \oplus P_5^0,\ P_5^0 \oplus P_5^1,\ \ldots,\ P_5^{255} \oplus P_5^0\}$. Since, according to Eq. 7.17, $P_5^i \oplus P_5^0 = Q_5^i \oplus Q_5^0$ and $ou^i = Q_5^i$ (mentioned above), the attacker can easily calculate the ordered sequence $ov = \{Q_5^0 \oplus Q_5^0,\ Q_5^1 \oplus Q_5^0,\ \ldots,\ Q_5^{255} \oplus Q_5^0\}$. This shows that ov depends on 93 parameters and can take 2^{744} possible values. $\square$

Since, there are 2^{744} possible ordered sequences, if we precompute and store these values in a hash table, then the precomputation complexity goes higher than brute force for Kalyna-256/512. In order to reduce the number of ordered sequences, we apply the Refined Differential Enumeration technique as follows:

Number of admissible ordered sequences. Consider the 6-round truncated differential trail in round 0 - round 5 (as shown in Fig. 7.10) where, the input difference is non-zero at one byte and output difference is non zero in 8 bytes. The probability of such a trail is 2^{-224} as follows: the one byte difference at $\Delta P[31]$ propagates to 32-byte difference in $\Delta Z_3[0-31]$ with probability 1. Next, the probability that 32-byte difference in $\Delta Z_3[0-31]$ propagates to 16-byte difference in $\Delta X_4[2,3,6,7,8,9,12,13,18,19,22,23,24,25,28,29]$ is 2^{-128}. This 16-byte difference in ΔX_4 propagates to 4-byte difference in $\Delta W_4[8,9,28,29]$ followed by 8-byte difference in $\Delta W_5[8-15]$ with a probability of 2^{-96}. Thus, the overall probability of the differential trail from ΔP to ΔW_5 is $2^{-(128+96)} = 2^{-224}$.

In other words, we require 2^{224} plaintext pairs to get a right pair. Once, we get a right pair, say $(P^0,\ P^1)$, we state the following *Observation 5* using this right pair.

Observation 5. Given a right pair (P^0, P^1) that follows the truncated differential trail $(\Delta P \rightarrow \Delta W_5)$, the 93 parameters corresponding to P^0, mentioned in Observation 4 can take one of atmost 2^{440} fixed 93-byte values (out of the total 2^{744} possible values), where each of these 2^{440} 93-byte values are defined by each of the 2^{440} values of the following 66 parameters:

- $Y_0^0[31]$ (1-byte)
- $\Delta Z_0[23]$ (1-byte)
- $X_1^0[16-23]$ (8-bytes)
- $Y_3^0[0-31]$ (32-bytes)
- $Y_4^0[2,3,6,7,8,9,12,13,18,19,22,23,24,25,28,29]$ (16-bytes)
- $Y_5^0[8,9,28,29]$ (4-bytes)
- $\Delta Z_5[8,9,12,13]$ (4-bytes)

Proof. Given a right pair (P^0, P^1), the knowledge of these 66 new parameters allows us to compute all the differences shown in Fig. 7.10 as follows. Knowledge of $Y_0^0[31]$ allows us to compute $X_0^0[31]$. Knowing $\Delta Z_0[23]$ allows one to compute the difference $\Delta X_1[16-23]$. Then, if the values of $X_1^0[16-23]$ are known, one can compute the corresponding $X_1^1[16-23]$ and compute ΔX_2.

From the bottom side, knowing $\Delta Z_5[8, 9, 12, 13]$ allows one to compute $\Delta Y_5[8, 9, 28, 29]$. Knowledge of $Y_5^0[8, 9, 28, 29]$ allows one to compute $Y_5^1[8, 9, 28, 29]$, cross the SB layer in round 5 and obtain $\Delta Y_4^0[2, 3, 6, 7, 8, 9, 12, 13, 18, 19, 22, 23, 24, 25, 28, 29]$. Proceeding in a similar manner, knowing $Y_4^0[2, 3, 6, 7, 8, 9, 12, 13, 18, 19, 22, 23, 24, 25, 28, 29]$ and $Y_3^0[0-31]$ allows one to compute $\Delta Y_2^0[0-31]$. Then, using *Property 1a.*, the possible values of X_2^0 and Y_2^0 can be computed. At this stage, the total possible values of these 65 parameters are $2^{66\times 8} = 2^{528}$.

Key Sieving. However, for each value of this 66-byte parameter, the following key bytes - $U_2[4, 5, 14, 15, 16, 17, 26, 27]$, K_3, $K_4[2, 3, 6, 7, 8, 9, 12, 13, 18, 19, 22, 23, 24, 25, 28, 29]$ and $K_5[8, 9, 28, 29]$ can be deduced as follows:

1. Knowledge of $X_1^0[16-23]$ allows us to compute the corresponding $Z_1^0[4, 5, 14, 15, 16, 17, 26, 27]$. Xoring these values with $X_2^0[4, 5, 14, 15, 16, 17, 26, 27]$ helps us in deducing $U_2[4, 5, 14, 15, 16, 17, 26, 27]$.

2. Knowledge of X_2^0 and Y_3^0 helps us in deducing K_3.

3. Knowledge of Y_3^0 and $Y_4^0[2, 3, 6, 7, 8, 9, 12, 13, 18, 19, 22, 23, 24, 25, 28, 29]$ can be used to deduce $K_4[2, 3, 6, 7, 8, 9, 12, 13, 18, 19, 22, 23, 24, 25, 28, 29]$.

4. Knowledge of $Y_4^0[2, 3, 6, 7, 8, 9, 12, 13, 18, 19, 22, 23, 24, 25, 28, 29]$ and $Y_5^0[8, 9, 28, 29]$ helps us in deducing $K_5[8, 9, 28, 29]$.

Now, according to the key schedule algorithm of Kalyna-128/256 from K_3, we can compute K_2 (according to Eq. 7.1) which allows us to compute the corresponding U_2. Thus, by comparing the computed $U_2[4, 5, 14, 15, 16, 17, 26, 27]$ with the deduced $U_2[4, 5, 14, 15, 16, 17, 26, 27]$, a sieve of 8-bytes (since matching probability is 2^{-64}) can be applied. Similarly, knowledge of $K_4[2, 3, 6, 7, 8, 9, 12, 13, 18, 19, 22, 23, 24, 25, 28, 29]$ allows us to compute K_5 [8, 28, 29] which can then be matched with the deduced $K_5[8, 28, 29]$. This allows us a filtering of further 3-bytes. Thus, by key sieving, a total of 11-byte filtering can be applied and the possible guesses of 66-byte parameter reduces from $2^{66\times 8}$ to $2^{(66-11)\times 8} = 2^{55\times 8} = 2^{440}$. □

Using *Observation 4* and *Observation 5*, we state the following third *Observation 6*:

Observation 6. Given (M^0, M^1, ... , M^{255}) and $f \xleftarrow{\$} \mathcal{F}$ and $T \xleftarrow{\$} \{0,1\}^{248}$, such that $T \,||\, M^0$ and $T \,||\, M^j$, (where, $j \in \{ 1, \ldots, 255 \}$) is a right pair that follows the differential trail shown in Fig. 7.10, atmost 2^{440} multisets v are possible.

Proof. From *Observation 4*, we know that each 93-byte parameter defines one ordered sequence and *Observation 5* restricts the possible values of these 93-byte parameters to 2^{440}. Thus, atmost 2^{440} ordered sequences are only possible for Kalyna-256/512. □

As the number of ordered sequences in case of 256-bit random permutation ($= 2^{2040}$) is much higher than 6-round Kalyna-256/512 ($= 2^{440}$), a valid distinguisher is therefore constructed.

7.6 Key Recovery Attack on 9-Round Kalyna-256 /512

In this section, we use our Observation 6 to launch meet-in-the-middle attack on 9-round Kalyna-256/512 to recover the key. The distinguisher is placed in round 0 to round 5 (as shown in Fig. 7.11) and three rounds are added at the bottom of the 6-round distinguisher. The attack consists of the following three phases:

7.6.1 Precomputation Phase

In this phase, we build a lookup table T to store 2^{440} sequences to be used for comparison in the online phase. The construction of this table requires us to create two more hash tables (T_0 and T_1) in the intermediate steps. The entire procedure is as follows:

1. For each K_3

    - We guess $\Delta Z_1[4, 5, 14, 15, 16, 17, 26, 27] \,||\, \Delta X_4$[2, 3, 6, 7, 8, 9, 12, 13, 18, 19, 22, 23, 24, 25, 28, 29] to compute ΔX_2 and ΔY_3 respectively. We resolve (ΔX_2 - ΔY_3) using *Property 1b* to compute the corresponding $X_2||Y_3$. We then deduce K_2 from K_3 and compute the corresponding value of Z_1[4, 5, 14, 15, 16, 17, 26, 27]. Using the guessed value of $\Delta Z_1[4, 5, 14, 15, 16, 17, 26, 27]$ and the computed value of $Z_1[0-3, 12-15]$, we compute $\Delta Z_0[16-23]$. If $\Delta Z_0[16-22] = 0$ (which happens with a probability of 2^{-56}), we store the corresponding $X_1[16-23] \,||\, \Delta Z_1[4, 5, 14, 15, 16, 17, 26, 27] \,||\, X_2 \,||\, X_3 \,||\, W_3[7, 8, 19] \,||\, \Delta X_4$[2, 3, 6, 7, 8, 9, 12, 13, 18, 19, 22, 23, 24, 25, 28, 29] at index K_3 in table T_0. There are about 2^{136} entries for each index.

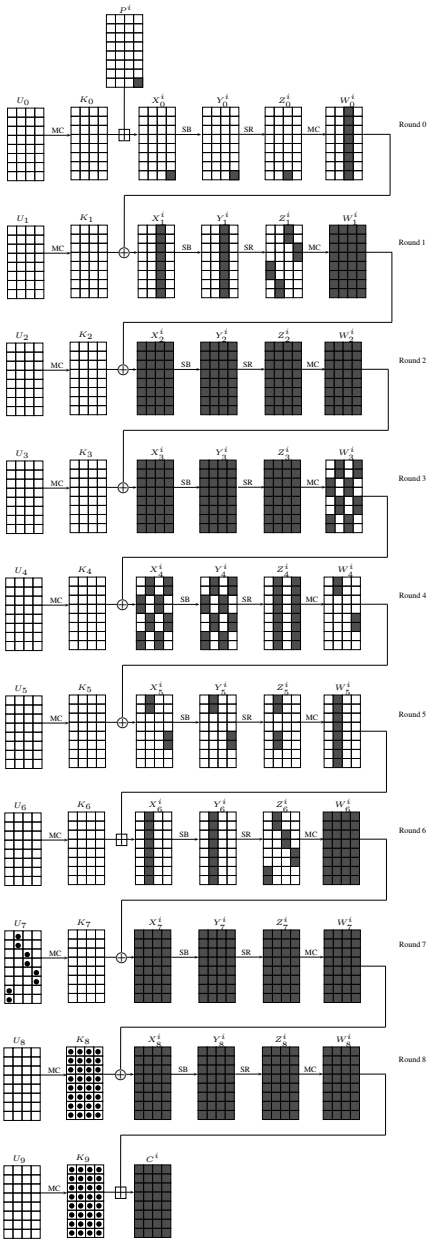


Figure 7.11: 9-round attack on Kalyna-256/512. The subkey bytes guessed are shown dotted.

2. For each guess of $\Delta Z_5[8, 9, 12, 13] \mid\mid Y_5[8, 9, 28, 29]$, compute $X_5[8, 9, 28, 29]$, $\Delta W_4[8, 9, 28, 29]$ and $\Delta Y_4[2, 3, 6, 7, 8, 9, 12, 13, 18, 19, 22, 23, 24, 25, 28, 29]$. Guess $Y_4[2, 3, 6, 7, 8, 9, 12, 13, 18, 19, 22, 23, 24, 25, 28, 29]$ to compute $X_4[2, 3, 6, 7, 8, 9, 12, 13, 18, 19, 22, 23, 24, 25, 28, 29]$ and $\Delta X_4[2, 3, 6, 7, 8, 9, 12, 13, 18, 19, 22, 23, 24, 25, 28, 29]$ in the backward direction and $W_4[8, 9, 28, 29]$ in the forward direction. From, $W_4[8, 9, 28, 29]$ and $X_5[8, 9, 28, 29]$ compute $K_5[8, 9, 28, 29]$. De-

duce $K_4[7, 8, 19]$ (where, $K_5[8] = K_4[19]$, $K_5[28] = K_4[7]$ and $K_5[29] = K_4[8]$). Using, $X_4[7, 8, 19]$ and $K_4[7, 8, 19]$, compute $W_3[7, 8, 19]$.

3. For each entry of $W_3[7, 8, 19] \mid\mid \Delta X_4$ [2, 3, 6, 7, 8, 9, 12, 13, 18, 19, 22, 23, 24, 25, 28, 29], we store $X_4[2, 3, 6, 7, 8, 9, 12, 13, 18, 19, 22, 23, 24, 25, 28, 29] \mid\mid X_5[8, 9, 28, 29]$ in a table T_1. There are 2^{40} entries per index.

4. For each of the 2^{256} index of K_3 in table T_0, we have 2^{136} entries of $W_3[7, 8, 19] \mid\mid \Delta X_4$ [2, 3, 6, 7, 8, 9, 12, 13, 18, 19, 22, 23, 24, 25, 28, 29] and corresponding to each of these we have 2^{40} entries of $X_4[2, 3, 6, 7, 8, 9, 12, 13, 18, 19, 22, 23, 24, 25, 28, 29] \mid\mid X_5[8, 9, 28, 29]$ in table T_1. So in all, after merging T_0 and T_1, we get $2^{256+136+40} = 2^{432}$ unique set of 65-byte parameters mentioned in *Observation 5*.

5. For each guess of $X_0[31]$, combine the above merged entries with $X_0[31]$ to complete the set of 66-parameters mentioned in *Observation 5*. Now, there are a total of $2^{432+8} = 2^{440}$ entries.

6. For each of these 2^{440} 66-byte parameters, we calculate the corresponding 93-byte parameters for all the elements of the δ-list and compute the ordered sequence $ov = \{ou^0 \oplus ou^0, ou^1 \oplus ou^0, \ldots, ou^{255} \oplus ou^0\}$. We store the ordered sequence along with the 93-byte parameters in the table T.

The time complexity to construct $T_0 = 2^{(32+8+16)\times 8} \times 2^{-2.17} = 2^{445.83}$. The time complexity to construct $T_1 = 2^{(4+4+16)\times 8} \times 2^{-2.17} = 2^{189.83}$. The time complexity to merge T_0 and T_1 along with each guess of $X_0[31] = 2^{256+136+40+8} = 2^{440}$. Finally, the time complexity to construct $T = 2^{440} \times 2^8 \times 2^{-0.58} = 2^{447.42}$. Hence, overall time complexity is $2^{445.83} + 2^{447.42} \approx 2^{447.83}$.

7.6.2 Online Phase

In this phase we extend the distinguisher in Section 7.10, by adding 3 more rounds at the bottom (as shown in Fig. 7.11). The steps of the online phase are as follows:

1. We encrypt 2^{209} structures of 2^8 plaintexts each where byte 31 takes all possible values and rest of the bytes are constants. We store the corresponding ciphertexts in the hash table.

2. For each of the 2^{224} (P_0, P_0') plaintext pairs, do the following:

   - We guess 2^{256} values of K_9 and deduce the corresponding values of K_8 from K_9. We decrypt each of the ciphertext pairs through 2 rounds, to get X_7 and ΔX_7. Then, we deduce the corresponding ΔW_6 and ΔZ_6.
   - We filter out the keys, which do not give zero difference at $\Delta Z_6[0-5, 10-17, 20-27, 30, 31]$. This creates a filtering of 2^{-192} and hence only 2^{64} key guesses are expected to remain.

- We pick one member of the pair, say P_0, create the δ-list by constructing the rest of the 255 plaintexts as $P_i = P_0 \oplus i$, where, $1 \leq i \leq 255$ and get their corresponding ciphertexts.
- For each of the remaining 2^{64} key guesses of K_8 and K_9, we guess $U_7[6, 7, 8, 9, 18, 19, 28, 29]$, compute the corresponding $Z_6[6, 7, 8, 9, 18, 19, 28, 29]$ and $Y_6[8 - 15]$ and then obtain the ordered sequence $\{ ou^0 \oplus ou^0, ou^1 \oplus ou^0, \ldots, ou^{255} \oplus ou^0\}$.
- We check whether this sequence exists in the precomputation table T or not. If not, then we discard the corresponding guesses.

Reason for counting ordered sequences instead of multisets. The probability for a wrong guess to pass the test is $2^{440} \times 2^{-2040} = 2^{-1600}$. Since we try only $2^{224+64} = 2^{288}$ ordered sequences, only the right subkey should verify the test.

If we had opted for mutliset attack on Kalyna-256/512, the total possible admissible multisets would have been 2^{432} (as the parameter $X_0[31]$ would not have been required). Therefore, the probability for a wrong guess to pass the test would have been $2^{432} \times 2^{-467.6} = 2^{-35.6}$ (similar to that described in Section 7.4). As mentioned above, since we try 2^{288} multisets, we would have got mutliple candidates for the right subkey and unable to recover the secret key.

7.6.3 Recovering the remaining Subkey bytes

The remaining subkeys recovery process is similar to that discussed in Section 7.4.3. When a match with an ordered sequence is found using a given plaintext-ciphertext pair, we choose one of the ciphertexts and perform the following steps:

1. We already know the corresponding K_8 and K_9 and $U_7[6, 7, 8, 9, 18, 19, 28, 29]$.
2. We guess the remaining 24 bytes of U_7, and deduce the corresponding 2^{192} values of K_7 and K_6.
3. For each 2^{192} guesses of (K_7, K_6), from X_7 we compute X_5. We discard the key guesses for which $X_5[8, 9, 28, 29]$ does not match with the values of $X_5[8, 9, 28, 29]$ obtained from the corresponding matched ordered sequence in the pre-computation table.
4. For the remaining $2^{192-32} = 2^{160}$ guesses of (K_9, K_8, K_7, K_6), we guess 2^{256} values of K_5. We deduce X_4 and discard the key guesses for which X_4 [2, 3, 6, 7, 8, 9, 12, 13, 18, 19, 22, 23, 24, 25, 28, 29] does not match with the values obtained corresponding to the correct ordered sequence from the precomputation table. From a total of $2^{160+256} = 2^{416}$ key guesses, 2^{288} key guesses are expected to remain.

5. We deduce K_4 from K_5 for the remaining key guesses and compute X_3. We compare this to the value obtained from the precomputation table corresponding to the correct ordered sequence and discard those that do not match. 2^{32} values of (K_9, K_8, K_7, K_6, K_5, K_4) are expected to remain.

6. One value of K_3 and K_2 corresponding to the matching sequence is already known from the pre-computation table. We deduce X_1 for the remaining 2^{32} values of (K_9, K_8, K_7, K_6, K_5, K_4, K_3, K_2).

7. We guess 2^{256} values of K_1, deduce K_0 and compute $X_0[23]$ and plaintext. We compare this to $X_0[23]$ obtained from the precomputation table and to the plaintext corresponding to ciphertext being decrypted respectively. We are left with only 2^{24} values of (K_9, K_8, K_7, K_6, K_5, K_4, K_3, K_2, K_1, K_0). We search these exhaustively to find the correct set of subkeys.

Complexities. The time complexity of the precomputation phase is $2^{447.83}$ Kalyna-128/256 encryptions. The time complexity of the online phase is dominated by step 2 (part 1) and is $2^{224} \times 2^{256} \times 2^{-2.17} = 2^{477.83}$. The time complexity of the subkey recovery phase is dominated by step 4 which is $2^{160} \times 2^{256} \times 2^{-3.17} = 2^{412.83}$. Clearly the time complexity of the whole attack is dominated by the time complexity of the online phase, i.e., $2^{477.83}$. It was shown in [61] that each 256-byte multiset requires 512-bits space. Hence, to store each entry in table T, we require 2048-bits to store the ordered sequence and $93 \times 8 = 744$-bits to store the 52-byte parameters, i.e., a total of 928-bits ($= 2^{11.45}$). Therefore, the memory complexity of this attack is $2^{440} \times 2^{11.45-8} = 2^{443.45}$ Kalyna 256-bit blocks. The data complexity of this attack is 2^{217} plaintexts.

7.7 Conclusions

In this chapter, we review the recent advancements in multiset attacks on AES and utilize them to launch key recovery attack on Kalyna-128/256 and Kalyna-256/512. We improve the previous best 7-round attack on both the variants to demonstrate the first 9-round attacks on the same. Our attacks on Kalyna-256/512 even improve upon the previous 7-round attack on it in terms of time and data complexities. We obtain these results by constructing new 6-round distinguishers on Kalyna and applying MITM attack on the rest of the rounds. Currently, this line of attack only works on Kalyna-b/2b variants and Kalyna variants in which block size and key size are equal appear to be safe. It would be an interesting problem to try applying multiset attacks on Kalyna-b/b. Presently, all five variants of Kalyna have been included in the Ukranian standard. However, our results as well as the previous 7-round attack show that compared to Kalyna-b/2b variants, Kalyna-b/b variants appear to be more robust.

Chapter 8

Conclusion

In this chapter, we summarize the cryptanalytic results presented in this thesis and give some possible directions for future research.

8.1 Summary

In this thesis, we focused on the security analysis of block ciphers and block cipher based hash functions. We studied two state-of-the-art cryptanalytic techniques namely – *Biclique Cryptanalysis* and *Multiset Attacks* and used them to provide the best attacks on three standardized block ciphers, i.e., AES, ARIA and Kalyna and Generalized Feistel Networks.

1. We studied biclique based key recovery attacks on AES and through a computer assisted search found improvements that lowered the attack costs compared to the original attack in [39] in Chapter 3. These attacks were applied to full round AES-128 (10-rounds), AES-192 (12-rounds) and AES-256 (14-rounds) with interesting observations and results. As part of the results, we proposed star-based bicliques which allowed us to launch attacks with the minimal data complexity in accordance with the unicity distance. Each attack required just 2-3 known plaintexts with success probability 1. This result can be used as a direct comparison with bruteforce attacks and shows that compared to brute force, biclique attack on AES will always have an advantage of atleast a factor of 2. We also found biclique attacks that are fastest when there is no restriction on data complexity. Through our automated results, we can safely say that as long as key recovery attacks are concerned, biclique attack in isolation does not pose any practical threat to AES security owing to its high time complexity. This can be assumed to hold true for other block ciphers as well whose designs are inspired from AES. However, it helps to better understand the security margin provided by the key used in these algorithms. Therefore, modern block ciphers have now started evaluating their security against this technique as a measure of their de-

sign assessment [6].

2. We next reviewed the application of biclique attacks in hash function settings. In Chapter 4, we utilized the biclique based key recovery attacks to find second-preimages on AES based hashing modes. In our attacks, we considered the initialization vector (IV) to be a public constant that cannot be changed by an attacker and showed that under this scenario, the biclique trails constructed for key recovery attack on AES-128 cannot be trivially used to launch second preimage attack on AES-128 based hash functions. We then constructed new biclique trails that satisfied the above restrictions and enabled an attacker to launch second preimage attacks on all 12 PGV hashing modes based on full round AES-128. In Chapter 5, we discussed a variant of biclique cryptanalysis termed as sliced biclique cryptanalysis which has been specially designed to find preimages and collisions in hash functions in known-key settings, i.e., when the key input to the block cipher is known to the attacker. We investigated the security of 4-branch, Type-2 based Generalized Feistel Networks (GFNs) and demonstrated the best 8-round collision attack on hash function based on this structure when the inner round function F was instantiated with double SP layers. Although sliced biclique cryptanalysis technique gives much higher advantage to an attacker in terms of time complexity as compared to regular bicliques, however its working demands special conditions to be imposed on the intermediate matching variable which is not always possible to find in some designs such as AES based hash functions. Moreover, the definition of sliced biclique attack will always lead to pseudo-collisions and pseudo-preimage attacks on Davies-Meyer based hash modes. Due to these challenges, we adopted a different approach to find preimage on AES based compression functions as discussed in Chapter 4.

3. We then again switched back to block cipher cryptanalysis and discussed multiset attacks in Chapters 6 and 7 which have been known to yield the most efficient key recovery attacks results (in terms of lowest time complexity) on AES. In Chapter 6, we analyzed the security of Korean Encryption Standard ARIA against this attack. We conducted multiset based key recovery attacks on 7 and 8-round ARIA-192 and ARIA-256 with improved time, memory and data complexities as compared to the previous best attack in [168]. Our attacks also demonstrated the first recovery of the secret master key unlike the previous attacks on ARIA which could only recover some intermediate round keys. In Chapter 7, we analyzed the security of recently announced Ukranian Encryption Standard Kalyna against multiset attack. We applied multiset attacks supplemented with further related advancements in this attack technique to recover the secret key from 9-round Kalyna-128/256 and Kalyna-256/512. This improved upon the previous best attack reported in [13] in terms of number of rounds attacked by 2. Although, the key schedule algorithm of Kalyna is stronger in comparison to AES as it does

not allow recovery of the master key from one subkey, still it allows recovery of odd-round keys from even-round keys and vice-versa. This was one of the crucial properties exploited by us to launch 9-round attacks on Kalyna variants. Comparatively, ARIA key schedule is much stronger as it does not have any such limitations. In fact, because of this property, we could not extend the number of rounds attacked in it. Hence, for any new block cipher design, adoption of key schedule algorithms similar to ARIA looks promising.

8.1.1 Future Work

In this section, we attempt to identify future research directions in which we can try applying the topics discussed in this thesis.

1. Presently, biclique technique for key recovery attacks suffer from very high time complexity as well as data complexity. Low data complexity attacks such as star-based bicliques and narrow bicliques look promising and needs to be investigated more. Further, lowering the time complexity of biclique attack to reasonable bounds looks a challenging direction to work on.

2. Biclique technique has been applied to block cipher and hash functions. However, their application on MAC functions has yet not been studied. An interesting case would be CBC-MAC based on AES.

3. Application of biclique technique to find preimages and collisions in double block length hash function such as MDC-2, MDC-4, Abreast DM, Tandem DM etc. is another area which has not yet been covered.

4. Generalized Feistel Network (GFN) based primitives have received lesser attention from biclique cryptanalysis as compared to Substitution - Permutation Network based primitives. Ciphers like CLEFIA, CAMELLIA and hash function like SHAvite-3 have yet not been tested against biclique cryptanalysis and can be one possible direction of research.

5. Another research direction would be to analyse other types of GFN structures, e.g., type-1, type-3 GFN etc. Their permutation properties are different from Type-2 GFN. Therefore, it would be interesting to study the relationship between the structure of GFN and its strength against biclique attack.

6. The current multiset attacks have been primarily focused on dedicated SP based block ciphers only. Their application to Feistel-SP functions have not been investigated yet. It would be interesting to study the propagation of multisets through Feistel structure and utilize them to recover the secret key if possible.

7. Recently in FSE'15, integral cryptanalysis was applied to break 6 rounds of AES-128 with secret S-box [169]. It would be interesting to investigate the application

of multiset attacks discussed in this thesis to AES and other block ciphers in secret key setting.

Bibliography

[1] http://www.slideshare.net/oliynykov/kalyna-english.

[2] SKIPJACK and KEA Algorithm Specifications Version 2.0. Technical report, National Institute of Standards and Technology (NIST), May 1998. Available from NIST:http://csrc.nist.gov/encryption/skipjack/skipjack.pdf.

[3] 3rd Generation Partnership Project. Specification of the 3GPP Confidentiality and Integrity Algorithms - Document 2: KASUMI Specification (Release 6). Technical Report 3GPP TS 35.202 V6.1.0 (2005-09), 2005.

[4] Farzaneh Abed, Christian Forler, Eik List, Stefan Lucks, and Jakob Wenzel. Biclique cryptanalysis of the PRESENT and LED lightweight ciphers. *IACR Cryptology ePrint Archive*, 2012:591, 2012. http://eprint.iacr.org/2012/591.

[5] Farzaneh Abed, Christian Forler, Eik List, Stefan Lucks, and Jakob Wenzel. A Framework for Automated Independent-Biclique Cryptanalysis. In Shiho Moriai, editor, *Fast Software Encryption - 20th International Workshop, FSE 2013, Singapore, March 11-13, 2013. Revised Selected Papers*, volume 8424 of *Lecture Notes in Computer Science*, pages 561–581. Springer, 2013.

[6] Farzaneh Abed, Eik List, and Stefan Lucks. On the Security of the Core of PRINCE Against Biclique and Differential Cryptanalysis. *IACRCryptology ePrint Archive*, 2012:712, 2012. https://eprint.iacr.org/2012/712.pdf.

[7] Charles M. Adams. Simple and Effective Key Scheduling for Symmetric Ciphers. *Workshop on Selected Areas of Cryptography, Queen's University, Kingston, Ontario, Canada, Proceedings*, pages 129–133, May 1994.

[8] Megha Agrawal, Donghoon Chang, Mohona Ghosh, and Somitra Kumar Sanadhya. Collision Attack on 4-Branch, Type-2 GFN Based Hash Functions Using Sliced Biclique Cryptanalysis Technique. In Dongdai Lin, Moti Yung, and Jianying Zhou, editors, *Information Security and Cryptology - 10th International Conference, Inscrypt 2014, Beijing, China, December 13-15, 2014, Revised Selected Papers*, volume 8957 of *Lecture Notes in Computer Science*, pages 343–360. Springer, 2014.

[9] Zahra Ahmadian, Mahmoud Salmasizadeh, and Mohammad Reza Aref. Biclique Cryptanalysis of the Full-Round KLEIN Block Cipher. *IACR Cryptology ePrint Archive*, 2013:97, 2013. `https://eprint.iacr.org/2013/097.pdf`.

[10] Akshima, Donghoon Chang, Mohona Ghosh, Aarushi Goel, and Somitra Kumar Sanadhya. Improved Meet-in-the-Middle Attacks on 7 and 8-Round ARIA-192 and ARIA-256. In Alex Biryukov and Vipul Goyal, editors, *Progress in Cryptology - INDOCRYPT 2015 - 16th International Conference on Cryptology in India, Bangalore, India, December 6-9, 2015, Proceedings*, volume 9462 of *Lecture Notes in Computer Science*, pages 198–217. Springer, 2015.

[11] Akshima, Donghoon Chang, Mohona Ghosh, Aarushi Goel, and Somitra Kumar Sanadhya. Single Key Recovery Attacks on 9-round Kalyna-128/256 and Kalyna-256/512. In Soonhak Kwon and Aaram Yun, editors, *Information Security and Cryptology - ICISC 2015 - 18th International Conference, Seoul, Korea, November 25-27, 2015, Revised Selected Papers*, volume 9558 of *Lecture Notes in Computer Science*. Springer, 2015. Available at:`https://eprint.iacr.org/2015/1227.pdf`.

[12] Ange Albertini, Jean-Philippe Aumasson, Maria Eichlseder, Florian Mendel, and Martin Schläffer. Malicious Hashing: Eve's Variant of SHA-1. In Antoine Joux and Amr M. Youssef, editors, *Selected Areas in Cryptography - SAC 2014 - 21st International Conference, Montreal, QC, Canada, August 14-15, 2014, Revised Selected Papers*, volume 8781 of *Lecture Notes in Computer Science*, pages 1–19. Springer, 2014.

[13] Riham AlTawy, Ahmed Abdelkhalek, and Amr M. Youssef. A Meet-in-the-Middle Attack on Reduced-Round Kalyna-b/2b. *IACR Cryptology ePrint Archive*, 2015:762, 2015. `http://eprint.iacr.org/2015/762`.

[14] Ross J. Anderson and Eli Biham. Two Practical and Provably Secure Block Ciphers: BEARS and LION. In Dieter Gollmann, editor, *FSE'96*, volume 1039 of *Lecture Notes in Computer Science*, pages 113–120. Springer, 1996.

[15] Kazumaro Aoki, Jian Guo, Krystian Matusiewicz, Yu Sasaki, and Lei Wang. Preimages for Step-Reduced SHA-2. In Mitsuru Matsui, editor, *Advances in Cryptology - ASIACRYPT 2009, 15th International Conference on the Theory and Application of Cryptology and Information Security, Tokyo, Japan, December 6-10, 2009. Proceedings*, volume 5912 of *Lecture Notes in Computer Science*, pages 578–597. Springer, 2009.

[16] Kazumaro Aoki, Tetsuya Ichikawa, Masayuki Kanda, Mitsuru Matsui, Shiho Moriai, Junko Nakajima, and Toshio Tokita. Camellia: A 128-Bit Block Cipher Suitable for Multiple Platforms - Design and Analysis. In Douglas R. Stinson and Stafford E. Tavares, editors, *Selected Areas in Cryptography, 7th Annual*

International Workshop, SAC 2000, Waterloo, Ontario, Canada, August 14-15, 2000, Proceedings, volume 2012 of *Lecture Notes in Computer Science*, pages 39–56. Springer, 2000.

[17] Kazumaro Aoki and Yu Sasaki. Preimage Attacks on One-Block MD4, 63-Step MD5 and More. In Roberto Maria Avanzi, Liam Keliher, and Francesco Sica, editors, *Selected Areas in Cryptography, 15th International Workshop, SAC 2008, Sackville, New Brunswick, Canada, August 14-15, Revised Selected Papers*, volume 5381 of *Lecture Notes in Computer Science*, pages 103–119. Springer, 2008.

[18] Kazumaro Aoki and Yu Sasaki. Meet-in-the-Middle Preimage Attacks Against Reduced SHA-0 and SHA-1. In Shai Halevi, editor, *Advances in Cryptology - CRYPTO 2009, 29th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 16-20, 2009. Proceedings*, volume 5677 of *Lecture Notes in Computer Science*, pages 70–89. Springer, 2009.

[19] Jean-Philippe Aumasson, Willi Meier, and Florian Mendel. Preimage Attacks on 3-Pass HAVAL and Step-Reduced MD5. In Roberto Maria Avanzi, Liam Keliher, and Francesco Sica, editors, *Selected Areas in Cryptography, 15th International Workshop, SAC 2008, Sackville, New Brunswick, Canada, August 14-15, Revised Selected Papers*, volume 5381 of *Lecture Notes in Computer Science*, pages 120–135. Springer, 2008.

[20] Behran Bahrak and Mohammad Reza Aref. Impossible Differential Attack on seven-round AES-128. *IET Information Security*, 2(2):28–32, June 2008.

[21] Paulo S. L. M. Barreto, Vincent Rijmen, Scopus Tecnologia S. A, and Cryptomathic Nv. The Whirlpool Hashing Function. In *First open NESSIE Workshop*, 2000. `http://www.larc.usp.br/~pbarreto/WhirlpoolPage.html`.

[22] Ray Beaulieu, Douglas Shors, Jason Smith, Stefan Treatman-Clark, Bryan Weeks, and Louis Wingers. The SIMON and SPECK lightweight block ciphers. In *Proceedings of the 52nd Annual Design Automation Conference, San Francisco, CA, USA, June 7-11, 2015*, pages 175:1–175:6. ACM, 2015.

[23] Eli Biham, Alex Biryukov, and Adi Shamir. Cryptanalysis of Skipjack Reduced to 31 Rounds Using Impossible Differentials. *J. Cryptology*, 18(4):291–311, 2005.

[24] Eli Biham, Orr Dunkelman, and Nathan Keller. The Rectangle Attack - Rectangling the Serpent. In Birgit Pfitzmann, editor, *Advances in Cryptology - EUROCRYPT 2001, International Conference on the Theory and Application of Cryptographic Techniques, Innsbruck, Austria, May 6-10, 2001, Proceeding*, volume 2045 of *Lecture Notes in Computer Science*, pages 340–357. Springer, 2001.

[25] Eli Biham, Orr Dunkelman, and Nathan Keller. Related-Key Boomerang and Rectangle Attacks. In Ronald Cramer, editor, *Advances in Cryptology - EUROCRYPT 2005, 24th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Aarhus, Denmark, May 22-26, 2005, Proceedings*, volume 3494 of *Lecture Notes in Computer Science*, pages 507–525. Springer, 2005.

[26] Eli Biham, Orr Dunkelman, and Nathan Keller. Related-Key Impossible Differential Attacks on 8-Round AES-192. In David Pointcheval, editor, *Topics in Cryptology - CT-RSA 2006, The Cryptographers' Track at the RSA Conference 2006, San Jose, CA, USA, February 13-17, 2006, Proceedings*, volume 3860 of *Lecture Notes in Computer Science*, pages 21–33. Springer, 2006.

[27] Eli Biham and Orr Dunkeman. The SHAvite-3 Hash Function. Submission to NIST SHA-3 competition. `www.cs.technion.ac.il/~orrd/SHAvite-3/`.

[28] Eli Biham and Adi Shamir. Differential Cryptanalysis of DES-like Cryptosystems. In Alfred Menezes and Scott A. Vanstone, editors, *Advances in Cryptology - CRYPTO '90, 10th Annual International Cryptology Conference, Santa Barbara, California, USA, August 11-15, 1990, Proceedings*, volume 537 of *Lecture Notes in Computer Science*, pages 2–21. Springer, 1990.

[29] Eli Biham and Adi Shamir. *Differential Cryptanalysis of the Data Encryption Standard*. Springer, 1993.

[30] Alex Biryukov, Christophe De Cannière, Joseph Lano, Siddika Berna Ors, and Bart Preneel. Security and Performance Analysis of ARIA, version 1.2. Technical report, Katholieke Universiteit Leuven, Belgium, 2004. `http://www.cosic.esat.kuleuven.be/publications/article-500.pdf`.

[31] Alex Biryukov and Dmitry Khovratovich. Related-Key Cryptanalysis of the Full AES-192 and AES-256. In *Asiacrypt 2009*, volume 5912 of *Lecture Notes in Computer Science*, pages 1–18. Springer, 2009.

[32] Alex Biryukov and Adi Shamir. Structural Cryptanalysis of SASAS. In Birgit Pfitzmann, editor, *Advances in Cryptology - EUROCRYPT 2001, International Conference on the Theory and Application of Cryptographic Techniques, Innsbruck, Austria, May 6-10, 2001, Proceeding*, volume 2045 of *Lecture Notes in Computer Science*, pages 394–405. Springer, 2001.

[33] John Black. The Ideal-Cipher Model, Revisited: An Uninstantiable Blockcipher-Based Hash Function. In Matthew J. B. Robshaw, editor, *Fast Software Encryption, 13th International Workshop, FSE 2006, Graz, Austria, March 15-17, 2006, Revised Selected Papers*, volume 4047 of *Lecture Notes in Computer Science*, pages 328–340. Springer, 2006.

[34] John Black, Phillip Rogaway, and Thomas Shrimpton. Black-Box Analysis of the Block-Cipher-Based Hash-Function Constructions from PGV. In Moti Yung, editor, *Advances in Cryptology - CRYPTO 2002, 22nd Annual International Cryptology Conference, Santa Barbara, California, USA, August 18-22, 2002, Proceedings*, volume 2442 of *Lecture Notes in Computer Science*, pages 320–335. Springer, 2002.

[35] Andrey Bogdanov. *Analysis and design of block cipher constructions*. PhD thesis, Ruhr University Bochum, 2010.

[36] Andrey Bogdanov. On the differential and linear efficiency of balanced Feistel networks. *Inf. Process. Lett.*, 110(20):861–866, 2010.

[37] Andrey Bogdanov, Donghoon Chang, Mohona Ghosh, and Somitra Kumar Sanadhya. Bicliques with Minimal Data and Time Complexity for AES. In Jooyoung Lee and Jongsung Kim, editors, *Information Security and Cryptology - ICISC 2014 - 17th International Conference, Seoul, Korea, December 3-5, 2014, Revised Selected Papers*, volume 8949 of *Lecture Notes in Computer Science*, pages 160–174. Springer, 2014.

[38] Andrey Bogdanov, Elif Bilge Kavun, Christof Paar, Christian Rechberger, and Tolga Yalcin. Better than Brute-Force Optimized Hardware Architecture for Effcient Biclique Attacks on AES-128. In *SHARCS'12 - Special-Purpose Hardware for Attacking Cryptographic Systems, March 2012, Washington D.C., USA*, 2012.

[39] Andrey Bogdanov, Dmitry Khovratovich, and Christian Rechberger. Biclique Cryptanalysis of the Full AES. In Dong Hoon Lee and Xiaoyun Wang, editors, *Advances in Cryptology - ASIACRYPT 2011 - 17th International Conference on the Theory and Application of Cryptology and Information Security, Seoul, South Korea, December 4-8, 2011. Proceedings*, volume 7073 of *Lecture Notes in Computer Science*, pages 344–371. Springer, 2011.

[40] Andrey Bogdanov, Lars R. Knudsen, Gregor Leander, Christof Paar, Axel Poschmann, Matthew J. B. Robshaw, Yannick Seurin, and C. Vikkelsoe. PRESENT: An Ultra-Lightweight Block Cipher. In Pascal Paillier and Ingrid Verbauwhede, editors, *Cryptographic Hardware and Embedded Systems - CHES 2007, 9th International Workshop, Vienna, Austria, September 10-13, 2007, Proceedings*, volume 4727 of *Lecture Notes in Computer Science*, pages 450–466. Springer, 2007.

[41] Andrey Bogdanov and Christian Rechberger. A 3-Subset Meet-in-the-Middle Attack: Cryptanalysis of the Lightweight Block Cipher KTANTAN. In *Selected Areas in Cryptography - 17th International Workshop, SAC 2010, Waterloo, Ontario, Canada, August 12-13, 2010, Revised Selected Papers*, pages 229–240, 2010.

[42] Andrey Bogdanov and Kyoji Shibutani. Generalized Feistel Networks Revisited. *Des. Codes Cryptography*, 66(1-3):75–97, 2013.

[43] Charles Bouillaguet, Patrick Derbez, and Pierre-Alain Fouque. Automatic Search of Attacks on Round-Reduced AES and Applications. In Phillip Rogaway, editor, *Advances in Cryptology - CRYPTO 2011 - 31st Annual Cryptology Conference, Santa Barbara, CA, USA, August 14-18, 2011. Proceedings*, volume 6841 of *Lecture Notes in Computer Science*, pages 169–187. Springer, 2011.

[44] B.O. Brachtl, D. Coppersmith, M.M. Hyden, S.M. Matyas, C.H.W. Meyer, J. Oseas, S. Pilpel, and M. Schilling. Data Authentication using Modification Detection Codes based on a Public One Way Encryption Function, 1990. US Patent 4,908,861, `http://www.google.co.in/patents/US4908861`.

[45] Carolynn Burwick, Don Coppersmith, Edward D'Avignon, Rosario Gennaro, Shai Halevi, Charanjit Jutla, Stephen M. Matyas, Luke O'Connor, Mohammad Peyravian, David Safford, and Nevenko Zunic. The MARS Encryption Algorithm, 1999. PDF available at:`http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.35.5887&rep=rep1&type=pdf`.

[46] Christophe De Cannière. *Analysis and Design of Symmetric Encryption Algorithms*. PhD thesis, Katholieke Universiteit Leuven, Belgium, May 2007.

[47] Christophe De Cannière, Orr Dunkelman, and Miroslav Knezevic. KATAN and KTANTAN - A Family of Small and Efficient Hardware-Oriented Block Ciphers. In Christophe Clavier and Kris Gaj, editors, *Cryptographic Hardware and Embedded Systems - CHES 2009, 11th International Workshop, Lausanne, Switzerland, September 6-9, 2009, Proceedings*, volume 5747 of *Lecture Notes in Computer Science*, pages 272–288. Springer, 2009.

[48] Anne Canteaut, María Naya-Plasencia, and Bastien Vayssière. Sieve-in-the-Middle: Improved MITM Attacks. In Ran Canetti and Juan A. Garay, editors, *Advances in Cryptology - CRYPTO 2013 - 33rd Annual Cryptology Conference, Santa Barbara, CA, USA, August 18-22, 2013. Proceedings, Part I*, volume 8042 of *Lecture Notes in Computer Science*, pages 222–240. Springer, 2013.

[49] Donghoon Chang, Mohona Ghosh, and Somitra Kumar Sanadhya. Biclique Cryptanalysis of full round AES-128 based hashing modes. In Dongdai Lin, Moti Yung, and Xiaofeng Wang, editors, *Information Security and Cryptology - 11th International Conference, Inscrypt 2015, Beijing, China, November 1-3, 2015, Revised Selected Papers*, volume To Be Announced of *Lecture Notes in Computer Science*. Springer, 2015.

[50] Donghoon Chang, Abhishek Kumar, and Somitra Kumar Sanadhya. Security Analysis of GFN: 8-Round Distinguisher for 4-Branch Type-2 GFN. In Goutam

Paul and Serge Vaudenay, editors, *INDOCRYPT'13*, volume 8250 of *Lecture Notes in Computer Science*, pages 136–148. Springer, 2013.

[51] David Chaum and Jan-Hendrik Evertse. Cryptanalysis of DES with a Reduced Number of Rounds. In Hugh C. Williams, editor, *Advances in Cryptology CRYPTO 85 Proceedings*, volume 218 of *Lecture Notes in Computer Science*, pages 192–211. Springer Berlin Heidelberg, 1986.

[52] Shao-zhen Chen and Tian-min Xu. Biclique key recovery for ARIA-256. *IET Information Security*, 8(5):259–264, 2014.

[53] Mustafa Çoban, Ferhat Karakoç, and Özkan Boztas. Biclique Cryptanalysis of TWINE. In Josef Pieprzyk, Ahmad-Reza Sadeghi, and Mark Manulis, editors, *Cryptology and Network Security, 11th International Conference, CANS 2012, Darmstadt, Germany, December 12-14, 2012. Proceedings*, volume 7712, pages 43–55. Springer, 2012.

[54] Jean-Sébastien Coron, Jacques Patarin, and Yannick Seurin. The Random Oracle Model and the Ideal Cipher Model Are Equivalent. In David Wagner, editor, *Advances in Cryptology - CRYPTO 2008, 28th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 17-21, 2008. Proceedings*, volume 5157 of *Lecture Notes in Computer Science*, pages 1–20. Springer, 2008.

[55] Joan Daemen, Lars R. Knudsen, and Vincent Rijmen. The Block Cipher SQUARE. In Eli Biham, editor, *Fast Software Encryption, 4th International Workshop, FSE '97, Haifa, Israel, January 20-22, 1997, Proceedings*, volume 1267 of *Lecture Notes in Computer Science*, pages 149–165. Springer, 1997.

[56] Joan Daemen and Vincent Rijmen. The Wide Trail Design Strategy. In Bahram Honary, editor, *Cryptography and Coding, 8th IMA International Conference, Cirencester, UK, December 17-19, 2001, Proceedings*, volume 2260 of *Lecture Notes in Computer Science*, pages 222–238. Springer, 2001.

[57] Joan Daemen and Vincent Rijmen. *The Design of Rijndael: AES - The Advanced Encryption Standard*. Springer, 2002.

[58] Joan Daemen and Vincent Rijmen. Understanding Two-Round Differentials in AES. In Roberto De Prisco and Moti Yung, editors, *Security and Cryptography for Networks, 5th International Conference, SCN 2006, Maiori, Italy, September 6-8, 2006, Proceedings*, volume 4116 of *Lecture Notes in Computer Science*, pages 78–94. Springer, 2006.

[59] Ivan Damgård. A Design Principle for Hash Functions. In Gilles Brassard, editor, *Advances in Cryptology - CRYPTO '89, 9th Annual International Cryptology Conference, Santa Barbara, California, USA, August 20-24, 1989, Proceedings*, volume 435 of *Lecture Notes in Computer Science*, pages 416–427. Springer, 1989.

[60] Hüseyin Demirci and Ali Aydin Selçuk. A Meet-in-the-Middle Attack on 8-Round AES. In Kaisa Nyberg, editor, *Fast Software Encryption, 15th International Workshop, FSE 2008, Lausanne, Switzerland, February 10-13, 2008, Revised Selected Papers*, volume 5086 of *Lecture Notes in Computer Science*, pages 116–126. Springer, 2008.

[61] Patrick Derbez, Pierre-Alain Fouque, and Jérémy Jean. Improved Key Recovery Attacks on Reduced-Round AES in the Single-Key Setting. In Thomas Johansson and Phong Q. Nguyen, editors, *Advances in Cryptology - EUROCRYPT 2013, 32nd Annual International Conference on the Theory and Applications of Cryptographic Techniques, Athens, Greece, May 26-30, 2013. Proceedings*, volume 7881 of *Lecture Notes in Computer Science*, pages 371–387. Springer, 2013.

[62] DES. Data Encryption Standard. In *FIPS PUB 46, Federal Information Processing Standards Publication*, pages 2–46, 1977.

[63] Anand Desai. The Security of All-or-Nothing Encryption: Protecting against Exhaustive Key Search. In Mihir Bellare, editor, *Advances in Cryptology - CRYPTO 2000, 20th Annual International Cryptology Conference, Santa Barbara, California, USA, August 20-24, 2000, Proceedings*, volume 1880 of *Lecture Notes in Computer Science*, pages 359–375. Springer, 2000.

[64] Carl D'Halluin, Gert Bijnens, Vincent Rijmen, and Bart Preneel. Attack on Six Rounds of Crypton. In Lars R. Knudsen, editor, *Fast Software Encryption, 6th International Workshop, FSE '99, Rome, Italy, March 24-26, 1999, Proceedings*, volume 1636 of *Lecture Notes in Computer Science*, pages 46–59. Springer, 1999.

[65] Whitfield Diffie and Martin E. Hellman. New directions in cryptography. *IEEE Transactions on Information Theory*, 22(6):644–654, 1976.

[66] Whitfield Diffie and Martin E. Hellman. Special Feature Exhaustive Cryptanalysis of the NBS Data Encryption Standard. *Computer*, 10(6):74–84, June 1977.

[67] Yevgeniy Dodis and Prashant Puniya. On the Relation Between the Ideal Cipher and the Random Oracle Models. In Shai Halevi and Tal Rabin, editors, *Theory of Cryptography, Third Theory of Cryptography Conference, TCC 2006, New York, NY, USA, March 4-7, 2006, Proceedings*, volume 3876 of *Lecture Notes in Computer Science*, pages 184–206. Springer, 2006.

[68] Le Dong, Wenling Wu, Shuang Wu, and Jian Zou. Known-key distinguishers on type-1 Feistel scheme and near-collision attacks on its hashing modes. *Frontiers of Computer Science*, 8(3):513–525, 2014.

[69] Chenghang Du and Jiazhe Chen. Impossible Differential Cryptanalysis of ARIA Reduced to 7 Rounds. In Swee-Huay Heng, Rebecca N. Wright, and Bok-Min Goi, editors, *Cryptology and Network Security - 9th International Conference, CANS*

2010, Kuala Lumpur, Malaysia, December 12-14, 2010. Proceedings, volume 6467 of *Lecture Notes in Computer Science*, pages 20–30. Springer, 2010.

[70] Orr Dunkelman and Nathan Keller. The effects of the omission of last round's mixcolumns on AES. *Inf. Process. Lett.*, 110(8-9):304–308, 2010.

[71] Orr Dunkelman, Nathan Keller, and Adi Shamir. Improved Single-Key Attacks on 8-Round AES-192 and AES-256. *J. Cryptology*, 28(3):397–422, 2015.

[72] Orr Dunkelman, Gautham Sekar, and Bart Preneel. Improved Meet-in-the-Middle Attacks on Reduced-Round DES. In K. Srinathan, C. Pandu Rangan, and Moti Yung, editors, *Progress in Cryptology - INDOCRYPT 2007, 8th International Conference on Cryptology in India, Chennai, India, December 9-13, 2007, Proceedings*, volume 4859 of *Lecture Notes in Computer Science*, pages 86–100. Springer, 2007.

[73] D. Eastlake and T. Hansen. US Secure Hash Algorithm (SHA and HMAC-SHA). RFC 4634, July 2006. `https://tools.ietf.org/html/rfc4634`.

[74] D. Eastlake and P. Jones. US Secure Hash Algorithm 1 (SHA1). RFC 3174, September 2001. `https://tools.ietf.org/html/rfc3174`.

[75] Niels Ferguson, John Kelsey, Stefan Lucks, Bruce Schneier, Michael Stay, David Wagner, and Doug Whiting. Improved Cryptanalysis of Rijndael. In Bruce Schneier, editor, *Fast Software Encryption, 7th International Workshop, FSE 2000, New York, NY, USA, April 10-12, 2000, Proceedings*, volume 1978 of *Lecture Notes in Computer Science*, pages 213–230. Springer, 2000.

[76] Ewan Fleischmann, Christian Forler, Michael Gorski, and Stefan Lucks. New Boomerang Attacks on ARIA. In Guang Gong and Kishan Chand Gupta, editors, *Progress in Cryptology - INDOCRYPT 2010 - 11th International Conference on Cryptology in India, Hyderabad, India, December 12-15, 2010. Proceedings*, volume 6498 of *Lecture Notes in Computer Science*, pages 163–175. Springer, 2010.

[77] Korean Agency for Technology and Standards. 128 bit block encryption algorithm ARIA - Part 1: General (in Korean). KS X 1213-1:2009, December 2009.

[78] Praveen Gauravaram, Lars R. Knudsen, Krystian Matusiewicz, Florian Mendel, Christian Rechberger, Martin Schläffer, and Søren S. Thomsen. Grøstl - a SHA-3 candidate. Submission to NIST (Round 3), 2011. `http://www.groestl.info/Groestl.pdf`.

[79] Henri Gilbert and Marine Minier. A Collision Attack on 7 Rounds of Rijndael. In *AES Candidate Conference*, pages 230–241, 2000.

[80] Jian Guo, San Ling, Christian Rechberger, and Huaxiong Wang. Advanced Meet-in-the-Middle Preimage Attacks: First Results on Full Tiger, and Improved Results on MD4 and SHA-2. In Masayuki Abe, editor, *Advances in Cryptology - ASIACRYPT 2010 - 16th International Conference on the Theory and Application of Cryptology and Information Security, Singapore, December 5-9, 2010. Proceedings*, volume 6477 of *Lecture Notes in Computer Science*, pages 56–75. Springer, 2010.

[81] Jian Guo, Thomas Peyrin, Axel Poschmann, and Matthew J. B. Robshaw. The LED Block Cipher. In Bart Preneel and Tsuyoshi Takagi, editors, *Cryptographic Hardware and Embedded Systems - CHES 2011 - 13th International Workshop, Nara, Japan, September 28 - October 1, 2011. Proceedings*, volume 6917 of *Lecture Notes in Computer Science*, pages 326–341. Springer, 2011.

[82] Kishan Chand Gupta and Indranil Ghosh Ray. On Constructions of MDS Matrices from Companion Matrices for Lightweight Cryptography. In Alfredo Cuzzocrea, Christian Kittl, Dimitris E. Simos, Edgar R. Weippl, and Lida Xu, editors, *Security Engineering and Intelligence Informatics - CD-ARES 2013 Workshops: MoCrySEn and SeCIHD, Regensburg, Germany, September 2-6, 2013. Proceedings*, volume 8128 of *Lecture Notes in Computer Science*, pages 29–43. Springer, 2013.

[83] Martin E. Hellman. A cryptanalytic time-memory trade-off. *IEEE Transactions on Information Theory*, 26(4):401–406, 1980.

[84] Shoichi Hirose. Some Plausible Constructions of Double-Block-Length Hash Functions. In Matthew J. B. Robshaw, editor, *Fast Software Encryption, 13th International Workshop, FSE 2006, Graz, Austria, March 15-17, 2006, Revised Selected Papers*, volume 4047 of *Lecture Notes in Computer Science*, pages 210–225. Springer, 2006.

[85] Viet Tung Hoang and Phillip Rogaway. On generalized feistel networks. In Tal Rabin, editor, *Advances in Cryptology - CRYPTO 2010, 30th Annual Cryptology Conference, Santa Barbara, CA, USA, August 15-19, 2010. Proceedings*, volume 6223 of *Lecture Notes in Computer Science*, pages 613–630. Springer, 2010.

[86] Deukjo Hong, Bonwook Koo, and Daesung Kwon. Biclique Attack on the Full HIGHT. In Howon Kim, editor, *Information Security and Cryptology - ICISC 2011 - 14th International Conference, Seoul, Korea, November 30 - December 2, 2011. Revised Selected Papers*, volume 7259 of *Lecture Notes in Computer Science*, pages 365–374. Springer, 2011.

[87] Deukjo Hong, Jaechul Sung, Seokhie Hong, Jongin Lim, Sangjin Lee, Bonseok Koo, Changhoon Lee, Donghoon Chang, Jaesang Lee, Kitae Jeong, Hyun Kim, Jongsung Kim, and Seongtaek Chee. HIGHT: A New Block Cipher Suitable for

Low-Resource Device. In Louis Goubin and Mitsuru Matsui, editors, *CHES'06*, volume 4249 of *Lecture Notes in Computer Science*, pages 46–59. Springer, 2006.

[88] Horst Feistel and William A. Notz and J. Lynn Smith. Some Cryptographic Techniques for Machine-to-Machine Data Communications. *Proc. IEEE*, 63(11):1545–1554, November 1975.

[89] Takanori Isobe. A Single-Key Attack on the Full GOST Block Cipher. In Antoine Joux, editor, *Fast Software Encryption - 18th International Workshop, FSE 2011, Lyngby, Denmark, February 13-16, 2011, Revised Selected Papers*, volume 6733 of *Lecture Notes in Computer Science*, pages 290–305. Springer, 2011.

[90] Takanori Isobe and Kyoji Shibutani. Security Analysis of the Lightweight Block Ciphers XTEA, LED and Piccolo. In Willy Susilo, Yi Mu, and Jennifer Seberry, editors, *Information Security and Privacy - 17th Australasian Conference, ACISP 2012, Wollongong, NSW, Australia, July 9-11, 2012. Proceedings*, volume 7372 of *Lecture Notes in Computer Science*, pages 71–86. Springer, 2012.

[91] Éliane Jaulmes, Antoine Joux, and Frédéric Valette. On the Security of Randomized CBC-MAC Beyond the Birthday Paradox Limit: A New Construction. In Joan Daemen and Vincent Rijmen, editors, *Fast Software Encryption, 9th International Workshop, FSE 2002, Leuven, Belgium, February 4-6, 2002, Revised Papers*, volume 2365 of *Lecture Notes in Computer Science*, pages 237–251. Springer, 2002.

[92] Kitae Jeong, Hyungchul Kang, Changhoon Lee, Jaechul Sung, and Seokhie Hong. Biclique Cryptanalysis of Lightweight Block Ciphers PRESENT, Piccolo and LED. *IACR Cryptology ePrint Archive*, 2012:621, 2012. `http://eprint.iacr.org/2012/621`.

[93] Jorge Nakahara Jr., Paulo S. L. M. Barreto, Bart Preneel, Joos Vandewalle, and H. Y. Kim. SQUARE attacks on reduced-round PES and IDEA block ciphers. *IACR Cryptology ePrint Archive*, 2001:68, 2001. `http://eprint.iacr.org/2001/068`.

[94] Pascal Junod. *Statistical Cryptanalysis of Block Ciphers*. PhD thesis, Ecole Polytechnique Federale de Lausanne, Switzerland, 2004.

[95] David Kahn. *The Codebreakers: The Comprehensive History of Secret Communication from Ancient Times to the Internet*. Scribner, New York, December 1996.

[96] Hyungchul Kang, Deukjo Hong, Dukjae Moon, Daesung Kwon, Jaechul Sung, and Seokhie Hong. Known-Key Attacks on Generalized Feistel Schemes with SP Round Function. *IEICE Transactions*, 95-A(9):1550–1560, 2012.

[97] Jonathan Katz and Yehuda Lindell. *Introduction to Modern Cryptography*. Chapman and Hall/CRC Press, 2007.

[98] John Kelsey, Tadayoshi Kohno, and Bruce Schneier. Amplified Boomerang Attacks Against Reduced-Round MARS and Serpent. In Bruce Schneier, editor, *Fast Software Encryption, 7th International Workshop, FSE 2000, New York, NY, USA, April 10-12, 2000, Proceedings*, volume 1978 of *Lecture Notes in Computer Science*, pages 75–93. Springer, 2000.

[99] Dmitry Khovratovich. Bicliques for Permutations: Collision and Preimage Attacks in Stronger Settings. In Xiaoyun Wang and Kazue Sako, editors, *Advances in Cryptology - ASIACRYPT 2012 - 18th International Conference on the Theory and Application of Cryptology and Information Security, Beijing, China, December 2-6, 2012. Proceedings*, volume 7658 of *Lecture Notes in Computer Science*, pages 544–561. Springer, 2012.

[100] Dmitry Khovratovich, Gaëtan Leurent, and Christian Rechberger. Narrow-Bicliques: Cryptanalysis of Full IDEA. In David Pointcheval and Thomas Johansson, editors, *Advances in Cryptology - EUROCRYPT 2012 - 31st Annual International Conference on the Theory and Applications of Cryptographic Techniques, Cambridge, UK, April 15-19, 2012. Proceedings*, volume 7237 of *Lecture Notes in Computer Science*, pages 392–410. Springer, 2012.

[101] Dmitry Khovratovich, María Naya-Plasencia, Andrea Röck, and Martin Schläffer. Cryptanalysis of *Luffa* v2 Components. In Alex Biryukov, Guang Gong, and Douglas R. Stinson, editors, *Selected Areas in Cryptography - 17th International Workshop, SAC 2010, Waterloo, Ontario, Canada, August 12-13, 2010, Revised Selected Papers*, volume 6544 of *Lecture Notes in Computer Science*, pages 388–409. Springer, 2010.

[102] Dmitry Khovratovich, Ivica Nikolic, and Christian Rechberger. Rotational Rebound Attacks on Reduced Skein. *J. Cryptology*, 27(3):452–479, 2014.

[103] Dmitry Khovratovich, Christian Rechberger, and Alexandra Savelieva. Bicliques for Preimages: Attacks on Skein-512 and the SHA-2 Family. In Anne Canteaut, editor, *Fast Software Encryption - 19th International Workshop, FSE 2012, Washington, DC, USA, March 19-21, 2012. Revised Selected Papers*, volume 7549 of *Lecture Notes in Computer Science*, pages 244–263. Springer, 2012.

[104] Woo-Hwan Kim, Jungkeun Lee, Je-Hong Park, and Daesung Kwon. Addition of the ARIA Cipher Suites to Transport Layer Security (TLS). RFC 6209, April 2011. `https://tools.ietf.org/html/rfc6209`.

[105] Lars R. Knudsen. Cryptanalysis of LOKI91. In Jennifer Seberry and Yuliang Zheng, editors, *Advances in Cryptology - AUSCRYPT '92, Workshop on the*

Theory and Application of Cryptographic Techniques, Gold Coast, Queensland, Australia, December 13-16, 1992, Proceedings, volume 718 of *Lecture Notes in Computer Science*, pages 196–208. Springer, 1992.

[106] Lars R. Knudsen. *Block Ciphers - Analysis, Design and Applications*. PhD thesis, Aarhus University, Denmark, 1994.

[107] Lars R. Knudsen. Truncated and Higher Order Differentials. In Bart Preneel, editor, *Fast Software Encryption: Second International Workshop. Leuven, Belgium, 14-16 December 1994, Proceedings*, volume 1008 of *Lecture Notes in Computer Science*, pages 196–211. Springer, 1994.

[108] Lars R. Knudsen, Gregor Leander, Axel Poschmann, and Matthew J. B. Robshaw. PRINTcipher: A Block Cipher for IC-Printing. In Stefan Mangard and François-Xavier Standaert, editors, *Cryptographic Hardware and Embedded Systems, CHES 2010, 12th International Workshop, Santa Barbara, CA, USA, August 17-20, 2010. Proceedings*, volume 6225 of *Lecture Notes in Computer Science*, pages 16–32. Springer, 2010.

[109] Lars R. Knudsen and Vincent Rijmen. Known-Key Distinguishers for Some Block Ciphers. In Kaoru Kurosawa, editor, *Advances in Cryptology - ASIACRYPT 2007, 13th International Conference on the Theory and Application of Cryptology and Information Security, Kuching, Malaysia, December 2-6, 2007, Proceedings*, volume 4833 of *Lecture Notes in Computer Science*, pages 315–324. Springer, 2007.

[110] Lars R. Knudsen and Matthew Robshaw. *The Block Cipher Companion*. Information Security and Cryptography. Springer, 2011.

[111] Lars R. Knudsen and David Wagner. Integral Cryptanalysis. In Joan Daemen and Vincent Rijmen, editors, *Fast Software Encryption, 9th International Workshop, FSE 2002, Leuven, Belgium, February 4-6, 2002, Revised Papers*, volume 2365 of *Lecture Notes in Computer Science*, pages 112–127. Springer, 2002.

[112] Neal Koblitz. Elliptic Curve Cryptosystems. *Mathematics of Computation*, 48(177):203–209, January 1987.

[113] Daesung Kwon, Jaesung Kim, Jungkeun Lee, Jooyoung Lee, and Choonsoo Kim. A Description of the ARIA Encryption Algorithm. RFC 5794, March 2010. `https://tools.ietf.org/html/rfc5794`.

[114] Daesung Kwon, Jaesung Kim, Sangwoo Park, Soo Hak Sung, Yaekwon Sohn, Jung Hwan Song, Yongjin Yeom, E-Joong Yoon, Sangjin Lee, Jaewon Lee, Seongtaek Chee andrebound Daewan Han, and Jin Hong. New Block Cipher: ARIA. In Jong In Lim and Dong Hoon Lee, editors, *Information Security and Cryptology - ICISC 2003, 6th International Conference, Seoul, Korea, November 27-28,*

2003, Revised Papers, volume 2971 of *Lecture Notes in Computer Science*, pages 432–445. Springer, 2003.

[115] RSA Laboratories. Additional PKCS #11 Mechanisms. PKCS #11 v2.20 Amendment 3 Revision 1, January 2007.

[116] Xuejia Lai and James L. Massey. A Proposal for a New Block Encryption Standard. In Ivan Damgård, editor, *Advances in Cryptology - EUROCRYPT '90, Workshop on the Theory and Application of of Cryptographic Techniques, Aarhus, Denmark, May 21-24, 1990, Proceedings*, volume 473 of *Lecture Notes in Computer Science*, pages 389–404. Springer, 1990.

[117] Xuejia Lai and James L. Massey. Hash Function Based on Block Ciphers. In Rainer A. Rueppel, editor, *Advances in Cryptology - EUROCRYPT '92, Workshop on the Theory and Application of of Cryptographic Techniques, Balatonfüred, Hungary, May 24-28, 1992, Proceedings*, volume 658 of *Lecture Notes in Computer Science*, pages 55–70. Springer, 1992.

[118] Mario Lamberger, Florian Mendel, Christian Rechberger, Vincent Rijmen, and Martin Schläffer. The Rebound Attack and Subspace Distinguishers: Application to Whirlpool. *IACR Cryptology ePrint Archive*, 2010:198, 2010.

[119] Susan K. Langford and Martin E. Hellman. Differential-Linear Cryptanalysis. In Yvo Desmedt, editor, *Advances in Cryptology - CRYPTO '94, 14th Annual International Cryptology Conference, Santa Barbara, California, USA, August 21-25, 1994, Proceedings*, volume 839 of *Lecture Notes in Computer Science*, pages 17–25. Springer, 1994.

[120] Gregor Leander, Mohamed Ahmed Abdelraheem, Hoda AlKhzaimi, and Erik Zenner. A Cryptanalysis of PRINTcipher: The Invariant Subspace Attack. In Phillip Rogaway, editor, *Advances in Cryptology - CRYPTO 2011 - 31st Annual Cryptology Conference, Santa Barbara, CA, USA, August 14-18, 2011. Proceedings*, volume 6841 of *Lecture Notes in Computer Science*, pages 206–221. Springer, 2011.

[121] Ji Li, Takanori Isobe, and Kyoji Shibutani. Converting Meet-In-The-Middle Preimage Attack into Pseudo Collision Attack: Application to SHA-2. In Anne Canteaut, editor, *Fast Software Encryption - 19th International Workshop, FSE 2012, Washington, DC, USA, March 19-21, 2012. Revised Selected Papers*, volume 7549 of *Lecture Notes in Computer Science*, pages 264–286. Springer, 2012.

[122] Leibo Li, Keting Jia, and Xiaoyun Wang. Improved Single-Key Attacks on 9-Round AES-192/256. In Carlos Cid and Christian Rechberger, editors, *Fast Software Encryption - 21st International Workshop, FSE 2014, London, UK, March 3-5, 2014. Revised Selected Papers*, volume 8540 of *Lecture Notes in Computer Science*, pages 127–146. Springer, 2014.

[123] Ruilin Li, Bing Sun, Peng Zhang, and Chao Li. New Impossible Differential Cryptanalysis of ARIA. *IACR Cryptology ePrint Archive*, 2008:227, 2008. `http://eprint.iacr.org/2008/227`.

[124] Yanjun Li, Wenling Wu, and Lei Zhang. Integral Attacks on Reduced-Round ARIA Block Cipher. In Jin Kwak, Robert H. Deng, Yoojae Won, and Guilin Wang, editors, *Information Security, Practice and Experience, 6th International Conference, ISPEC 2010, Seoul, Korea, May 12-13, 2010. Proceedings*, volume 6047 of *Lecture Notes in Computer Science*, pages 19–29. Springer, 2010.

[125] Jiqiang Lu. The (related-key) impossible boomerang attack and its application to the AES block cipher. *Des. Codes Cryptography*, 60(2):123–143, 2011.

[126] Jiqiang Lu, Orr Dunkelman, Nathan Keller, and Jongsung Kim. New Impossible Differential Attacks on AES. In Dipanwita Roy Chowdhury, Vincent Rijmen, and Abhijit Das, editors, *Progress in Cryptology - INDOCRYPT 2008, 9th International Conference on Cryptology in India, Kharagpur, India, December 14-17, 2008. Proceedings*, volume 5365 of *Lecture Notes in Computer Science*, pages 279–293. Springer, 2008.

[127] Stefan Lucks. Attacking Seven Rounds of Rijndael under 192-bit and 256-bit Keys. In *AES Candidate Conference*, pages 215–229, 2000.

[128] Stefan Lucks. The Saturation Attack - A Bait for Twofish. In Mitsuru Matsui, editor, *Fast Software Encryption, 8th International Workshop, FSE 2001 Yokohama, Japan, April 2-4, 2001, Revised Papers*, volume 2355 of *Lecture Notes in Computer Science*, pages 1–15. Springer, 2001.

[129] Hamid Mala. Biclique Cryptanalysis of the Block Cipher SQUARE. *IACR Cryptology ePrint Archive*, 2011:500, 2011. `http://eprint.iacr.org/2011/500`.

[130] Hamid Mala, Mohammad Dakhilalian, Vincent Rijmen, and Mahmoud Modarres-Hashemi. Improved Impossible Differential Cryptanalysis of 7-Round AES-128. In Guang Gong and Kishan Chand Gupta, editors, *Progress in Cryptology - INDOCRYPT 2010 - 11th International Conference on Cryptology in India, Hyderabad, India, December 12-15, 2010. Proceedings*, volume 6498 of *Lecture Notes in Computer Science*, pages 282–291. Springer, 2010.

[131] Mitsuru Matsui. Linear Cryptoanalysis Method for DES Cipher. In Tor Helleseth, editor, *Advances in Cryptology - EUROCRYPT '93, Workshop on the Theory and Application of of Cryptographic Techniques, Lofthus, Norway, May 23-27, 1993, Proceedings*, volume 765 of *Lecture Notes in Computer Science*, pages 386–397. Springer, 1993.

[132] Florian Mendel, Thomas Peyrin, Christian Rechberger, and Martin Schläffer. Improved Cryptanalysis of the Reduced Grøstl Compression Function, ECHO Permutation and AES Block Cipher. In Michael J. Jacobson Jr., Vincent Rijmen, and Reihaneh Safavi-Naini, editors, *Selected Areas in Cryptography, 16th Annual International Workshop, SAC 2009, Calgary, Alberta, Canada, August 13-14, 2009, Revised Selected Papers*, volume 5867 of *Lecture Notes in Computer Science*, pages 16–35. Springer, 2009.

[133] Alfred Menezes, Paul C. van Oorschot, and Scott A. Vanstone. *Handbook of Applied Cryptography*. CRC Press, 1996.

[134] Ralph C. Merkle. One Way Hash Functions and DES. In Gilles Brassard, editor, *Advances in Cryptology - CRYPTO '89, 9th Annual International Cryptology Conference, Santa Barbara, California, USA, August 20-24, 1989, Proceedings*, volume 435 of *Lecture Notes in Computer Science*, pages 428–446. Springer, 1989.

[135] Roger M. Needham and David J. Wheeler. TEA Extensions. Technical report, University of Cambridge, October 1997. Archive available at:`http://www.cl.cam.ac.uk/ftp/users/djw3/xtea.ps`.

[136] Ferguson Niels, Lucks Stefan, Schneier Bruce, Whiting Doug, Bellare Mihir, Kohno Tadayoshi, Callas Jon, and Walker Jesse. The Skein Hash Function Family. Submission to NIST (Round 3), October 2010. Version 1.3, `https://www.schneier.com/cryptography/paperfiles/skein1.3.pdf`.

[137] National Institute of Standards and Technology. Advanced Encryption Standard. *NIST FIPS PUB 197*, November 2001. `http://csrc.nist.gov/publications/fips/fips197/fips-197.pdf`.

[138] Roman Oliynykov, Ivan Gorbenko, Oleksandr Kazymyrov, Victor Ruzhentsev, Oleksandr Kuznetsov, Yurii Gorbenko, Oleksandr Dyrda, Viktor Dolgov, Andrii Pushkaryov, Ruslan Mordvinov, and Dmytro Kaidalov. A New Encryption Standard of Ukraine: The Kalyna Block Cipher. *IACR Cryptology ePrint Archive*, 2015:650, 2015. `http://eprint.iacr.org/2015/650`.

[139] Thomas Peyrin, Henri Gilbert, Frédéric Muller, and Matthew J. B. Robshaw. Combining Compression Functions and Block Cipher-Based Hash Functions. In Xuejia Lai and Kefei Chen, editors, *Advances in Cryptology - ASIACRYPT 2006, 12th International Conference on the Theory and Application of Cryptology and Information Security, Shanghai, China, December 3-7, 2006, Proceedings*, volume 4284 of *Lecture Notes in Computer Science*, pages 315–331. Springer, 2006.

[140] Raphael Chung-Wei Phan. Impossible differential cryptanalysis of 7-round Advanced Encryption Standard (AES). *Information Processing Letters*, 91(1):33–38, 2004.

[141] Bart Preneel, René Govaerts, and Joos Vandewalle. Hash Functions Based on Block Ciphers: A Synthetic Approach. In Douglas R. Stinson, editor, *Advances in Cryptology - CRYPTO '93, 13th Annual International Cryptology Conference, Santa Barbara, California, USA, August 22-26, 1993, Proceedings*, volume 773 of *Lecture Notes in Computer Science*, pages 368–378. Springer, 1993.

[142] Bart Preneel and Vincent Rijmen, editors. *State of the Art in Applied Cryptography, Course on Computer Security and Industrial Cryptography, Leuven, Belgium, June 3-6, 1997. Revised Lectures*, volume 1528 of *Lecture Notes in Computer Science*. Springer, 1998.

[143] Rivest R. The MD5 Message-Digest Algorithm. RFC 1321, April 1992. `https://www.ietf.org/rfc/rfc1321.txt`.

[144] Vincent Rijmen, Joan Daemen, Bart Preneel, Antoon Bosselaers, and Erik De Win. The Cipher SHARK. In Dieter Gollmann, editor, *Fast Software Encryption, Third International Workshop, Cambridge, UK, February 21-23, 1996, Proceedings*, volume 1039 of *Lecture Notes in Computer Science*, pages 99–111. Springer, 1996.

[145] Ronald L. Rivest, Matthew J. B. Robshaw, and Yiqun Lisa Yin. RC6 as the AES. In *AES Candidate Conference*, pages 337–342, 2000.

[146] Ronald L. Rivest, Adi Shamir, and Leonard M. Adleman. A Method for Obtaining Digital Signatures and Public-Key Cryptosystems (Reprint). *Commun. ACM*, 26(1):96–99, 1983.

[147] Li Rongjia and Jin Chenhui. Meet-in-the-middle attacks on 10-round AES-256. *Designs, Codes and Cryptography*, pages 1–13, 2015.

[148] Yu Sasaki. Double-SP Is Weaker Than Single-SP: Rebound Attacks on Feistel Ciphers with Several Rounds. In Steven D. Galbraith and Mridul Nandi, editors, *Progress in Cryptology - INDOCRYPT 2012, 13th International Conference on Cryptology in India, Kolkata, India, December 9-12, 2012. Proceedings*, volume 7668 of *Lecture Notes in Computer Science*, pages 265–282. Springer, 2012.

[149] Yu Sasaki. Meet-in-the-Middle Preimage Attacks on AES Hashing Modes and an Application to Whirlpool. *IEICE Transactions*, 96-A(1):121–130, 2013.

[150] Yu Sasaki and Kazumaro Aoki. Preimage Attacks on 3, 4, and 5-Pass HAVAL. In Josef Pieprzyk, editor, *Advances in Cryptology - ASIACRYPT 2008, 14th International Conference on the Theory and Application of Cryptology and Information Security, Melbourne, Australia, December 7-11, 2008. Proceedings*, volume 5350 of *Lecture Notes in Computer Science*, pages 253–271. Springer, 2008.

[151] Yu Sasaki and Kazumaro Aoki. Preimage Attacks on Step-Reduced MD5. In Yi Mu, Willy Susilo, and Jennifer Seberry, editors, *Information Security and Privacy, 13th Australasian Conference, ACISP 2008, Wollongong, Australia, July 7-9, 2008, Proceedings*, volume 5107 of *Lecture Notes in Computer Science*, pages 282–296. Springer, 2008.

[152] Yu Sasaki and Kazumaro Aoki. Finding Preimages in Full MD5 Faster Than Exhaustive Search. In Antoine Joux, editor, *Advances in Cryptology - EUROCRYPT 2009, 28th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Cologne, Germany, April 26-30, 2009. Proceedings*, volume 5479 of *Lecture Notes in Computer Science*, pages 134–152. Springer, 2009.

[153] Yu Sasaki and Kan Yasuda. Known-Key Distinguishers on 11-Round Feistel and Collision Attacks on Its Hashing Modes. In Antoine Joux, editor, *FSE'11*, volume 6733 of *Lecture Notes in Computer Science*, pages 397–415. Springer, 2011.

[154] Bruce Schneier. A Self-Study Course in Block-Cipher Cryptanalysis. *Cryptologia*, 24(1):18–33, 2000.

[155] Bruce Schneier and John Kelsey. Unbalanced Feistel Networks and Block Cipher Design. In Dieter Gollmann, editor, *FSE'96*, volume 1039 of *Lecture Notes in Computer Science*, pages 121–144. Springer, 1996.

[156] Gautham Sekar, Nicky Mouha, Vesselin Velichkov, and Bart Preneel. Meet-in-the-Middle Attacks on Reduced-Round XTEA. In Aggelos Kiayias, editor, *Topics in Cryptology - CT-RSA 2011 - The Cryptographers' Track at the RSA Conference 2011, San Francisco, CA, USA, February 14-18, 2011. Proceedings*, volume 6558 of *Lecture Notes in Computer Science*, pages 250–267. Springer, 2011.

[157] Claude Elwood Shannon. Communication Theory of Secrecy Systems. *Bell System Technical Journal, Vol 28, pp. 656715*, October 1949.

[158] Kyoji Shibutani, Takanori Isobe, Harunaga Hiwatari, Atsushi Mitsuda, Toru Akishita, and Taizo Shirai. Piccolo: An Ultra-Lightweight Blockcipher. In Bart Preneel and Tsuyoshi Takagi, editors, *Cryptographic Hardware and Embedded Systems - CHES 2011 - 13th International Workshop, Nara, Japan, September 28 - October 1, 2011. Proceedings*, volume 6917 of *Lecture Notes in Computer Science*, pages 342–357. Springer, 2011.

[159] Taizo Shirai and Kyoji Shibutani. Improving Immunity of Feistel Ciphers against Differential Cryptanalysis by Using Multiple MDS Matrices. In Bimal K. Roy and Willi Meier, editors, *FSE'04*, volume 3017 of *Lecture Notes in Computer Science*, pages 260–278. Springer, 2004.

[160] Taizo Shirai, Kyoji Shibutani, Toru Akishita, Shiho Moriai, and Tetsu Iwata. The 128-Bit Blockcipher CLEFIA (Extended Abstract). In Alex Biryukov, editor, *FSE'07*, volume 4593 of *Lecture Notes in Computer Science*, pages 181–195. Springer, 2007.

[161] Simon Singh. *The Code Book*. Fourth Estate, 1999.

[162] Arthur Sorkin. Lucifer, a Cryptographic Algorithm. *Cryptologia*, 8(1):22–42, 1984.

[163] William Stallings. *Cryptography and Network Security - Principles and Practice (3. ed.)*. Prentice Hall, 2003.

[164] Douglas R. Stinson. *Cryptography - theory and practice*. Discrete mathematics and its applications series. CRC Press, 1995.

[165] Bozhan Su, Wenling Wu, Shuang Wu, and Le Dong. Near-Collisions on the Reduced-Round Compression Functions of Skein and BLAKE. In Swee-Huay Heng, Rebecca N. Wright, and Bok-Min Goi, editors, *Cryptology and Network Security - 9th International Conference, CANS 2010, Kuala Lumpur, Malaysia, December 12-14, 2010. Proceedings*, volume 6467 of *Lecture Notes in Computer Science*, pages 124–139. Springer, 2010.

[166] Tomoyasu Suzaki and Kazuhiko Minematsu. Improving the Generalized Feistel. In Seokhie Hong and Tetsu Iwata, editors, *Fast Software Encryption, 17th International Workshop, FSE 2010, Seoul, Korea, February 7-10, 2010, Revised Selected Papers*, volume 6147 of *Lecture Notes in Computer Science*, pages 19–39. Springer, 2010.

[167] Tomoyasu Suzaki, Kazuhiko Minematsu, Sumio Morioka, and Eita Kobayashi. TWINE : A Lightweight Block Cipher for Multiple Platforms. In Lars R. Knudsen and Huapeng Wu, editors, *Selected Areas in Cryptography, 19th International Conference, SAC 2012, Windsor, ON, Canada, August 15-16, 2012, Revised Selected Papers*, volume 7707 of *Lecture Notes in Computer Science*, pages 339–354. Springer, 2012.

[168] Xuehai Tang, Bing Sun, Ruilin Li, Chao Li, and Juhua Yin. A Meet-in-the-Middle attack on reduced-round ARIA. *Journal of Systems and Software*, 84(10):1685–1692, 2011.

[169] Tyge Tiessen, Lars R. Knudsen, Stefan Kölbl, and Martin M. Lauridsen. Security of the AES with a Secret S-Box. In Gregor Leander, editor, *Fast Software Encryption - 22nd International Workshop, FSE 2015, Istanbul, Turkey, March 8-11, 2015, Revised Selected Papers*, volume 9054 of *Lecture Notes in Computer Science*, pages 175–189. Springer, 2015.

[170] Serge Vaudenay. On the Need for Multipermutations: Cryptanalysis of MD4 and SAFER. In Bart Preneel, editor, *Fast Software Encryption: Second International Workshop. Leuven, Belgium, 14-16 December 1994, Proceedings*, volume 1008 of *Lecture Notes in Computer Science*, pages 286–297. Springer, 1994.

[171] Serge Vaudenay. On the Lai-Massey Scheme. In Kwok-Yan Lam, Eiji Okamoto, and Chaoping Xing, editors, *Advances in Cryptology - ASIACRYPT '99, International Conference on the Theory and Applications of Cryptology and Information Security, Singapore, November 14-18, 1999, Proceedings*, volume 1716 of *Lecture Notes in Computer Science*, pages 8–19. Springer, 1999.

[172] Vesselin Velichkov, Nicky Mouha, Christophe De Cannière, and Bart Preneel. The Additive Differential Probability of ARX. In Antoine Joux, editor, *Fast Software Encryption - 18th International Workshop, FSE 2011, Lyngby, Denmark, February 13-16, 2011, Revised Selected Papers*, volume 6733 of *Lecture Notes in Computer Science*, pages 342–358. Springer, 2011.

[173] David Wagner. The Boomerang Attack. In Lars R. Knudsen, editor, *Fast Software Encryption, 6th International Workshop, FSE '99, Rome, Italy, March 24-26, 1999, Proceedings*, volume 1636 of *Lecture Notes in Computer Science*, pages 156–170. Springer, 1999.

[174] Yanfeng Wang, Wenling Wu, and Xiaoli Yu. Biclique Cryptanalysis of Reduced-Round Piccolo Block Cipher. In Mark Dermot Ryan, Ben Smyth, and Guilin Wang, editors, *Information Security Practice and Experience - 8th International Conference, ISPEC 2012, Hangzhou, China, April 9-12, 2012. Proceedings*, volume 7232 of *Lecture Notes in Computer Science*, pages 337–352. Springer, 2012.

[175] David J. Wheeler and Roger M. Needham. TEA, A Tiny Encryption Algorithm. In Bart Preneel, editor, *Fast Software Encryption: Second International Workshop. Leuven, Belgium, 14-16 December 1994, Proceedings*, volume 1008 of *Lecture Notes in Computer Science*, pages 363–366. Springer, 1994.

[176] Wenling Wu, Wentao Zhang, and Dengguo Feng. Impossible Differential Cryptanalysis of Reduced-Round ARIA and Camellia. *J. Comput. Sci. Technol.*, 22(3):449–456, 2007.

[177] Wenling Wu, Wentao Zhang, and Dongdai Lin. Security on Generalized Feistel Scheme with SP Round Function. *I. J. Network Security*, 3(3):215–224, 2006.

[178] Muhammad Reza Z'aba. Analysis of Linear Relationships in Block Ciphers. Master's thesis, Queensland University of Technology, May 2010.

[179] Wentao Zhang, Wenling Wu, and Dengguo Feng. New Results on Impossible Differential Cryptanalysis of Reduced AES. In Kil-Hyun Nam and Gwangsoo Rhee, editors, *Information Security and Cryptology - ICISC 2007, 10th International*

Conference, Seoul, Korea, November 29-30, 2007, Proceedings, volume 4817 of *Lecture Notes in Computer Science*, pages 239–250. Springer, 2007.

[180] Yuliang Zheng, Tsutomu Matsumoto, and Hideki Imai. On the Construction of Block Ciphers Provably Secure and Not Relying on Any Unproved Hypotheses. In Gilles Brassard, editor, *CRYPTO'89*, volume 435 of *Lecture Notes in Computer Science*, pages 461–480. Springer, 1989.

[181] Bo Zhu, Kefei Chen, and Xuejia Lai. Bitwise Higher Order Differential Cryptanalysis. In Liqun Chen and Moti Yung, editors, *Trusted Systems, First International Conference, INTRUST 2009, Beijing, China, December 17-19, 2009. Revised Selected Papers*, volume 6163 of *Lecture Notes in Computer Science*, pages 250–262. Springer, 2009.

Appendix A

Proofs

In this section, we will prove how the base structure which we chose for bicliques in Section 4.6.1 produces non-overlapping keys/messages within a same group and between groups.

A.1 Biclique Structure when IV is known and acts as the message input to block cipher E

For the base message (shown in Fig. 4.15) that is used for the biclique structure in Fig. 4.14(a), our aim is to prove that when Δ_i and ∇_j differences are injected in this base message (as shown in Fig. A.1), we are able to partition the message space into 2^{112} groups with 2^{16} messages in each and the inter and intra group messages generated are non-overlapping. The ∇j_1, ∇j_2, ∇j_3 and ∇j_4 are differences produced from ∇j as shown in Fig. A.2.

i	j_1		
		j_2	
			j_3
j_4			

Figure A.1: Δ_i and ∇_j differences in base message

$$\begin{pmatrix} b_{01} \oplus j_1 \\ b_{12} \oplus j_2 \\ b_{23} \oplus j_3 \\ b_{30} \oplus j_4 \end{pmatrix} = ISB, ISR, IMC \begin{pmatrix} c_{01} \oplus j \\ c_{12} \\ c_{23} \\ c_{30} \end{pmatrix}$$

Figure A.2: Relation between $\nabla j, \nabla j_1, \nabla j_2, \nabla j_3, \nabla j_4$

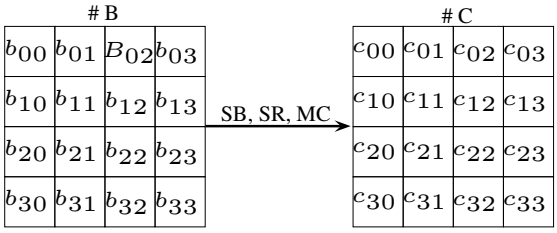


Figure A.3: Relation between #B and #C states

Here, $b_{i,j}$ and $c_{i,j}$ $(0 \leq i, j \leq 3)$ represent the base values of corresponding bytes in the intermediate states #B and #C respectively as shown in Fig. A.3. #B and #C are #3 and #4 states in Fig. 4.14(a).

Aim: Given any two base messages B, B', any two Δ_i differences i, i', any two ∇_j differences j, j' $(0 \leq \text{i,j} \leq 2^8)$, we want to prove that B[i,j] $\neq B[i', j']$ i.e., messages generated are non-overlapping. We will prove this statement case-by-case. Cases (1-4) cover inter group messages whereas Cases (5-7) cover within group messages. For all the proofs discussed below, we will refer to Fig. A.4, A.5, A.6 for better understanding.
Case 1. Given $B \neq B'$, $i = i', j = j'$, b_{00}=b_{10}=b'_{00}=b'_{10}=0, to show: $B[i, j] \neq B'[i', j']$
Proof: We will prove this setting by 'proof by contraposition', i.e., if $B[i, j] = B'[i', j']$, $i = i', j = j'$, b_{00}=b_{10}=b'_{00}=b'_{10}=0, $\implies B = B'$

In Fig. A.6, if $B[i, j] = B'[i', j'] \implies C[i, j] = C'[i', j'] \implies c_{0,2} = c'_{0,2}$, $c_{0,3} = c'_{0,3}$, $c_{1,1} = c'_{1,1}$, $c_{1,2} = c'_{1,2}$, $c_{1,3} = c'_{1,3}$, $c_{2,1} = c'_{2,1}$, $c_{2,2} = c'_{2,2}$,$c_{2,3} = c'_{2,3}$, $c_{3,1} = c'_{3,1}$, $c_{3,2} = c'_{3,2}$ and $c_{3,3} = c'_{3,3}$. Since $C[i, j] = C'[i', j'] \implies c_{0,1} \oplus j = c'_{0,1} \oplus j'$. As $j = j' \implies c_{0,1} = c'_{0,1}$. Hence, **12** bytes in state C and corresponding bytes in state C' share equal values. This relation automatically transcends to related byte positions in B and B' after application of *InvMixColumns*, *InvShiftRows* and *InvSubBytes* (as shown in Fig. A.4), i.e., $b_{0,1} = b'_{0,1}$, $b_{0,2} = b'_{0,2}$, $b_{0,3} = b'_{0,3}$, $b_{1,0} = b'_{1,0}$, $b_{1,2} = b'_{1,2}$, $b_{1,3} = b'_{1,3}$, $b_{2,0} = b'_{2,0}$, $b_{2,1} = b'_{2,1}$, $b_{2,3} = b'_{2,3}$, $b_{3,0} = b'_{3,0}$, $b_{3,1} = b'_{3,1}$ and $b_{3,2} = b'_{3,2}$, 12 bytes in B and B' respectively also have same base values). As we have assumed $B[i, j] = B'[i', j'] \implies b_{1,1} = b'_{1,1}$, $b_{2,2} = b'_{2,2}$ and $b_{3,3} = b'_{3,3}$ as these base values are not affected by Δ_i and ∇_j differences (as seen in Fig. A.6). Since in states B and B', $b_{0,0} = b'_{0,0} = 0$, hence all

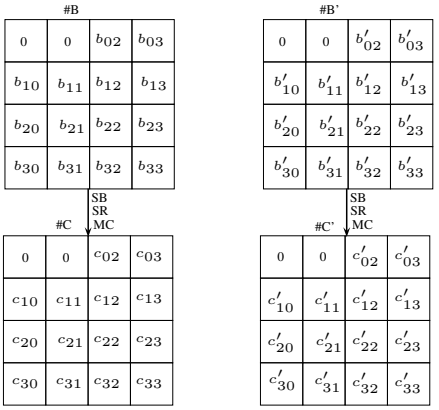


Figure A.4: Relation between base states B and C. The labels inside each box denote the base values of the corresponding byte positions

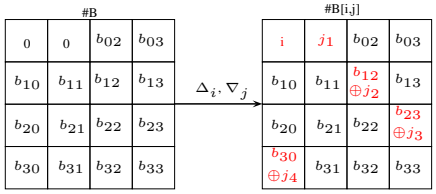


Figure A.5: Modification of state $\#B$ after applying Δ_i and ∇_j differences. Same relation exists between $\#B'$ and $\#B'[i,j]$

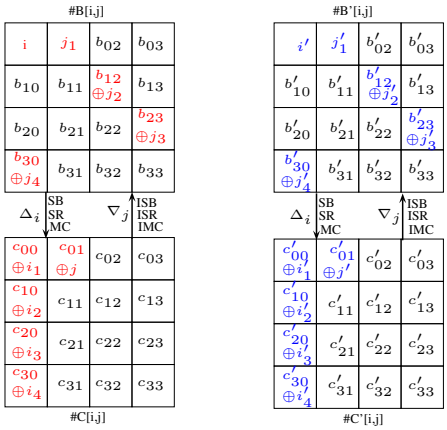


Figure A.6: Relation between states $\#B[i,j]$, $\#C[i,j]$ and $\#B'[i,j]$, $\#C'[i,j]$

16 byte positions in B and corresponding byte positions in B' share same base values. Hence $B = B'$. This proves that our initial proposition is correct.
Case 2. Given $B \neq B'$, $i = i'$, $j \neq j'$, b_{00}=b_{01}=b'_{00}=b'_{01}=0, to show: $B[i,j] \neq B'[i',j']$
Proof: We will prove this setting by 'proof by contradiction', i.e., let us assume if $B \neq B'$, $i = i'$, $j = j'$, b_{00}=b_{10}=b'_{00}=b'_{10}=0, $\implies$ $B[i,j] = B'[i',j']$

In Fig. A.6, if $B[i,j] = B'[i',j'] \implies C[i,j] = C'[i',j'] \implies c_{0,1} \oplus j = c'_{0,1} \oplus j'$. Since j $\neq$ j' $\implies$ $c_{0,1} \neq c'_{0,1}$. As a result after applying *InvMixColumns* and *InvSubBytes* on them the bytes generated i.e., $b_{0,1}$ and $b'_{0,1}$ should also satisfy the relation - $b_{0,1} \neq b'_{0,1}$. But $b_{0,1} = b'_{0,1} = 0$ (as seen in Fig. A.3). Hence, a contradiction arises implying our assumed proposition is wrong. Therefore, our initial proposition is correct.

Case 3. Given $B \neq B'$, $i \neq i'$, $j = j'$, b_{00}=b_{01}=b'_{00}=b'_{01}=0, to show: $B[i,j] \neq B'[i',j']$
Proof: In this setting since $i \neq i'$, hence $B[i,j] \neq B'[i',j']$ always as they will always differ at zeroth byte position (Fig. A.6).
Case 4. Given $B \neq B'$, $i \neq i'$, $j \neq j'$, b_{00}=b_{01}=b'_{00}=b'_{01}=0, to show: $B[i,j] \neq B'[i',j']$
Proof: Proof similar to as discussed in *Case 3.*

Case 5. Given $B = B'$, $i \neq i'$, $j \neq j'$, b_{00}=b_{01}=b'_{00}=b'_{01}=0, to show: $B[i,j] \neq B'[i',j']$
Proof: Proof similar to as discussed in *Case 3.*

Case 6. Given $B = B'$, $i \neq i'$, $j = j'$, b_{00}=b_{01}=b'_{00}=b'_{01}=0, to show: $B[i,j] \neq B'[i',j']$
Proof: Proof similar to as discussed in *Case 3.*

Case 7. Given $B = B'$, $i = i'$, $j \neq j'$, b_{00}=b_{01}=b'_{00}=b'_{01}=0, to show: $B[i,j] \neq B'[i',j']$
Proof: Since $B = B' \implies C = C' \implies c_{0,1} = c'_{0,1}$. As $j \neq j' \implies c_{0,1} \oplus j \neq c'_{0,1} \oplus j' \implies C[i,j] \neq C'[i',j']$ always as they will everytime differ at fourth byte position (Fig. A.6). As a result $B[i,j] \neq B'[i',j']$ always due to bijection relation between states B and C.

Hence we proved that in all cases $M[i,j]$'s so generated are non-overlapping.

Appendix B

Derivation of Eq. 7.3 defined in Section 7.3

$$\begin{pmatrix} AD_x & 95_x & 76_x & A8_x & 2F_x & 49_x & D7_x & CA_x \\ CA_x & AD_x & 95_x & 76_x & A8_x & 2F_x & 49_x & D7_x \\ D7_x & CA_x & AD_x & 95_X & 76_x & A8_x & 2F_x & 49_x \\ 49_x & D7_x & CA_x & AD_x & 95_x & 76_x & A8_x & 2F_x \\ 2F_x & 49_x & D7_x & CA_x & AD_x & 95_x & 76_x & A8_x \\ A8_x & 2F_x & 49_x & D7_x & CA_x & AD_x & 95_x & 76_x \\ 76_x & A8_x & 2F_x & 49_x & D7_x & CA_x & AD_x & 95_x \\ 95_x & 76_x & A8_x & 2F_x & 49_x & D7_x & CA_x & AD_x \end{pmatrix} \times \begin{pmatrix} W_j[8] \\ W_j[9] \\ W_j[10] \\ W_j[11] \\ Uk_1 \\ W_j[13] \\ W_j[14] \\ W_j[15] \end{pmatrix} = \begin{pmatrix} Uk_2 \\ Uk_3 \\ Uk_4 \\ Uk_5 \\ Z_j[12] \\ Z_j[13] \\ Z_j[14] \\ Z_j[15] \end{pmatrix}$$

Using Inverse Mix Column operation, $Z_j[12]$, $Z_j[13]$, $Z_j[14]$ and $Z_j[15]$ can be written as:

$$2F_x{\cdot}W_j[8] \oplus 49_x{\cdot}W_j[9] \oplus D7_x{\cdot}W_j[10] \oplus CA_x{\cdot}W_j[11] \oplus AD_x{\cdot}Uk_1 \oplus 95_x{\cdot}W_j[13] \oplus 76_x{\cdot}W_j[14] \oplus A8_x{\cdot}W_j[15] = Z_j[12] \quad \text{(B.1)}$$

$$A8_x{\cdot}W_j[8] \oplus 2F_x{\cdot}W_j[9] \oplus 49_x{\cdot}W_j[10] \oplus D7_x{\cdot}W_j[11] \oplus CA_x{\cdot}Uk_1 \oplus AD_x{\cdot}W_j[13] \oplus 95_x{\cdot}W_j[14] \oplus 76_x{\cdot}W_j[15] = Z_j[13] \quad \text{(B.2)}$$

$$76_x{\cdot}W_j[8] \oplus A8_x{\cdot}W_j[9] \oplus 2F_x{\cdot}W_j[10] \oplus 49_x{\cdot}W_j[11] \oplus D7_x{\cdot}Uk_1 \oplus CA_x{\cdot}W_j[13] \oplus AD_x{\cdot}W_j[14] \oplus 95_x{\cdot}W_j[15] = Z_j[14] \quad \text{(B.3)}$$

$$95_x{\cdot}W_j[8] \oplus 76_x{\cdot}W_j[9] \oplus A8_x{\cdot}W_j[10] \oplus 2F_x{\cdot}W_j[11] \oplus 49_x{\cdot}Uk_1 \oplus D7_x{\cdot}W_j[13] \oplus CA_x{\cdot}W_j[14] \oplus AD_x{\cdot}W_j[15] = Z_j[15] \quad \text{(B.4)}$$

If we combine the above equations in the following way:

$$CA_x \cdot (Eq\ B.1) \oplus AD_x \cdot (Eq\ B.2) \oplus 49_x \cdot (Eq\ B.3) \oplus D7_x \cdot (Eq\ B.4) \quad \text{(B.5)}$$

We can eliminate the unknown variable Uk_1 and obtain:

$$\begin{aligned} 94_x \cdot W_j[8] \oplus B4_x \cdot W_j[9] \oplus 4E_x \cdot W_j[10] \oplus 7E_x \cdot W_j[11] &= CA_x \cdot Z_j[12] \oplus AD_x \cdot Z_j[13] \\ \oplus\ C0_x \cdot W_j[13] \oplus DA_x \cdot W_j[14] \oplus C5_x \cdot W_j[15] &\quad \oplus\ 49_x \cdot Z_j[14] \oplus D7_x \cdot Z_j[15] \end{aligned} \quad \text{(B.6)}$$

Appendix C

Derivation of Eq. 7.13 defined in Section 7.5

$$\begin{pmatrix} AD_x & 95_x & 76_x & A8_x & 2F_x & 49_x & D7_x & CA_x \\ CA_x & AD_x & 95_x & 76_x & A8_x & 2F_x & 49_x & D7_x \\ D7_x & CA_x & AD_x & 95_X & 76_x & A8_x & 2F_x & 49_x \\ 49_x & D7_x & CA_x & AD_x & 95_x & 76_x & A8_x & 2F_x \\ 2F_x & 49_x & D7_x & CA_x & AD_x & 95_x & 76_x & A8_x \\ A8_x & 2F_x & 49_x & D7_x & CA_x & AD_x & 95_x & 76_x \\ 76_x & A8_x & 2F_x & 49_x & D7_x & CA_x & AD_x & 95_x \\ 95_x & 76_x & A8_x & 2F_x & 49_x & D7_x & CA_x & AD_x \end{pmatrix} \times \begin{pmatrix} W_j[8] \\ W_j[9] \\ W_j[10] \\ W_j[11] \\ W_j[12] \\ W_j[13] \\ W_j[14] \\ W_j[15] \end{pmatrix} = \begin{pmatrix} Z_j[8] \\ Z_j[9] \\ Uk_1 \\ Uk_2 \\ Z_j[12] \\ Z_j[13] \\ Uk_3 \\ Uk_4 \end{pmatrix}$$

Using Inverse Mix Column operation, $Z_j[8]$, $Z_j[9]$, $Z_j[12]$ and $Z_j[13]$ can be written as:

$$AD_x \cdot W_j[8] \oplus 95_x \cdot W_j[9] \oplus 76_x \cdot W_j[10] \oplus A8_x \cdot W_j[11] \oplus 2F_x \cdot W_j[12] \oplus 49_x \cdot W_j[13] \oplus D7_x \cdot W_j[14] \oplus CA_x \cdot W_j[15] = Z_j[8]$$

$$CA_x \cdot W_j[8] \oplus AD_x \cdot W_j[9] \oplus 95_x \cdot W_j[10] \oplus 76_x \cdot W_j[11] \oplus A8_x \cdot W_j[12] \oplus 2F_x \cdot W_j[13] \oplus 49_x \cdot W_j[14] \oplus D7_x \cdot W_j[15] = Z_j[9]$$

$$25_x \cdot W_j[8] \oplus 49_x \cdot W_j[9] \oplus D7_x \cdot W_j[10] \oplus CA_x \cdot W_j[11] \oplus AD_x \cdot W_j[12] \oplus 95_x \cdot W_j[13] \oplus 76_x \cdot W_j[14] \oplus A8_x \cdot W_j[15] = Z_j[12]$$

$$A8_x \cdot W_j[8] \oplus 25_x \cdot W_j[9] \oplus 49_x \cdot W_j[10] \oplus D7_x \cdot W_j[11] \oplus CA_x \cdot W_j[12] \oplus AD_x \cdot W_j[13] \oplus 95_x \cdot W_j[14] \oplus 76_x \cdot W_j[15] = Z_j[13]$$

If we xor together the above four equations, then we get

$$\begin{aligned} Z_j[8] \oplus Z_j[9] \oplus Z_j[12] \oplus Z_j[13] \;=\; & EA_x \cdot W_j[8] \oplus 54_x \cdot W_j[9] \oplus 7D_x \cdot W_j[10] \oplus C3_x \cdot W_j[11] \oplus E0_x \cdot W_j[12] \oplus \\ & 5E_x \cdot W_j[13] \oplus 7D_x \cdot W_j[14] \oplus C3_x \cdot W_j[15] \end{aligned}$$