EXTENSIVE CRYPTANALYSIS OF AUTHENTICATED ENCRYPTION WITH ASSOCIATED DATA ALGORITHM COLM

İLGİLİ VERİ İÇEREN ASILLANMIŞ ŞİFRELEME ALGORİTMASI COLM'un KAPSAMLI KRİPTANALİZİ

SIRRI ERDEM ULUSOY

PROF. DR. MEHMET ÖNDER EFE

Supervisor

ASSOC. PROF. DR. ORHUN KARA

2nd Supervisor

Submitted to

Graduate School of Science and Engineering of Hacettepe University

as a Partial Fulfillment to the Requirements

for the Award of the Degree of Doctor of Philosophy

in Computer Engineering

July 2023

ABSTRACT

EXTENSIVE CRYPTANALYSIS OF AUTHENTICATED ENCRYPTION WITH ASSOCIATED DATA ALGORITHM COLM

Sırrı Erdem ULUSOY

Doctor of Philosophy, Computer Engineering

Supervisor: Prof. Dr. Mehmet Önder EFE

2nd Supervisor: Assoc. Prof. Dr. Orhun KARA

July 2023, 146 pages

The main objective of an Authenticated Encryption with Associated Data (AEAD) algorithm is to keep the encrypted plaintext secret until its tag is validated. There are two main methods related to the cryptanalysis of AEAD algorithms that can render this objective invalid. These methods are plaintext recovery attacks (simulating the decryption oracle) and tag guessing attacks (producing the valid tag of a given ciphertext). There are also various kinds of forgery attacks against AEAD algorithms in which the adversary tries to construct a valid ciphertext. The resistance of COLM against these methods is studied in this thesis. COLM is one of the AEAD algorithms that won the CAESAR Competition in the Defense in Depth use case. The ciphers chosen in the Defense in Depth portfolio are supposed to contain multiple security layers to provide robust security. The main motivation of this thesis is to examine if COLM indeed satisfies defense-in-depth security. In this thesis, we show that COLM is as secure as its secret whitening parameter L. We demonstrate that COLM cannot resist any attacks mounted against AEAD algorithms once L is known. To the best of our knowledge, we give the first example of querying an EME/EMD (Encrypt-linearMix-Encrypt/Decrypt) AEAD scheme in its decryption direction for arbitrary ciphertext, namely, either a forgery

or tag guessing attack. Moreover, we construct SEBC/SDBC (Simulation models of Encryption/Decryption oracle of the underlying Block Cipher) of COLM. These models are the first examples of an authenticated EME scheme simultaneously. The combination of SEBC/SDBC is a powerful tool to mount a universal forgery attack, a tag guessing attack, and a plaintext recovery attack. All of these attacks have $O(N)$ time complexities once L is recovered in the offline phase, indicating that the security of COLM against plaintext recovery and tag guessing attacks is limited by the birthday bound. Besides exploiting SEBC/SDBC, we mount a pair of plaintext recovery attacks and another universal forgery attack by taking advantage of weaknesses in the structure of COLM. Finally, we suggest some improvements to prevent our attacks and build stronger EME schemes.

Keywords: COLM, Cryptanalysis, Authenticated Encryption with Associated Data, CAESAR Competition, Forgery, Tag Guessing, Plaintext Recovery, SEBC, SDBC

ÖZET

İLGİLİ VERİ İÇEREN ASILLANMIŞ ŞİFRELEME ALGORİTMASI COLM'un KAPSAMLI KRİPTANALİZİ

Sırrı Erdem ULUSOY

Doktora, Bilgisayar Mühendisliği

Danışman: Prof. Dr. Mehmet Önder EFE

Eş Danışman: Assoc. Prof. Dr. Orhun KARA

Haziran 2023, 146 sayfa

İlişkilendirilmiş veri içeren asıllamalı şifreleme algoritmalarının (AEAD) esas görevi, bir şifreli bir mesajın etiketi doğrulanmadığı sürece mesajı gizli tutmaktır. AEAD algoritmalarının kriptanalizinde bu görevi geçersiz kılma amacı olan iki temel atak yöntemi mevcuttur. Bu ataklar açık metin ele geçirme atağı (şifre çözme kahininin benzeştirilmesi) ve etiket tahmini atağı (verilen bir şifreli mesajın etiketini üretmek). Bu atakların yanında AEAD algoritmalarına karşı çeşitli sahtecilik atakları da bulunmaktadır. Sahtecilik saldırılarında, saldırgan etiketiyle beraber geçerli şifreli metin üretmeye çalışır. Bu tez çalışmasında COLM algoritmasının bu üç atak türüne karşı direnci incelenmiştir. COLM, CAESAR yarışmasında katmanlı güvenlik kullanım senaryosuna seçilen AEAD algoritmalarından biridir. Katmanlı güvenlik kullanım senaryosu için seçilen şifreleme algoritmalarının sağlam güvenlik sağlamaları için çoklu güvenlik tabakaları içermeleri beklenir. Bu tezin temel motivasyonu, COLM'un gerçekten katmanlı güvenlik sağlayıp sağlamadığının incelenmesidir. Bu tez çalışmasıyla, COLM'un güvenlik seviyesinin gizli beyazlatma parametresi L'nin güvenlik seviyesinde olduğunu gösterdik. L bilindiğinde COLM'un AEAD algoritmalarına karşı yapılan hiçbir atağa karşı güvenli olmadığını

sunduk. Bildiğimiz kadarıyla, EME/EMD (Şifrele-Doğrusal Karıştır-Şifrele/Şifre Çöz) yapıdaki bir AEAD inşasını rasgele şifreli mesajlar için şifre çözme yönünde sorgulayan atakların (sahtecilik ve etiket tahmini ataklarının) ilk örneklerini sunuyoruz. Bunun yanında, COLM'un SEBC/SDBC'ni (yapıtaşı blok şifreleme algoritmasının şifreleme/şifre çözme kahinlerinin benzetim modelleri) inşa ettik. Bu modeller asıllamaları bir EME yapısı için eş zamanlı olarak geliştirilen ilk modellerdir. SEBC/SDBC modellerinin birleşmesi evrensel sahtecilik, anahtar tahmini atağı ve açık metin ele geçirme atakları tertip etmek için güçlü bir araç olmaktadır. L çevrimdışı aşamada hesaplandıktan sonra bu atakların tamamının güvenlik seviyesi $O(n)$ zaman karmaşıklığına düşmektedir. Açık metin ele geçirme ve etiket tahmini ataklarına karşı COLM'un güvenlik seviyesinin doğum günü atağıyla sınırlı olduğu anlaşılmaktadır. SEBC/SDBC'den faydalanmanın yanı sıra, COLM'un yapısındaki zayıflıkları kullanarak bir çift açık metin ele geçirme atağı ve evrensel sahtekarlık atağı tertip ettik. Son olarak, ataklarımızın önlenerek daha güçlü EME yapılarının inşa edilebilmesi için bazı önlemler öneriyoruz.

Anahtar Kelimeler: COLM, Kriptanaliz, İlgili Veri içeren Asıllanmış Şifreleme, CAESAR Yarışması, Sahtecilik, Etiket Tahmini, Açık Metin Tahmini, SEBC, SDBC

ACKNOWLEDGEMENTS

I would like to thank to my advisor, Prof. Dr. Mehmet Önder EFE who made this work possible. It became easier to overcome the difficulties wth his guidance and support. I would also like to thank to my co-advisor Assoc. Prof. Dr. Orhun KARA for working side by side with me and teaching me the perspective of cryptographic research community on the academic work.

I would also like to thank my commitee members Prof. Dr. Ahmet Burak CAN, Prof. Dr. Tolga GİRİCİ, Assoc. Prof. Dr. Murat AYDOS and Assoc. Prof. Dr. Adem TEKEREK for their valuable time to review my work and for giving brilliant feedback and suggestions.

I would also like to thank to my parents, Rafet ULUSOY and Hatice ULUSOY, for giving me a back whenever I need their support.

I would also like to thank to my wife, Handan ULUSOY for her patience while I work on my Ph.D. for long hours. I would also like to thank my children, Nazlı Gülümser ULUSOY and Kardelen Zeynep ULUSOY, who are not aware of anything but able to fill me with energy with their small smiles.

I would also like to thank to my friends, Dr. Atakan ARSLAN, Dr. Abdurrahman BAYRAK and Dr. Mehmet Emin GÖNEN for their help during this process.

CONTENTS

Page

TABLES

FIGURES

ABBREVIATIONS

AEAD	:	**A**uthenticated **E**ncryption with **A**ssociated **D**ata
AES	:	**A**dvanced **E**ncryption **S**tandard
CIA	:	**C**onfidentiality **I**ntegrity **A**uthenticity
CT	:	**C**ipher**t**ext
DES	:	**D**ata **E**ncryption **S**tandard
DHKE	:	**D**iffie **H**ellman **K**ey **E**xchange
$\mathbf{D_K(X)}$	:	Decryption query of the block X with a block cipher
ECC	:	**E**lliptic **C**urve **C**ryptography
$\mathbf{E_K(X)}$	:	Encryption query of the block X with a block cipher
ELmD	:	**E**ncrypt **L**inear-**m**ix **D**ecrypt
GCM	:	**G**alois **C**ounter **M**ode
GMAC	:	**G**alois **M**essage **A**uthentication **C**ode
MAC	:	**M**essage **A**uthentication **C**ode
NIST	:	**N**ational **I**nstitute of **S**tandards and **T**echnology
PT	:	**P**lain**t**ext
RSA	:	**R**ivest **S**hamir **A**dleman
SDBC	:	**S**imulation of **D**ecryption Oracle of the Underlying **B**lock **C**ipher
SEBC	:	**S**imulation of **E**ncryption Oracle of the Underlying **B**lock **C**ipher
SHA	:	**S**ecure **H**ash **A**lgorithm
TDEA	:	**T**riple **D**ata **E**ncryption **A**lgorithm

1. INTRODUCTION

Cryptology is an exciting scientific discipline that underlies both cryptography, the practice of secure communication and data storage in the presence of adversaries, and cryptanalysis, the practice of exploring vulnerabilities in cryptographic structures. Cryptography serves as the common tool to ensure data confidentiality, integrity, authenticity, and non-repudiation in many settings, including e-commerce, online banking, and military communications. Cryptology has a rich and diverse history dating back to ancient times when people used various techniques to keep their messages safe from adversaries. For example, hieroglyphs and other symbols were used to encode messages on papyrus scrolls in ancient Egypt. Cryptography and cryptanalysis have continuously evolved and improved over time. Many noteworthy advances including the famous Caesar cipher, used by Julius Caesar himself, the Enigma machine, a remarkable invention of the Germans for use in World War II, and the AES, which is the main tool for secure communication in the contemporary world. In the present day, cryptography forms the backbone of modern communication and information security.

Cryptology is the discipline that studies the construction and assessment of codes, ciphers, and relevant algorithms used to secure information against unauthorized access. Its objective is not only to create practical tools for secure communication channels and data storage, but also to break or bypass existing security measures. Cryptology is an interdisciplinary field that involves various interconnected areas, such as steganography, which involves hiding information within other data to prevent detection. Since its inception, cryptology has developed and refined numerous methods and techniques to protect sensitive data, including both classical and modern cryptographic approaches. As technology continues to advance and becoming more sophisticated, the importance and significance of cryptology are deeply felt by modern society, making it an indispensable area of study for information security and privacy.

Cryptography is the field concerned with the secure transmission and storage of information

against malicious parties. The techniques employed by cryptography ensure the confidentiality, integrity, authenticity, and non-repudiation of data. Cryptography is utilized in numerous areas, including e-commerce, online banking, and military communications. As the field has evolved over time, modern cryptography heavily relies on mathematical foundations and the utilization of advanced algorithms to encrypt and decrypt data. Cryptography is an ever-evolving field that continually adapts to new security requirements and counter emerging attacks.

Cryptanalysis, on the other hand, aims to find and exploit vulnerabilities in codes and ciphers to gain unauthorized access to information. Cryptanalysis is the core scientific discipline that helps researchers and practitioners evaluate the security of cryptographic systems and develop more reliable protection techniques. This thesis focuses on the cryptanalysis of AEAD schemes, which are commonly used in modern cryptography to safeguard both transmitted and stored data. The ability to exploit vulnerabilities in AEAD schemes is crucial for understanding their security and developing advanced, unbreakable authentication, and encryption methods.

Information security is built upon the concepts of cryptology, cryptography, and cryptanalysis. These concepts protect the confidentiality, integrity, authenticity, and availability of data, collectively known as the CIA triad of information security. The principles of the CIA triad are also adopted in cryptology. The main objective of cryptography is to ensure confidentiality, which involves protecting data from unauthorized reading and modification. Cryptography ensures that only authorized parties can read the information by encryption, and through authentication or signature schemes, only authorized parties can create or modify data. Cryptanalysis, on the other hand, tests the soundness of cryptographic systems and attempts to identify weaknesses in advance. Cryptanalysis is crucial for ensuring both the confidentiality and integrity of data. Cryptology, combination of cryptography and cryptanalysis, not only aims to protect the secrecy and authenticity of data but also provides non-repudiation by verifying the identities of communicating parties. These principles form the fundamentals of the CIA triad and emphasize the significance of cryptology in the extensive discipline of information security.

Cryptographic primitives are the foundational structures of cryptography. Cryptography utilizes these primitives to provide services for the CIA triad by appropriately applying mathematical algorithms and protocols. Transmitted and stored data are protected using these structures. Cryptographic primitives can be classified into symmetric and asymmetric cryptography.

In symmetric cryptography, which is the older form of cryptographic primitives, a single shared key is used for both encryption and decryption. The primary aim of symmetric cryptography is to ensure the confidentiality of sensitive information by encrypting it and preventing unauthorized access. The shared key in symmetric cryptography must be kept secret to prevent adversaries possess the key and imporsante the key holder and gaining the ability to decrypt or modify the information at will. In the contemporary world, the most common symmetric-key cryptographic primitives are TDEA [2] and AES [3]. These algorithms are designed to provide high levels of security while also considering computational efficiency. With these combined properties, these algorithms are suitable for a wide range of applications.

Asymmetric cryptography, in contrast to symmetric cryptography, uses a pair of keys: a public key for encryption and signature verification, and a private key for decryption and signing. Asymmetric cryptography enables confidentiality, authenticity, and integrity. By verifying the identities of the communicating parties, asymmetric cryptography ensures that the data remains intact and authentic. Parties can rely on the data and be certain that it has not been modified in any way. The hallmark of asymmetric cryptography arises in the context of non-repudiation. In symmetric cryptography, where at least two parties possess the same key; therefore, non-repudiation cannot be provided as a service. The most widely used asymmetric cryptographic primitives include the RSA algorithm, used for secure communication and digital signatures, DHKE, a popular key exchange algorithm, and ECC, an algorithm commonly used for key exchange and digital signatures.

Cryptographic hash functions are another significant cryptographic primitive that serves to ensure data integrity. Hash functions generate a fixed-length hash value or message

digest from variable-length input data. Cryptographic hash functions are designed to be collision-resistant and one-way, meaning it should be difficult to generate the original message from the hash value or a second message from a given value. Acting as a one-way function, cryptographic hash functions are considered to produce a digital fingerprint of the input. By comparing the hash value of the original data with the received data, the integrity of the data can be easily verified. It should be noted that hash functions do not utilize a key, so any adversary can alter the original data and compute the hash of the altered data to deceive the receiving party.

Symmetric and asymmetric cryptographic primitives, as well as hash functions, are at the core of securing modern communication and data storage systems by serving as fundamental tools for confidentiality, integrity, and authenticity. The usage area of these cryptographic primitives also varies according to their security level and application performance. MACs are used to provide both authenticity and integrity of a message, bridging the gap in providing integrity with high performance. MACs generate fixed-size values similar to hash functions; however, MACs use a secret key to generate the fixed-size value. In this way, only parties holding the secret key are able to generate and verify this value. With their performance and security properties, MACs are also an important structure in many cryptographic applications, including secure communication and electronic transactions. Overall, cryptographic primitives, including symmetric and asymmetric cryptography, hash functions, and MACs, form a powerful toolkit and work in harmony according to their service and performance capabilities against a wide range of threats and attacks, ultimately achieving the goals of the CIA triad. In this way, modern communication and data storage systems remain secure.

In the digitally connected contemporary world, considering a single security issue in the CIA triad does not provide the required protection. When multiple cryptographic building blocks are used together, the system becomes overwhelmed due to high complexity and inefficiency. To reduce complexity in communication and storage systems, AEAD schemes have been developed. In an AEAD scheme, both confidentiality and integrity tasks are handled together. AEAD schemes take on the duties of both symmetric encryption and message

authentication codes, which take care of confidentiality and integrity, respectively. Phillip Rogaway introduced AEAD to the literature with his paper "Authenticated-Encryption with Associated-Data" [4] in 2002. Rogaway provided a new framework for building AEAD schemes. With this framework, it became possible to perform encryption and authentication functions with a single key in a single structure. The structure introduced by Rogaway supports not only the data to be kept confidential but also the data that requires only authentication, such as headers and metadata. In this way, a wide range of applications can be protected in a more secure and efficient manner. The benefits of AEAD are not limited to simplicity and security. AEAD algorithms eliminate certain types of attacks, such as chosen ciphertext attack and chosen plaintext attacks (in some cases), as a natural consequence of their definition. As AEAD provides protection for additional data besides the confidential data, application designs become more flexible. Rogaway's paper on AEAD has become a guiding light in the development of new schemes that provide both improved security and efficiency. As the importance of secure communication and data storage continues to grow, the cruciality of AEAD becomes crystal clear for modern cryptography. Today, even in the announcement of the competition of lightweight ciphers [5], NIST requires the candidates to provide AEAD.

Cryptographic primitives and structures have been developed over the years, and evaluating their security level has been an open problem. Researchers from all over the world have analyzed and improved cryptographic algorithm design and evaluation techniques. As part of the standardization processes, cryptographic competitions are adopted as a good use case to draw the attention of researchers and cryptographers from all over the world to the competing candidates. A cryptographic competition is a contest challenging cryptographers and researchers to design and evaluate new cryptographic algorithms and protocols. Various entities, including academic institutions, private companies, and government agencies organize these competitions, which can last several years. The aim of these competitions is to identify the best algorithm or protocol for a specific security application, such as encryption, digital signatures, or authentication. The widely used encryption standard AES [3], hash function SHA3 [6] are the most well-known examples of standards developed through

competitions. These competitions are beneficial not only in the standardization process, but they also contribute to understanding and improving cryptographic and cryptanalysis methods. To take advantage of the competitive approach, CAESAR competition [7] was held to select a final portfolio of AEADs.

CAESAR competition was announced in January 2013, and 57 competitors participated in the competition. The submitted candidates competed in three specified use cases: lightweight use case, high-performance use case, and defense in depth use case. The terms of the lightweight use case were set with the aim of resource-constrained devices, the terms of the high-performance use case were set with the aim of high-bandwidth applications, and the terms of the defense in depth use case were set with the aim of systems where the highest level of security is a subject of concern.

The evaluation of the candidates was based on their security, performance, and other relevant factors. The importance of the evaluation metric differs from use case to use case. The decision committee announced that only public evaluations in the literature would be taken into account in the decision process to rely on the collective evaluation of experts. The selection of the final portfolio of the AEADs took four rounds of evaluation, with two choices for each of the three defined use cases included in the final portfolio. Our research focuses on the extensive cryptanalysis of COLM algorithm [8] which was selected for the defense in depth use case portfolio. Defense in Depth is a multilayered security approach in the formal definition. A cryptographic scheme designed with the Defense in Depth principle should possess a set of desirable properties to maintain its security and reliability in various scenarios. In CAESAR competition, the desired properties mainly include authenticity despite nonce misuse, limited privacy damage from nonce misuse, authenticity despite the release of unverified plaintexts, limited privacy damage from the release of unverified plaintexts, and robustness in scenarios where large amounts of data ($PT - CT$ pairs) exist.

COLM cipher is an AEAD scheme designed while CAESAR competition was ongoing. Most of the AEAD schemes participating in the competition were designed before the competition began, but COLM was not originally entered into the competition by its

designers. Initially, two distinct research groups independently submitted two different AEAD algorithms, ELmD [9] and COPA [10]. ELmD and COPA algorithms share common underlying structures. After the second round of the competition, the designers of the two algorithms decided to merge their algorithms into a single one, COLM. Although the new cipher preserves many of the design properties of the preceding algorithms, it also has some modifications to improve performance without sacrificing security. For example, COLM adopted COPA's encrypt-linearmix-encrypt style instead of ELmD's encrypt-linearmix-decrypt style, and COPA's simpler linear mix layer is adopted to process the AD, while ELmD's more complex linear mix layer is adopted to process the PT in COLM. While designing the new cipher by combining the properties of the predecessors, the designers kept in mind the goal to increase and/or maintain the high performance of the AEAD schemes while maintaining the desired level of security in the resulting cipher.

In this thesis, our aim was to conduct an extensive analysis of COLM and discover any possible vulnerabilities in it. At the outset of AEAD algorithms, the oracle is expected to return $\perp$ for illegitimate queries. We achieved our goal by identifying weaknesses in AEAD COLM, notably by demonstrating that it can be queried in the decryption direction, thus breaking the initial rule of AEAD structures. In another part of our study, we built SEBC and SDBC models for COLM. Consequently, we were able to mount universal forgery, plaintext recovery, and tag guessing attacks against COLM. Furthermore, exploiting the weaknesses in the structure of COLM, we mounted various universal forgery, existential forgery, and plaintext recovery attacks in different scenarios. Lastly, we introduced how the COLM oracle can be exploited to gather data for mounting key recovery attacks against the underlying block cipher of COLM. We explained how this data can be utilized to mount key recovery attacks against round-reduced AES, considering that AES is the chosen underlying cipher for COLM.

Throughout our research, we have demonstrated the security vulnerabilities of the COLM cipher and highlighted the importance of continuous analysis and evaluation of cryptographic algorithms. By doing so, our objective was to expose potential weaknesses that adversaries may exploit and emphasize new criteria that should be considered in the design and

implementation of secure AEAD schemes. The generalization of attacks presented in this thesis to generalized ELmD/ELmE structures remains a topic for future research topic.

1.1. Scope Of The Thesis

The main focus of this study is the security level of the AEAD scheme COLM, which has been selected for the Defense in Depth portfolio of the CAESAR competition. As a member of the Defense in Depth portfolio in a cryptographic competition, COLM is expected to demonstrate exceptional resistance against major attacks developed against AEADs. The primary objective of an AEAD scheme is to prevent unauthorized decryption queries to the oracle. In this thesis, we not only successfully query a Defense in Depth AEAD structure, but also build simulation models of the underlying block cipher in both encryption and decryption directions. By simulating the underlying block cipher in both directions, we deepen our understanding of the cryptographic properties of COLM. Additionally, we introduce a range of attacks against COLM, with a particular focus on attacks specifically designed for AEADs, such as universal forgery, existential forgery, plaintext recovery, and tag guessing attacks.

Both the structure and the underlying block cipher of an AEAD scheme are crucially important for ensuring its security. Attackers may target either the structure or the underlying block cipher. One way to attack an AEAD scheme is to exploit the structure to build simulation models of the underlying block cipher. Once an attacker successfully builds a simulation of the underlying block cipher, she can mount various types of attacks. As part of our analysis, we demonstrate how the COLM oracle can be queried to simulate the underlying block cipher in both encryption and decryption directions. We take advantage of the flaws in the structure of COLM to build these models. The presence of these flaws indicates that COLM is not capable of providing the expected level of security as previously believed.

In the literature, there are specific attacks, such as forgery attacks, plaintext recovery attacks, and tag guessing attacks, that primarily target AEAD structures. AEAD algorithms should be

designed to withstand these attacks. In the scope of our study, we successfully mounted these attacks against COLM using various methods. Specifically, we introduced forgery attacks, as well as plaintext recovery and tag guessing attacks, both by utilizing our simulation models and by exploiting flaws in the design of COLM.

In summary, we present a comprehensive analysis of the AEAD scheme COLM. Our study reveals that COLM is vulnerable to a range of attacks, including universal and existential forgeries, plaintext recovery, and tag guessing attacks. These attacks against COLM can be mounted in various methods, namely by building simulation models of the underlying block cipher and exploiting design flaws in COLM. This thesis demonstrates that COLM is not as strong as previously believed and does not meet the desired and required 128-bit security level against attacks like plaintext recovery and tag guessing. It is necessary to redesign the COLM scheme with further improvements to enhance its security and ensure that it provides the desired level of protection.

1.2. Contributions

Our study makes several contributions to the field of AEAD scheme security analysis, which serve as guiding principles in the design of future AEAD algorithms. One of our key contributions is the introduction of an illegitimate decryption query in the COLM AEAD scheme, which reveals the parameter $S = D_K(0)$. Additionally, we built SEBC and SDBC models of COLM. An SDBC model of ELmD and an SEBC model of COLM were built in [11] and [12], respectively. Notably, our SDBC model allows for querying an AEAD structure in the decryption direction. This is significant because an AEAD scheme should resist decryption direction queries by requiring tag validation. By querying the underlying block cipher in both encryption and decryption directions, an adversary gains significant capabilities against the AEAD scheme and can mount various important attacks. Through the simulation of the underlying cipher using decryption direction queries, we have made important contributions to the analysis of AEAD cipher security.

Building SEBC and SDBC models of COLM is not the sole focus of our study. In another part of our study, we exploit the structure of COLM and introduce plaintext recovery and tag guessing attacks against COLM. These attacks are the first of their kind in the literature, targeting a CAESAR competition winner. The significance of our attacks is heightened by being the first to explore such vulnerabilities in a competition winner. Our results highlight the current weaknesses and emphasize the need for further improvements to be considered in future designs, ensuring that they provide the desired level of protection.

Finally, we evaluated the effects of our results on the complexity of mounting key recovery attacks against the underlying block cipher of COLM. In this scope, we examined how MitM, impossible differential, and biclique attacks can be mounted against round-reduced AES, the underlying block cipher of COLM. As part of the complexity analysis, we also calculated the required number of COLM queries to mount these attacks.

To sum up, our contributions involve attacks against the CAESAR winner COLM and its underlying block cipher. They can be summarized as follows:

- The first illegitimate decryption query of a CAESAR winner, namely COLM.
- The first simulation models of the underlying block cipher in both encryption and decryption directions of COLM.
- Building a simulation model of the underlying block cipher by querying an AEAD scheme in the decryption direction.
- Various AEAD attacks against a CAESAR winner, including:
  - Forgery attacks.
  - Tag guessing attack (the first one against a CAESAR winner).
  - Plaintext recovery attacks (the first one against a CAESAR winner).
- Complexity analysis of key recovery attacks, namely the MitM attack, impossible differential attack, and biclique attack, against round-reduced AES, the underlying block cipher of COLM.

- Discussion of the required number of AEAD queries for these attacks.

Overall, our findings will contribute to the development of new design principles for future AEAD schemes and advance the design and security analysis of AEAD ciphers.

1.3. Organization

The organization of the thesis is as follows:

- Chapter 1 presents our motivation, contributions, and the scope of the thesis.
- Chapter 2 provides background information on cryptography and AEAD schemes.
- Chapter 3 summarizes the existing literature by reviewing works on AEAD schemes and COLM.
- Chapter 4 introduces how the parameter $S = D_K(0)$ can be recovered as a tag guessing and plaintext recovery attack against COLM.
- Chapter 5 explains the construction of the SEBC and SDBC models of COLM.
- Chapter 6 demonstrates the utilization of the models introduced in Chapter 5 to mount universal forgery, tag guessing, and plaintext recovery attacks against COLM.
- Chapter 7 introduces how the structure of COLM can be exploited to mount plaintext recovery with missing tag.
- Chapter 8 demonstrates how CT-blocks can be permuted to mount various universal forgery and plaintext recovery attacks against COLM.
- Chapter 9 gives a complexity analysis of using the COLM AEAD scheme to mount MitM, impossible differential, and biclique key recovery attacks against round-reduced AES, the underlying block cipher of COLM.
- Chapter 10 summarizes the thesis and discusses possible future directions.

2. BACKGROUND OVERVIEW

2.1. Milestones of Cryptography

Humanity's need for secure communication dates back to ancient times. The first known cryptographic method is the Caesar Cipher, named after Julius Caesar, the Roman general and statesman. The Caesar Cipher is the oldest cryptographic method and represents the first instance of symmetric encryption, a simple substitution cipher. The understanding of secure communication and the structure of symmetric encryption have undergone significant changes since the time of the Caesar Cipher. As time has passed, advancements in practicality have influenced the implementations. In [13], Kerckhoffs published six principles for a secure and practical communication system, which have been widely adopted. These principles for secure communication are as follows:

1. The system should be mathematically undecipherable, and if that is not possible, it should still be significantly difficult to decipher;
2. The security of the system should not be compromised by its publicity or capture by the enemy;
3. The key must be communicated and recalled without the need for written notes, and it must also be easy for correspondents to change the key at will;
4. The system must have been adapted for telegraph communication
5. The system must be portable enough to be transported or operated by a single person;
6. Ultimately, the system must be user-friendly and not require detailed guidelines.

In the modern approach, there have been changes in the interpretation of these principles. The Principle 1 has been widely adopted, and the mathematical computation requirements to decipher a system are well-defined for various systems. With the advancements

in technology, it has become impossible to design a secure system that can fulfill the requirements stated in Principle 3, Principle 5, and Principle 6 using only human capabilities. However, the responsibilities outlined in these principles have been transferred to devices, and the design of these devices adheres to the principles. Present-day communication systems still employ the same signaling approach as telegraph systems, thus following Principle 4 as is. Finally, Principle 2 remains at the core of cryptographic system design in the contemporary world.

In [14], Claude Shannon published his works on cryptography, focusing on the concept of secrecy in general. He proved that ultimate confidentiality is only achievable when the one-time pad is utilized. In other words, perfect secrecy is possible only when the length of the key is at least as long as the message. In his work, he also remodeled Kerckhoff's 2^{nd} Principle as "the enemy knows the system."

In the following years, questions regarding the integrity and authenticity of messages arose, leading to various studies [15–18] that discussed and suggested solutions to preserve the authentication and integrity of messages.

Another significant milestone in the history of cryptography is the development of public-key cryptography. With the invention of public-key cryptography, digital signatures and key agreement methods became available. Signatures provide an alternative for authentication, with their own benefits and drawbacks. The most common public-key methods used today are Diffie-Hellman [19] and RSA [20].

Cryptographic structures aim not only to protect parties from third parties but also from each other. These structures include cryptographic protocols such as:

1. Zero-knowledge proofs: cryptographic primitives that allow a party to prove a statement without revealing anything beyond the truth of the statement [21].

2. Secure multiparty computation: cryptographic protocols through which distributed parties can perform computations while keeping their information secret [22].

3. Homomorphic encryption: cryptographic primitives that enable algebraic computations to be performed over encrypted data [22].

2.2. Details of AEAD Schemes

AEAD schemes are built to replace both symmetric encryption and MACs. In symmetric encryption, secrecy is provided by a pre-shared key between parties, and third parties not owning the key are unable to access the hidden message. Symmetric encryption schemes may use an IV. The notation for symmetric encryption is:

$$CT = E_K(PT) \text{ or } CT = E_K(IV, PT)$$

$$PT = D_K(CT) \text{ or } PT = D_K(IV, CT)$$

Unlike symmetric encryption, the message (PT) is public in MAC structures. The primary aim of MAC structures is to prove that the message is generated by a party having the secret key. MAC schemes may also utilize an IV. The notation for MAC is:

$$tag = MAC_K(PT) \text{ or } tag = MAC_K(IV, PT)$$

$$valid \times \perp = Verify_K(PT, tag) \text{ or } valid \times \perp = Verify_K(IV, PT, tag)$$

Authenticated encryption with associated data (AEAD) is a cryptographic primitive that provides both confidentiality and integrity of the data simultaneously in a single structure. AEAD schemes consider data to consist of two parts: associated data (e.g., headers or metadata), where only integrity is important, and the message, where both integrity and confidentiality are important. Before the term AEAD was introduced in the literature, confidentiality and integrity were provided by different cryptographic primitives: encryption and message authentication codes (MACs), respectively. AEAD schemes enable the achievement of joint confidentiality and integrity aims in a single structure with a single key setting. Due to lower processing complexity and simpler usage, AEAD rapidly replaces

its conventional alternatives, encryption and MACs, and is particularly preferred in secure communication channels. In today's interconnected world, AEAD schemes are increasingly accepted as a means of secure data transmission. AEAD schemes are widely adopted in communication protocols, including SSL/TLS [23], SSH [24], and IPSec [25], amplifying the importance of AEAD schemes.

AEAD schemes generally utilize a randomly generated nonce to strengthen the security against chosen plaintext attacks. Since the authenticity of the message is also protected with a tag, messages are not released unless the tag is verified. This control mechanism prevents the query of chosen ciphertext. With the additional terms of nonce, tag, and not releasing unverified plaintext, AEAD schemes have their own notations as follows:

$$(CT, tag) = E_K(N, A, M)$$

for encryption and

$$M \times \perp = D_K(N, A, C, T)$$

for decryption. During encryption, AEAD schemes take a nonce and optional additional data, in addition to the message, and produce a ciphertext and tag to protect the confidentiality and authenticity of the ciphertext. In symmetric encryption, the message is always released when the decryption oracle is queried. In contrast, an AEAD scheme releases the symbol $\perp$ instead of the message when the tag cannot be verified.

In the standard procedure, AEAD implementations generate the nonce as part of the data processing. In other words, a user cannot determine and provide the nonce externally. However, designers may choose to give attackers the option to determine the nonce in order to claim a higher security level. This concept is referred to as nonce-misuse in the literature. Another security level that designers may aim for is to protect against the release of unverified plaintext. Although, in a flawless operation, an AEAD scheme does not release unverified plaintext, such releases may occur due to implementation errors or side-channel attacks. To mitigate these flaws and achieve a stronger security level, designers may aim for a scenario

where the partial release of unverified plaintext does not compromise the confidentiality or integrity of the unreleased part.

2.3. AEAD Built Methods

There are various alternative methods for building AEAD schemes. The most fundamental of these methods are:

- Encrypt-then-MAC (EtM)
- Encrypt-and-MAC (E&M)
- MAC-then-Encrypt (MtE).

Encrypt-then-MAC is the only method proven to be strongly unforgeable. The most common AEAD scheme, AES-GCM, is built using this method. In this method, a subkey is derived from the main key, and authentication is provided with the subkey. Unlike the other methods, the ciphertext is not decrypted in this method unless it is verified. During encryption, encryption is performed first, and then the tag is generated. During decryption, decryption and verification can be performed in parallel. However, to avoid the risk of leaking unverified plaintext, the tag is verified first, and the plaintext is decrypted only if the tag is valid.

Encrypt-and-MAC is the method in which the ciphertext and the tag are produced in parallel. Before verifying the tag, the ciphertext has to be decrypted, which introduces the risk of plaintext leakage. The E&M method is used in SSH.

MAC-then-Encrypt is the method where the tag is produced first and then encrypted together with the plaintext. In this method, the plaintext has to be decrypted to be verified, similar to the E&M method. Therefore, the risk of plaintext leakage also exists in this method. Attacks taking advantage of padding errors, such as Lucky Thirteen [26], can be mounted against this model. This method is used in SSL/TLS implementations. In this

method, encryption and MAC algorithms are run sequentially during both encryption and decryption.

2.4. Attacks against AEAD

Since the confidentiality and authenticity of the data are aimed to be protected simultaneously in AEAD schemes, they are expected to resist attacks targeting either authenticity or secrecy. AEAD schemes must have strong structures both to detect any unauthorized modification of the data, whether it is on the associated data or the message, and to keep the data secret unless its authentication and integrity are verified. Some of the common attacks that AEADs may encounter are forgery attacks, tag guessing attacks, and plaintext recovery attacks.

In forgery attacks, an attacker tries to construct a valid ciphertext and tag pair. There are two main types of forgery attacks:

1. Existential Forgery
2. (Semi-)Universal Forgery.

For the definitions of forgery attacks, we mainly adopt the definitions in [11] and [12]. According to the adopted definitions, in an existential forgery, the attacker generates a valid ciphertext and tag pair whose plaintext is unknown. In a universal forgery attack, the attacker generates a valid ciphertext and tag pair and knows the corresponding plaintext without querying the forged ciphertext and tag pair. A semi-universal forgery attack is a variant of the universal forgery attack where the attacker may query the ciphertext to specify the plaintext, but she has to modify the associated data in her query. Since the different types of forgeries have different aims, they require different levels of computational effort. Depending on the sensitivity of the data, forgery attacks might be very destructive, giving the attacker a chance to modify classified data.

Another attack method aiming to break the authenticity of the message is the tag guessing attack. In a tag guessing attack, the attacker tries to build the tag of an associated

data-ciphertext pair. In this attack model, either an authentication tag might be missing or the attacker wants to modify a ciphertext without knowing the original tag. The attacker may guess different tags for the given associated data-ciphertext pairs until the valid tag is found or query any data in the AEAD oracle. The guessing procedure might be brute force or based on cryptanalysis techniques. To be considered safe, the computational security level of an AEAD scheme should not be lower than the minimum of the tag or the key length. In cases where the key is shorter than the tag, the attacker may guess brute force to recover the key. Therefore, the key has to be at least as long as the tag length to prevent tag guessing attacks.

A further attack to bypass the verification of an AEAD scheme is a plaintext recovery attack. Plaintext recovery attacks achieve almost the same result as tag guessing attacks. In a tag guessing attack, the attacker aims to produce the valid tag to decipher a given $AD - CT$ pair. On the other hand, in a plaintext recovery attack, the attacker aims to decipher the plaintext directly without querying the $AD - CT - tag$ tuple. Plaintext recovery attacks may cause similar issues as tag guessing attacks. The computational complexity against plaintext recovery attacks should be the same as that against tag guessing attacks.

To withstand the explained attacks, an AEAD scheme has to be designed and implemented carefully. Robust security mechanisms have to be put in place to prevent any unauthorized modification of the data, including associated data, ciphertext, and the tag. To meet the necessary security targets, designers may benefit from unpredictable nonces or IVs, as well as strong cryptographic primitives. To have confidence in the claimed security level of the designed scheme, the AEAD scheme has to be rigorously cryptanalyzed before utilization. After carefully following these design rules and conducting thorough analysis, organizations can be sure of the safety of their sensitive data from access or modification by third parties.

2.5. AES-GCM: NIST's Recommended Mode of Operation for AEAD

NIST's recommended block cipher mode of operation, AES-GCM, is the most ubiquitous AEAD scheme. AES-GCM is composed of the symmetric block cipher AES being used

in counter mode of operation and the GCM for authentication. Some of the advantages of AES-GCM are:

1. It is parallelizable
2. It utilizes a block cipher; therefore, it can be modified by just changing the underlying block cipher.
3. It can be implemented efficiently on different platforms.

Due to its advantages, AES-GCM is widely adopted and implemented in many cryptographic protocols and libraries. Additionally, AES-GCM is efficient in terms of computation, making it suitable for lightweight operations such as mobile devices and IoT devices.

AES-GCM derives its strength from its two components: AES block cipher and the GCM authentication mode. Furthermore, to enhance the unforgeability of the system, the lengths of the AD and the PT are taken into account when calculating the tag. Masking the tag with the encryption of IV is another protection procedure to make the system resistant against forgery and tag guessing attacks. With all of its security measures, AES-GCM provides a high level of security and integrity protection for sensitive data, making it a popular choice for general-purpose usage. On the other hand, the repetition of the IV severely degrades the security level of AES-GCM and may result in catastrophic security violations. Extra attention should be paid to ensure a unique IV for each message. While AES-GCM is a general-purpose design, for certain operations with a specific purpose, it may be outperformed by another AEAD scheme designed with dedicated purposes such as security requirements, performance, platform, and other factors.

2.6. CAESAR Competition

To fulfill the need for purpose-specific AEAD requirements, CAESAR competition was organized in 2013. The competition consisted of four rounds, with candidates being eliminated based on public discussions regarding their performance and security.

In the final round, there are seven candidates:

- ACORN
- AEGIS
- Ascon
- COLM
- Deoxys
- MORUS
- OCB.

The competition defined three use cases for specific purposes:

1. Lightweight applications
2. High-performance applications
3. Defense in depth.

The CAESAR finalists were announced in [27], and the selection criteria for each use case were also listed in detail.

The lightweight applications category aims to choose a cipher that can run efficiently on resource-constrained environments, particularly 8-bit CPUs. The selected ciphers should have a small hardware footprint and/or be compatible with 8-bit CPUs. Since these ciphers are designed for resource-constrained environments, they are expected to process short messages in general. Resistance against side-channel attacks is also desired. Performance-wise, the ciphers should be efficient in terms of energy/bit on hardware and fast on 8-bit CPUs.

The high-performance applications category aims to choose a cipher that can efficiently run on advanced hardware and process large amounts of data. The primary target hardware is 64-bit CPUs and/or dedicated hardware, with an alternative target being 32-bit CPUs. The processing time for data should not vary significantly based on the message length.

The defense in depth category does not have a specific target running medium like the other use cases. Instead, it focuses solely on resistance to harsh attack scenarios, specifically nonce misuse and the release of unverified plaintext. The most critical scenario involves protecting authenticity even if the nonce is misused. Another scenario deals with mitigating partial privacy violations caused by nonce misuse. Similarly, protecting authenticity and privacy is desired if some plaintext leaks without being verified. Lastly, this use case aims to be resistant to attacks requiring excessive amounts of data.

After public reviews by experts and the designers' public defenses, two ciphers were chosen for each use case. The selection committee expressed a preference for the lightweight and defense in depth use cases, while for the high-performance use case, they announced a draw between the winners. The final portfolio is as follows:

1. Lightweight use case:
    1. Ascon
    2. ACORN
2. High-performance use case:
    1. AEGIS-128
    2. OCB
3. Defense in depth use case:
    1. Deoxys-II
    2. COLM

2.7. CAESAR Winner COLM and its Predecessor

2.7.1. AES-COPA

AES-COPA is an ELmE (Encrypt-Linear mix-Encrypt) type block cipher designed as an authenticated encryption algorithm.

AES-COPA uses AES as the underlying block cipher, and all three key length options (128 bits, 192 bits, 256 bits) are adopted. Another security input for AES-COPA is a public message number, serving as a nonce, with a constant length defined to be 128 bits. To provide authenticity and integrity, AES-COPA generates a tag of variable length ranging from 64 bits to 128 bits.

To process the data, AES-COPA employs both linear mixing and symmetric encryption. In the linear mixing phase, the value of a block is calculated by performing an XOR operation on the previous blocks. In the symmetric encryption layer, the blocks are encrypted using the chosen version of AES algorithm with a symmetric key K.

Before a cryptographic operation, the subkey $L = E_K(0)$ is produced by encrypting zero-vector 0^{128} with the key K. This subkey is then used to mask all input and output blocks. In the masking procedure, each block is masked with a different multiple of the L-value. For each of the AD, PT and CT a different multiplicative of L is chosen as the base value. The masking value for each block is obtained by multiplying the masking value of the previous block with 2 in $GF(2)$. The base values are $3^2 \cdot L$, $3 \cdot L$ and $2 \cdot L$ for AD, PT and CT respectively. The formulations of the whitening masks differ in the last blocks, which will be explained further in the process description.

The main processes in AES-COPA can be listed as follows:

1. Processing AD
2. Encryption

3. Decryption

4. Tag generation.

Since the block length of AES, the underlying block cipher, is 128 bits, the input data is divided into 128-bit subblocks. If the last subblock is not a complete block, it is padded with the pattern 10^* where $^* = 127 - |T^*[l]|$. The data is divided into subblocks as follows:

$$A{=}(A[1], \ldots, A[\alpha - 1], A^*[\alpha]),$$
$$M{=}(M[1], \ldots, M[l - 1], M^*[l]),$$
$$C{=}(C[1], \ldots, C[l - 1], C^*[l]).$$

The encryption and decryption block diagrams of the algorithm are given in Figures 2.1 and 2.2, respectively. In these figures, the E_K block and D_K block denote AES encryption and decryption operations, respectively.

Previously, we mentioned that during encryption and decryption, each of the AD, PT, and CT blocks are XORed with different whitening masks. Whitening masks are derived according to the equations below:

$$\Delta_{A[i]} = \begin{cases} 3^3 \cdot 2^{i-1} \cdot L & \text{if } i < \alpha, \\ 3^5 \cdot 2^{i-1} \cdot L & \text{if } i = \alpha \;\&\; \left|A^*[\alpha]\right| < 128, \\ 3^4 \cdot 2^{i-1} \cdot L & \text{if } i = \alpha \;\&\; \left|A^*[\alpha]\right| = 128. \end{cases}$$

$$\Delta_{M[i]} = \begin{cases} 7 \cdot 2^{i-1} \cdot 3 \cdot L & \text{if } i = l \;\&\; \left|M^*[l]\right| < 128, \\ 2^{i-1} \cdot 3 \cdot L & \text{otherwise.} \end{cases}$$

$$\Delta_{C[i]} = 2^i \cdot L.$$

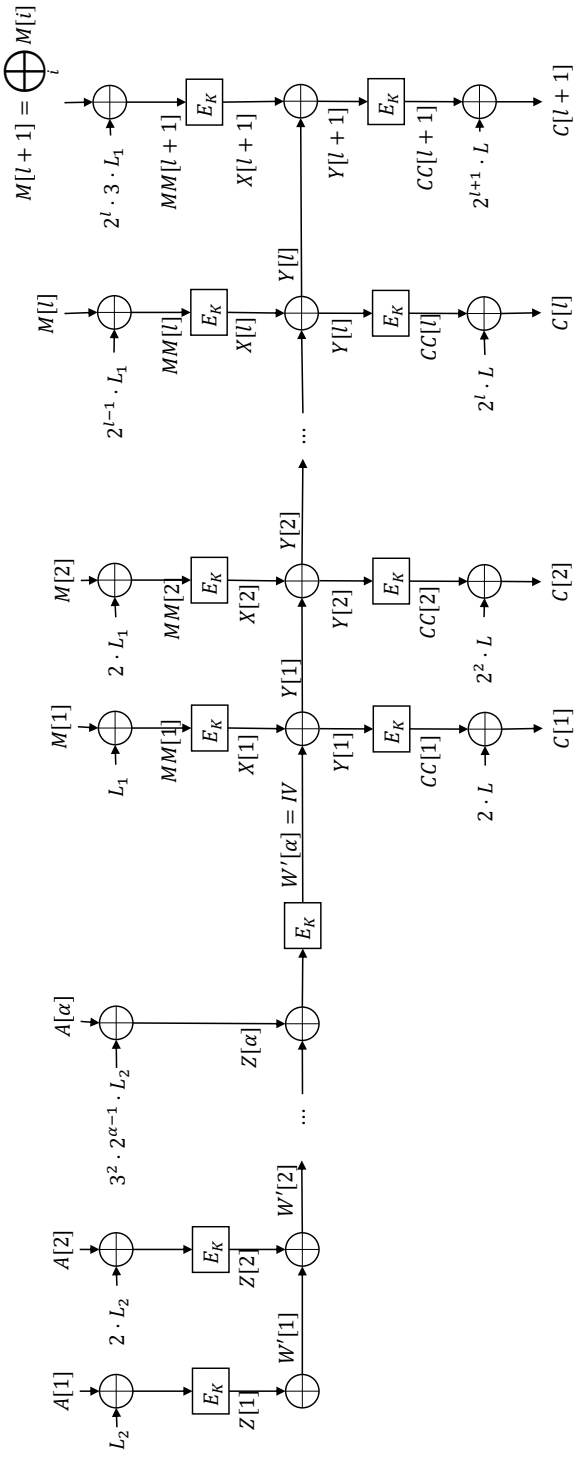

Figure 2.1 Encryption Diagram of AES-COPA

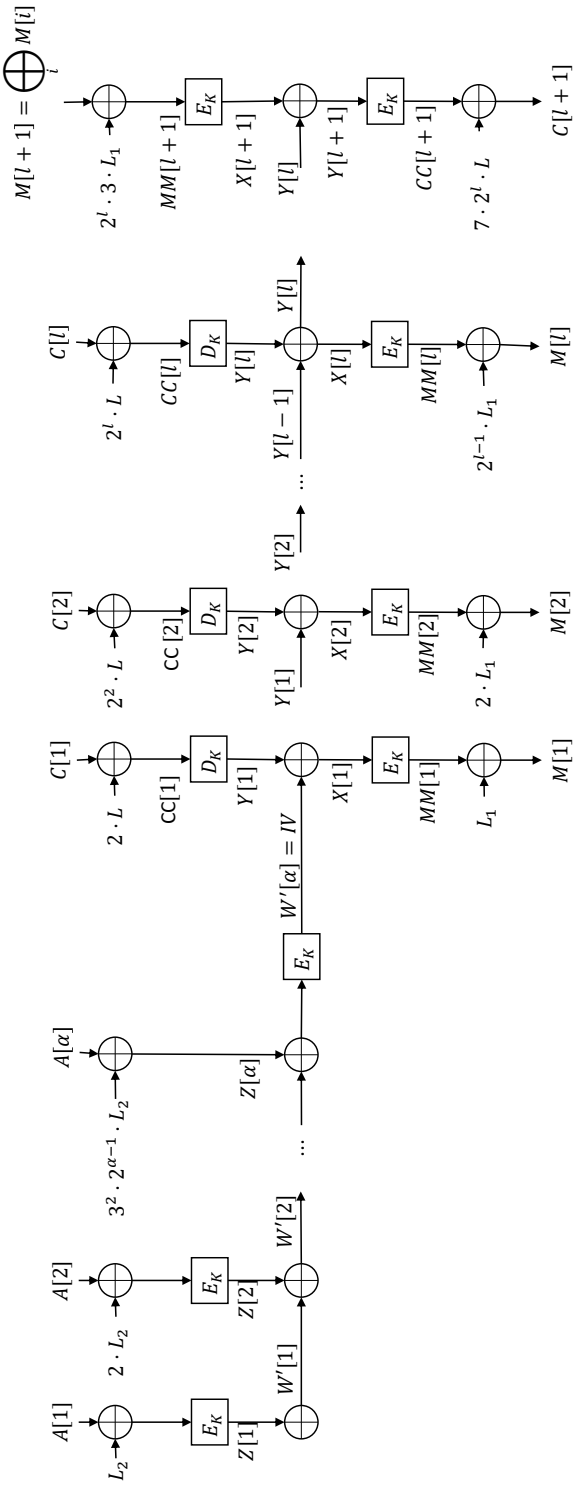

Figure 2.2 Decryption Diagram of AES-COPA

Also, two different masks (Δ_0 for the M-block, Δ_1 for the T-block) are generated for the tag generation process as follows:

$$\Delta_0 = 3^2 \cdot \Delta_{M[l]}.$$
$$\Delta_1 = 7 \cdot \Delta_{C[l]}.$$

After whitening, the input is ready for symmetric encryption. As part of the PMAC1 process, the AD-blocks are encrypted and XORed with each other, with the exception of the last block. The last blocks are involved in the XOR operation itself, and after XORing, the obtained intermediate value is encrypted and results in the V-value. The PT-blocks are processed in a similar way to the AD-blocks. The PT blocks are also involved in the generation of the tag, and the CT is created from the PT blocks. Therefore, the intermediate value after the process of each PT-block is encrypted and masked to obtain a CT-block. The final step of AES-COPA is generating the tag. In this step, an additional PT-block is constructed by XORing all input blocks and is processed in a similar way as the previous blocks to construct the tag.

In the decryption process, the AD is processed in the same way as the encryption process. The necessary intermediate value required to calculate the i^{th} PT-block is obtained by decrypting the $i - 1^{st}$ and i^{th} CT-blocks and XORing the outputs, except the first block. For the first block, the V-value obtained from the process of the AD is used instead of the intermediate value obtained from the $i - 1^{st}$ CT-block. After all PT-blocks are obtained, they are processed in the same way as in tag generation to create the control of the tag. If the calculated tag matches the received tag, the PT is released.

The formulation of the processes in AES-COPA is given below. The first process is the calculation of the V-value from the associated data AD, denoted as A, and the public

message number, denoted as N:

$$\begin{aligned}
A &= A\|N \\
AA[i] &= A[i] \oplus \Delta_A[i] && \text{for } i = 1, \ldots, \alpha, \\
Z[i] &= E_K(AA[i]) && \text{for } i = 1, \ldots, \alpha - 1, \\
W'[1] &= A[1], \\
W'[i] &= W[i-1] \oplus Z[i] && \text{for } i = 2, \ldots, \alpha - 1, \\
W'[\alpha] &= W[\alpha - 1] \oplus AA[\alpha], \\
V &= E_K(W[\alpha])
\end{aligned}$$

V and PT are the inputs of the encryption process. In the encryption process, besides the CT, the S-value is generated to be used in the calculation of the tag. How to obtain the IV from the AD has been explained recently. Encryption has three phases:

- Symmetric encryption of blocks with underlying block cipher ($X[i] = E_K(MM[i])$)
- Linear mixing of intermediate blocks ($Y[i] = X[i] \oplus Y[i-1]$)
- Symmetric encryption of mixed blocks with underlying block cipher ($CC[i] = E_K(Y[i])$)

Pseudo code for whole encryption process is as follows:

$$\begin{aligned}
Y[0] &= V, \\
MM[i] &= M[i] \oplus \Delta_M[i] && \text{for } i = 1, \ldots, d, \\
X[i] &= E_K(MM[i]) && \text{for } i = 1, \ldots, d \\
Y[i] &= Y[i-1] \oplus X[i] && \text{for } i = 1, \ldots, d, \\
CC[i] &= E_K(Y[i]) && \text{for } i = 1, \ldots, d, \\
C[i] &= CC[i] \oplus \Delta_C[i] && \text{for } i = 1, \ldots, d.
\end{aligned}$$

After the encryption process, the tag generation process is initiated. How an incomplete last block is padded has already been explained. The tag generation process begins with XORing all PT-blocks to construct the last input block. After masking, the last block is encrypted and XORed with the S-value generated during the encryption process. The tag T is generated by encrypting and masking the last intermediate value. The output of the last process is 128 bits. If a shorter tag length is chosen, the tag is shortened accordingly. This process can be formulated as follows:

$$\Sigma = M[1] \oplus M[2] \oplus \cdots \oplus M[d],$$
$$T = E_K(E_K(\Sigma \oplus \Delta_0) \oplus S) \oplus \Delta_1$$

During the decryption process, V is generated in the same way as in the encryption process. Then, by using the inverse of the sub-operations in reverse order, PT is obtained. We will not explain it in detail but give the pseudocode:

$$\begin{aligned}
Y[0] &= V, \\
CC[i] &= C[i] \oplus \Delta_C[i] && \text{for } i = 1, \ldots, d, \\
Y[i] &= D_K(MM[i]) && \text{for } i = 1, \ldots, d \\
X[i] &= Y[i-1] \oplus Y[i] && \text{for } i = 1, \ldots, d, \\
MM[i] &= D_K(Y[i]) && \text{for } i = 1, \ldots, d, \\
M[i] &= MM[i] \oplus \Delta_M[i] && \text{for } i = 1, \ldots, d.
\end{aligned}$$

It should be noted that in the decryption process, although there are $d + 1$ blocks, the loop is followed until the d^{th} block. The last block is used to verify the tag and has a special operation. To verify the tag, $M[d + 1]$ is generated in the same way as in the tag generation process after the decryption process is finished. Finally, $\widehat{T}$ is generated in the same way as T is generated in the tag generation process. $\widehat{T}$ and T are compared before releasing the PT. If the control mechanism checks, the PT is released; otherwise, only $\perp$ is returned.

Pseudocode for the tag verification process is:

$$\Sigma = M[1] \oplus M[2] \oplus \cdots \oplus M[d],$$
$$\widehat{T} = E_K(E_K(\Sigma \oplus \Delta_0) \oplus S) \oplus \Delta_1$$
$$T \stackrel{?}{=} \widehat{T}.$$

2.7.2. ELmD

ELmD is designed as an authenticated encryption algorithm. Unlike COPA and COLM, ELmD is an Encrypt-Linear mix-Decrypt type block cipher. The biggest advantage of using decryption instead of encryption in the second cipher part of the algorithm is that the same mechanism can be used during both encryption and decryption queries. However, this advantage becomes a disadvantage for devices/systems that only require either authenticated encryption or decryption because both symmetric encryption and decryption structures have to be implemented regardless of system needs. For ELmD, the defined single key length is 128 bits. ELmD accepts arbitrary data up to 2^{64} bits for each of AD and PT, and outputs CT with the same length as PT, a tag of length from 128 bits to 255 bits, and optional intermediate tags if determined in the parameter list.

ELmD consists of two types of operations: linear mixing and symmetric encryption/decryption. It has the most complex linear mix among the three structures. The linear operations in ELmD are defined in $GF(2^{128})$. In the symmetric encryption layer, the blocks are encrypted with either AES-128 algorithm or reduced-round AES-128 algorithm using a symmetric key K.

Before either the encryption or decryption process starts, the algorithm produces three subkeys to be used as whitening keys. These subkeys are called L, L_1, and L_2. Their

formulas are given in the following equations, respectively:

$$L{=}E_K(0),$$
$$L_1{=}3 \cdot L,$$
$$L_2{=}3^2 \cdot L.$$

The main subkey L, the second subkey L_1, and the third subkey L_2 are used as whitening masks for PT, AD, and CT respectively. The details of the whitening process will be explained when the encryption process is described.

ELmD algorithm has three main processes:

1. IV-generation,
2. Symmetric query;
   (a) Encryption,
   (b) Decryption,
3. Authentication;
   (a) Tag generation,
   (b) Tag verification.

The authenticated encryption consists of IV-generation, symmetric encryption, symmetric decryption, and tag generation. The authenticated decryption consists of the same operations as authenticated encryption, but tag generation is replaced with tag verification. In all processes, AES-128 algorithm or round-reduced AES-128 algorithm is used as the underlying block cipher. AD, PT, CT, and the tag are selectively given as input to these processes. Since AES-128 takes only 128-bit input blocks, the input data is divided into 128-bit subblocks. If any of the last blocks of AD or PT is shorter than 128 bits, they are padded with the pattern 10^* where $^* = 127 - |T^*[l]|$ is the number needed to complete the

last block to 128 bits. Here, T denotes either A or M according to the content. The data is divided into subblocks as:

$$A{=}(A[1], \ldots, A[\alpha - 1], A^*[\alpha]),$$
$$M{=}(M[1], \ldots, M[l - 1], M^*[l]),$$
$$C{=}(C[1], \ldots, C[l - 1], C^*[l]).$$

The encryption block diagram of ELmD is shown in Figure 2.3. Since encryption and decryption only differ in the masking and linear mixing parts, the decryption block diagram of ELmD is omitted. In this figure, the E_K block and D_K block denote 128-bit (round-reduced) AES encryption and decryption operations, respectively.

Previously, we mentioned that whitening masks are used before symmetric encryption and after symmetric decryption. Another whitening mask is used for each AD, PT, and CT block. The derivation of each whitening mask is given by the equations below:

$$\Delta_{A[i]} = \begin{cases} 7 \cdot 2^{i-2} \cdot L_1 & \text{if } i = d \;\&\; \left|A^*[\alpha]\right| < 128, \\ 2^{i-1} \cdot L_1 & \text{if } i = d \;\&\; \left|A^*[d]\right| = 128, \\ 2^i \cdot L_1 & \text{otherwise.} \end{cases}$$

$$\Delta_{M[i]} = \begin{cases} 7^2 \cdot 2^{i-2} \cdot L & \text{if } i \in \{l, l+1\} \;\&\; \left|M^*[d]\right| < 128, \\ 7 \cdot 2^{i-2} \cdot L & \text{if } i \in \{l, l+1\} \;\&\; \left|M^*[d]\right| = 128, \\ 2^i \cdot L & \text{otherwise.} \end{cases}$$

$$\Delta_{C[i]} = 2^{i+it} \cdot L_2.$$

$$\Delta_{T[it]} = 2^{i+it} \cdot L_2.$$

In this equation, i denotes the count of the processed CT block, and it denotes the count of the processed intermediate tag, if there exists any.

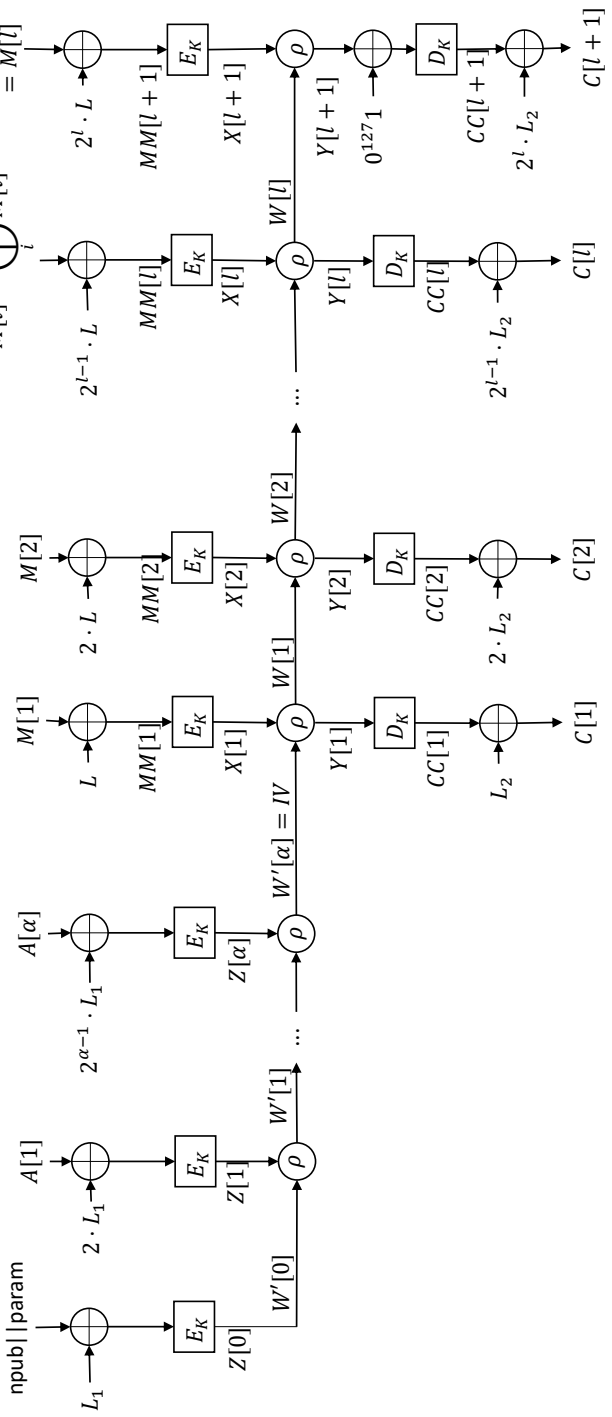

Figure 2.3 Encryption Diagram of ELmD

After whitening, the input is ready for symmetric encryption. The AD and PT blocks are encrypted and then subjected to a complicated linear operation, denoted as ρ. The ρ operation is formulated as:

$$\begin{aligned}(y, st') &= \rho(x, st),\\ y &= x \oplus 3 \cdot st,\\ st' &= x \oplus 2 \cdot st.\end{aligned}$$

The intermediate value obtained after processing the last AD block is called the IV. After the linear operations, starting from the first PT block, the resulting values are decrypted and XORed with the corresponding whitening mask (from the L_2 set).

In the authenticated decryption process, the IV is generated in the same way as in the authenticated encryption process. On the other hand, the CT blocks are encrypted after whitening, and the encrypted blocks are processed with the inverse ρ operation, denoted as ρ^{-1}. The ρ^{-1} operation is formulated as:

$$\begin{aligned}(x, st') &= \rho^{-1}(y, st),\\ x &= y \oplus 3 \cdot st,\\ st' &= y \oplus st.\end{aligned}$$

In ρ and ρ^{-1} linear operations, st denotes the current state value, and st' denotes the next state value for the i^{th} block (in Figure 2.3, $W[i]$ and $W[i+1]$, respectively).

After the linear operations, the resulting values are decrypted and XORed once more with the corresponding whitening mask (from the L set).

We mentioned that the IV is the intermediate value obtained after the last AD block is processed. The entire process is formulated as follows:

$$\begin{aligned}
A[0] &= npub \| param \\
AA[i] &= A[i] \oplus \Delta_A[i] \qquad \text{for } i = 0, \ldots, d, \\
Z[i] &= E_K(AA[i]) \qquad \text{for } i = 0, \ldots, d, \\
W'[0] &= 0, \\
W'[i] &= W[i-1] \oplus Z[i] \text{ for } i = 1, \ldots, d, \\
IV &= W'[d].
\end{aligned}$$

We consider IV and PT as the core inputs of the encryption process. The method of obtaining IV from AD was explained earlier. The encryption process consists of three phases:

- Symmetric encryption of blocks with the underlying block cipher ($X[i] = E_K(MM[i])$)
- Linear mixing of intermediate blocks ($(Y[i], W[i]) = \rho(X[i], W[i-1])$)
- Symmetric decryption of mixed blocks with the underlying block cipher ($CC[i] = E_K(X[i])$)

The pseudocode for the entire encryption process is as follows:

$$
\begin{aligned}
W[0] &= IV, \\
M[l] &= \begin{cases} \bigoplus_{i=1}^{l-1} M[i] \oplus (M^*[l] \| 10*) & if \left| M^*[l] \right| < 128, \\ \bigoplus_{i=1}^{l-1} M[i] \oplus M^*[l] & \text{otherwise} \end{cases} \\
M[l+1] &= M[l] \\
MM[i] &= M[i] \oplus \Delta_M[i] && \text{for } i = 1, \ldots, l+1, \\
X[i] &= E_K(MM[i]) && \text{for } i = 1, \ldots, l+1 \\
(Y[i], W[i]) &= \rho(X[i], W[i-1]) && \text{for } i = 1, \ldots, l+1, \\
CC[i] &= D_K(Y[i]) && \text{for } i = 1, \ldots, l, \\
C[i] &= CC[i] \oplus \Delta_C[i] && \text{for } i = 1, \ldots, l. \\
TT[it] &= D_K(W[i]) && \text{for } it = 1, \ldots, t, \\
T[it] &= TT[i] \oplus \Delta_T[it] && \text{for } it = 1, \ldots, t. \\
CC[l+1] &= D_K(Y[l+1] \oplus 1) \\
C[l+1] &= CC[l+1] \oplus \Delta_C[i].
\end{aligned}
$$

In the encryption process, the last block of the given plaintext undergoes a different procedure compared to other blocks. We have already explained how an incomplete block (shorter than 128 bits) is padded. Subsequently, all input blocks are XORed with each other to form the last input block. The newly generated block is treated as the l^{th} and $l + 1^{st}$ blocks in the algorithm. Once the encryption process is completed, all CC and TT blocks are XORed with their respective whitening masks (Δ_C and Δ_T, respectively), resulting in the generation of CT and intermediate tags. Consequently, the generated CT will be 128 bits to 255 bits longer than PT if there are no intermediate tags, and an additional 128 bits longer for each intermediate tag present. The extra bits are considered as the tag and are used to authenticate the message in the decryption process.

During the decryption process, IV is generated in the same manner as in the encryption process. Then, by employing the same encryption and decryption operations, along with the

inverse of the linear operation in the same order, the PT is obtained. Since this process is analogous to the encryption process, we will not provide a detailed explanation but instead present the pseudocode:

$$
\begin{aligned}
W[0] &= IV, \\
CC[i] &= C[i] \oplus \Delta_C[i] && \text{for } i = 1, \ldots, l, \\
Y[i] &= E_K(CC[i]) && \text{for } i = 1, \ldots, l, \\
(X[i], W[i]) &= \rho^{-1}(Y[i], W[i-1]) && \text{for } i = 1, \ldots, l, \\
MM[i] &= D_K(X[i]) && \text{for } i = 1, \ldots, l, \\
M[i] &= MM[i] \oplus \Delta_M[i] && \text{for } i = 1, \ldots, l, \\
M^*[l] &= \bigoplus_{i=1}^{l} \oplus M[i], \\
M[l+1] &= M^*[l]
\end{aligned}
$$

It is important to note that in the decryption process, even though there are $l + t + 1$ blocks, the loop is executed for the l blocks. The remaining block is used for tag verification and involves a special operation. To verify the tag, $M[l]$ is assigned to $M[l + 1]$ and encrypted to obtain the corresponding CT block ($C'[l+1]$). The bits of $C[l+1]$ block that are available are compared with the corresponding bits in $C[l + 1]$. Additionally, the padding pattern of $M^*[l]$ is checked to ensure it matches the expected padding pattern. If both of these verification mechanisms pass, the PT is released. Otherwise, only $\perp$ is returned. The pseudocode for

the tag verification process is as follows:

$$
\begin{aligned}
TT[it] &= T[it] \oplus \Delta_T[it] && \text{for } it = 1, \ldots, t, \\
W'[i+it] &= E_K(TT[it]) && \text{for } it = 1, \ldots, t, \\
W[i+it] &\stackrel{?}{=} W'[i+it], \\
M[l+1] &= M[l], \\
MM[l+1] &= M[l+1] \oplus \Delta_M[l+1], \\
X[l+1] &= E_K(MM[l+1]), \\
(Y[l+1], W[l+1]) &= \rho(X[l+1], W[l]), \\
CC[l+1] &= E_K(Y[l+1]), \\
C'[l+1] &= CC[l] \oplus \Delta_C[l+1], \\
C[l+1] &\stackrel{?}{=} C'[l+1].
\end{aligned}
$$

2.7.3. COLM

COLM is an ELmE (Encrypt-Linear mix-Encrypt) type authenticated encryption block cipher. It is a combination and modification of the COPA [10] and ELmD [9] algorithms preceding COLM.

During the encryption process, COLM takes a 128-bit key, arbitrary length AD and PT as inputs, and produces CT, along with a tag as output and optional intermediate tags if specified in the parameters.

COLM utilizes two types of operations: linear mixing and symmetric encryption. In the linear mix layer, inputs are multiplied and then XORed with each other. These multiplications are defined in the $GF(2^{128})$. In the symmetric encryption layer, AES-128 algorithm is used to encrypt the blocks with a symmetric key K.

Before the encryption or decryption process begins, the algorithm generates three subkeys, referred to as L, L_1, and L_2, which serve as whitening masks. Their formulas are given as

follows:

$$L{=}E_K(0),$$
$$L_1{=}3 \cdot L,$$
$$L_2{=}3^2 \cdot L.$$

The main subkey L, the second subkey L_1, and the third subkey L_2 are used as whitening masks for PT, AD, and CT respectively. The details of the whitening process will be explained when describing the encryption process.

COLM algorithm consists of three main processes: encryption, decryption, and tag validation. Both encryption and decryption involve a common preprocessing step called IV-generation. AES-128 algorithm is used as the underlying block cipher for all four processes. AD, PT, CT, and the tag are selectively given as input to these processes. Since AES-128 operates on 128-bit blocks, the input data is divided into subblocks of 128 bits. If the last blocks of AD or PT are shorter than 128 bits, they are padded with the pattern 10^*, where $^* = 127 - |T^*[l]|$. Here, T refers to either A or M depending on the content. The data is divided into subblocks as follows:

$$A{=}(A[1], \ldots, A[\alpha - 1], A^*[\alpha]),$$
$$M{=}(M[1], \ldots, M[l - 1], M^*[l]),$$
$$C{=}(C[1], \ldots, C[l - 1], C^*[l]).$$

The block diagrams for encryption and decryption of the COLM algorithm are depicted in Figures 2.4 and 2.5 respectively. In these figures, the E_K block represents AES-128 encryption operation, and the D_K block represents AES-128 decryption operation.

As mentioned earlier, during encryption and decryption processes, the AD, PT, and CT blocks are XORed with their corresponding whitening masks. Additionally, each data block is XORed with an individual whitening mask. The formulas for deriving these whitening

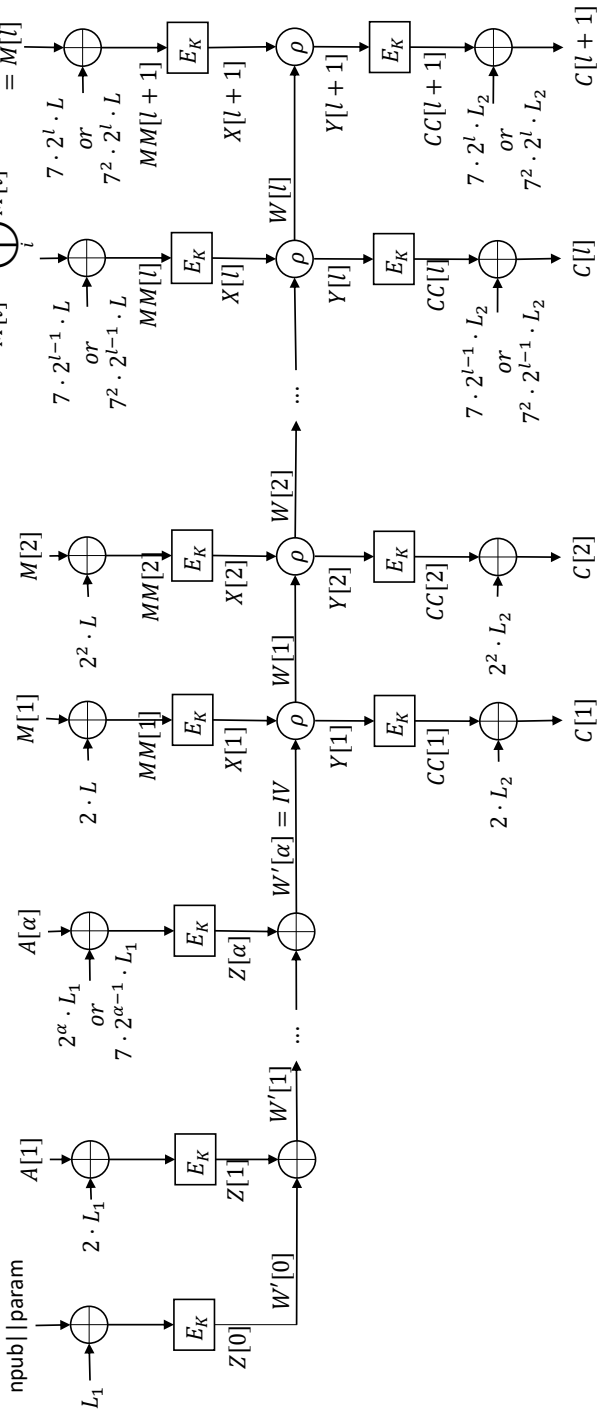

Figure 2.4 Encryption Diagram of COLM

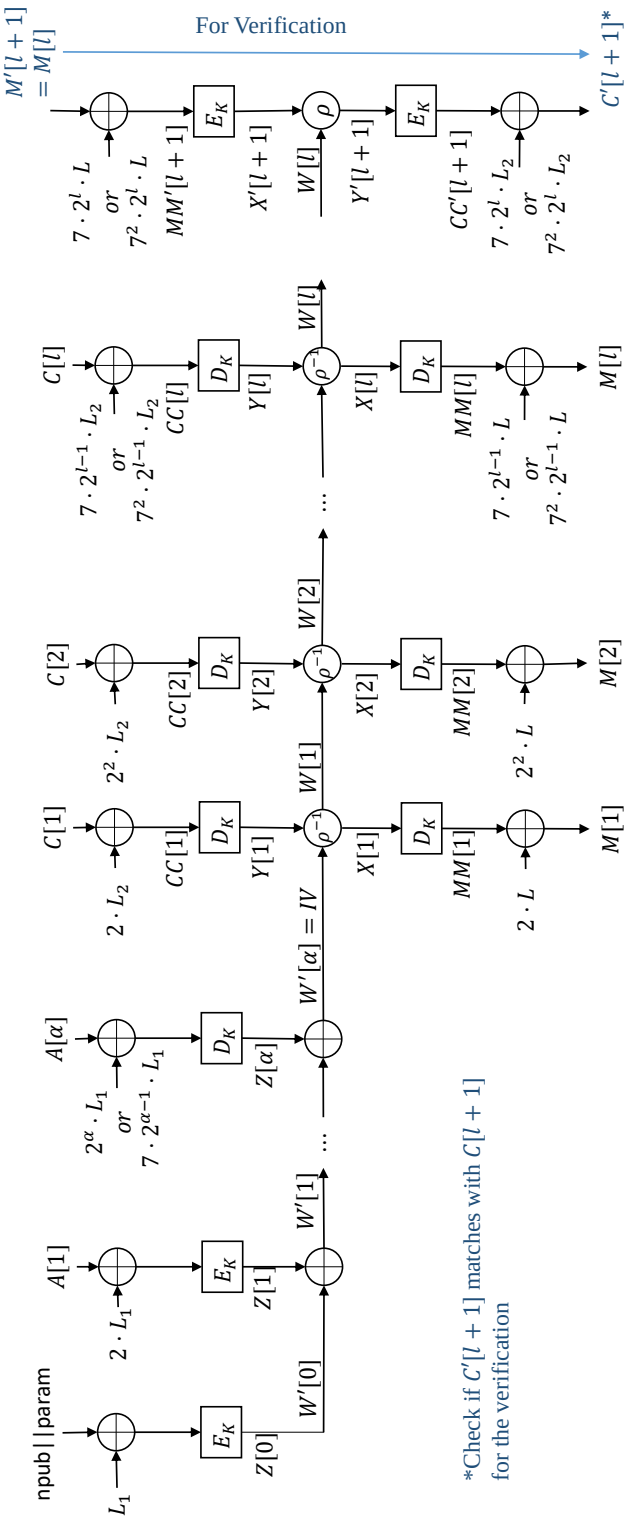

Figure 2.5 Decryption Diagram of COLM

masks are provided in the following equations:

$$\Delta_{A[i]} = \begin{cases} 7 \cdot 2^{i-1} \cdot L_1 & \text{if } i = \alpha \ \& \left|A^*[\alpha]\right| < 128, \\ 2^i \cdot L_1 & \text{otherwise.} \end{cases}$$

$$\Delta_{M[i]} = \begin{cases} 7 \cdot 2^{i-1} \cdot L & \text{if } i \in \{l, l+1\} \ \& \left|M^*[l]\right| = 128, \\ 7^2 \cdot 2^{i-1} \cdot L & \text{if } i \in \{l, l+1\} \ \& \left|M^*[l]\right| < 128, \\ 2^i \cdot L & \text{otherwise.} \end{cases}$$

$$\Delta_{C[i]} = \begin{cases} 7 \cdot 2^{i-1} \cdot L_2 & \text{if } i \in \{l, l+1\} \ \& \left|M^*[l]\right| = 128, \\ 7^2 \cdot 2^{i-1} \cdot L_2 & \text{if } i \in \{l, l+1\} \ \& \left|M^*[l]\right| < 128, \\ 2^i \cdot L_2 & \text{otherwise.} \end{cases}$$

After the whitening process, the input is prepared for symmetric encryption. The AD blocks are encrypted and then XORed with each other to generate the initialization vector (IV). The PT blocks are encrypted and subjected to a more complex linear operation called ρ. The ρ operation is defined as follows:

$$\begin{aligned} (y, st') &= \rho(x, st), \\ y &= x \oplus 3 \cdot st, \\ st' &= x \oplus 2 \cdot st. \end{aligned}$$

After the linear operations, the resulting values are encrypted and XORed once again with their corresponding whitening mask from the L_2 set.

The CT blocks are decrypted after the whitening process, and the decrypted blocks undergo the inverse ρ operation, denoted as ρ^{-1}. The ρ^{-1} operation is defined as follows:

$$\begin{aligned} (x, st') &= \rho^{-1}(y, st), \\ x &= y \oplus 3 \cdot st, \\ st' &= y \oplus st. \end{aligned}$$

In both the ρ and ρ^{-1} linear operations, st represents the current state value, and st' represents the next state value for the i^{th} block (denoted as $W[i]$ and $W[i+1]$ in Figures 2.4 and 2.5, respectively).

After the linear operations, the resulting values are decrypted and XORed once again with their corresponding whitening mask from the L set.

As previously mentioned, the AD blocks are used to generate the IV, and the entire process can be formulated as follows:

$$\begin{aligned}
A[0] &= npub \| param \\
AA[i] &= A[i] \oplus \Delta_A[i] \qquad \text{for } i = 1, \ldots, \alpha, \\
Z[i] &= E_K(AA[i]) \qquad \text{for } i = 1, \ldots, \alpha, \\
W'[0] &= E_K(AA[0]), \\
W'[i] &= W[i-1] \oplus Z[i] \text{ for } i = 1, \ldots, \alpha, \\
IV &= W'[\alpha].
\end{aligned}$$

We consider IV and PT as the core inputs of the encryption process. The method for obtaining IV from AD has been recently explained. The encryption process consists of three phases:

- Symmetric encryption of blocks with the underlying block cipher ($X[i] = E_K(MM[i])$)
- Linear mixing of intermediate blocks ($(Y[i], W[i]) = \rho(X[i], W[i-1])$)
- Symmetric encryption of mixed blocks with the underlying block cipher ($CC[i] = E_K(X[i])$)

The pseudocode for the entire encryption process is as follows:

$$
\begin{aligned}
W[0] &= IV, \\
MM[i] &= M[i] \oplus \Delta_M[i] && \text{for } i = 1, \ldots, l+1, \\
X[i] &= E_K(MM[i]) && \text{for } i = 1, \ldots, l+1 \\
(Y[i], W[i]) &= \rho(X[i], W[i-1]) && \text{for } i = 1, \ldots, l+1, \\
CC[i] &= E_K(Y[i]) && \text{for } i = 1, \ldots, l+1, \\
C[i] &= CC[i-1] \oplus \Delta_C[i] && \text{for } i = 1, \ldots, l+1.
\end{aligned}
$$

In the encryption process, the last block of the given plaintext has a different procedure compared to the other blocks. We have already explained how an incomplete block (shorter than 128 bits) is padded. Afterward, all input blocks are XORed with each other to form the last input block. The generated new block is given to the algorithm as the l^{th} and $l + 1^{st}$ blocks. After the encryption process is completed, all CC blocks are XORed with their corresponding whitening mask (L_2), and the ciphertext CT is generated. However, if the PT was padded, bits are removed from CT with a length equal to the padding bits. As a result, the generated CT will be exactly 128 bits longer than PT. These extra bits are considered as the tag and are used to authenticate the message in the decryption process.

During the decryption process, IV is generated in the same way as in the encryption process. Then, by using the inverse of the sub-operations in reverse order, the plaintext PT is

obtained. We do not explain it in detail here but provide the pseudocode:

$$
\begin{aligned}
W[0] &= IV, \\
CC[i] &= C[i] \oplus \Delta_C[i] && \text{for } i = 1, \ldots, l, \\
Y[i] &= E_K^{-1}(CC[i]) && \text{for } i = 1, \ldots, l, \\
(X[i], W[i]) &= \rho^{-1}(Y[i], W[i-1]) && \text{for } i = 1, \ldots, l, \\
MM[i] &= E_K^{-1}(X[i]) && \text{for } i = 1, \ldots, l, \\
M[i] &= MM[i] \oplus \Delta_M[i] && \text{for } i = 1, \ldots, l, \\
M^*[l] &= M[1] \oplus \cdots \oplus M[l].
\end{aligned}
$$

It should be noted that in the decryption process, even though there are $l+1$ blocks, the loop is followed only until the l^{th}-block. The last block is used for tag verification and undergoes a special operation. To verify the tag, $M[l]$ is assigned to $M[l+1]$ and encrypted to obtain a corresponding CT block ($C'[l+1]$). The available bits of the $C[l+1]$ block are compared with the bits that correspond to ones in $C[l+1]$. Additionally, the $M^*[l]$ is checked to confirm whether it has the expected padding pattern. If both of these control mechanisms pass, the plaintext PT is released; otherwise, only $\perp$ is returned. The pseudocode for the tag verification process is as follows:

$$
\begin{aligned}
M[l+1] &= M[l], \\
MM[l+1] &= M[l+1] \oplus \Delta_M[l+1], \\
X[l+1] &= E_K(MM[l+1]), \\
(Y[l+1], W[l+1]) &= \rho(X[l+1], W[l]), \\
CC[l+1] &= E_K(Y[l+1]), \\
C'[l+1] &= CC[l] \oplus \Delta_C[l+1], \\
C[l+1] &\stackrel{?}{=} C'[l+1].
\end{aligned}
$$

3. RELATED WORK

Providing authentication and encryption simultaneously has been under focus of the literature for decades. Various studies ([15–18]) have been published to consider them together. However, these solutions may not be able to fully meet the true demand. In real-world applications, in addition to secret data, whose integrity and confidentiality are both important, there also exists informative data, such as headers, that have to be transmitted in clear while maintaining their integrity. In 2002, Rogaway published his fundamental work ([4]) by modeling data as a compound of two parts: AD and PT. Rogaway introduced a new structure to guarantee the integrity of both AD and PT simultaneously while securing the confidentiality of only the PT. After formulating and naming the AEAD problem, he proposed an efficient solution in general and for the specific case of the authenticated-encryption scheme OCB. In the general setting, he assessed *nonce stealing* and *ciphertext translation* methods to make an authenticated-encryption scheme support associated data. For the case of OCB, he successfully combined OCB and the pseudorandom function PMAC and proved that the combination remains sound even when the same key is used in both schemes. His contributions significantly advanced the state-of-the-art in AEAD and served as the basis for subsequent research in this area.

Following [4], attention to the AEAD topic has increased, and numerous works related to AEAD have been published. [28–31] are examples of new AEAD designs. The goal of these designs is to increase security and/or performance without compromising the other. In parallel with new designs, analyses have also been conducted, such as [32–35]. While analyzing the developed designs and identifying weaknesses and vulnerabilities in the schemes, these works have also suggested methods to improve security in terms of both authentication and confidentiality where possible. With the start of the CAESAR competition, the focus on AEAD schemes has increased, and many more works (e.g., [11, 12, 36–45]) have been published to evaluate the security of AEAD schemes. Alongside security considerations, studies such as [1, 46–50] have been published to propose efficient and robust implementations of CAESAR candidates.

Since the beginning of the CAESAR competition, COLM and its ancestors have been under the focus of AEAD research groups. Numerous studies have been published on COLM and its predecessors. In these studies, not only the security levels of these AEAD schemes are evaluated, but also strategies to implement them efficiently on various platforms are developed. A milestone among the analysis studies is Lu's work [36], where an attack to recover $L = E_K(0)$ is mounted against COPA and Marble algorithms. His method is widely adopted against ciphers that use the sub-key L for masking the PT and/or CT blocks. The method was adapted to ELmD and COLM by Bay et al. in [11] and by Forler in [37], respectively. Based on this technique, we assume that the L parameter of COLM has already been discovered. Lu's method is explained in Section 3.1.

3.1. Almost Universal Forgery Attacks against COPA

Lu analyzed the security of the CAESAR submissions Marble and COPA authenticated encryption algorithms and mounted attacks against them in [36]. The attacks mounted by Lu are beyond birthday attack complexity. Lu conducted both forgery attacks and tag guessing attacks with the same complexity. The computational complexity level of a forgery attack is expected to be below the boundary of a birthday attack. Since Lu's attacks are slightly above the security level of forgery attacks, they are referred to as almost universal forgery attacks. Additionally, a cipher should withstand tag guessing attacks with computational complexity less than 128 bits, and Lu demonstrates that tag guessing attacks can be mounted against AES-COPA with the same level of complexity as the forgery attacks. Lu's attack can be divided into two parts: the recovery of the subkey L in the first part, and mounting the universal forgery attack with a probability of 1 in the second part.

Lu introduces two different methods for the first part, recovering the subkey L part, of his attack;

1. variable associated data,
2. constant associated data.

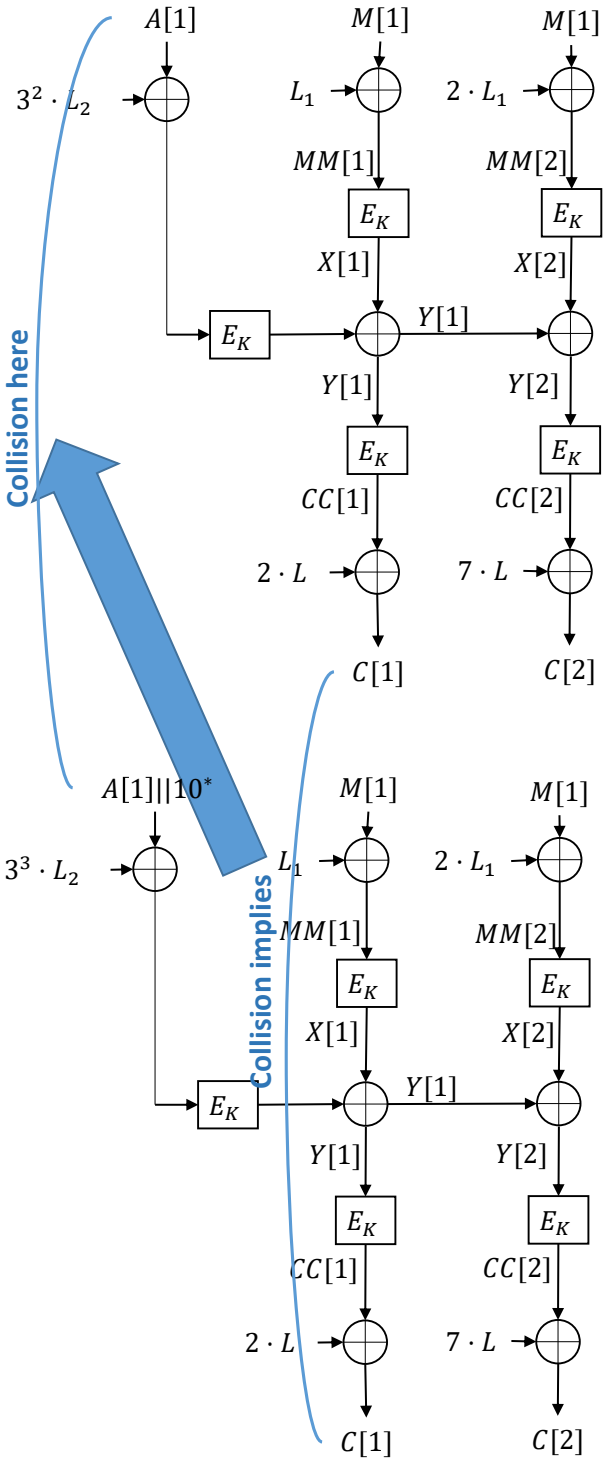

Figure 3.1 Recovering L-parameter under Variable Associated Data Setting

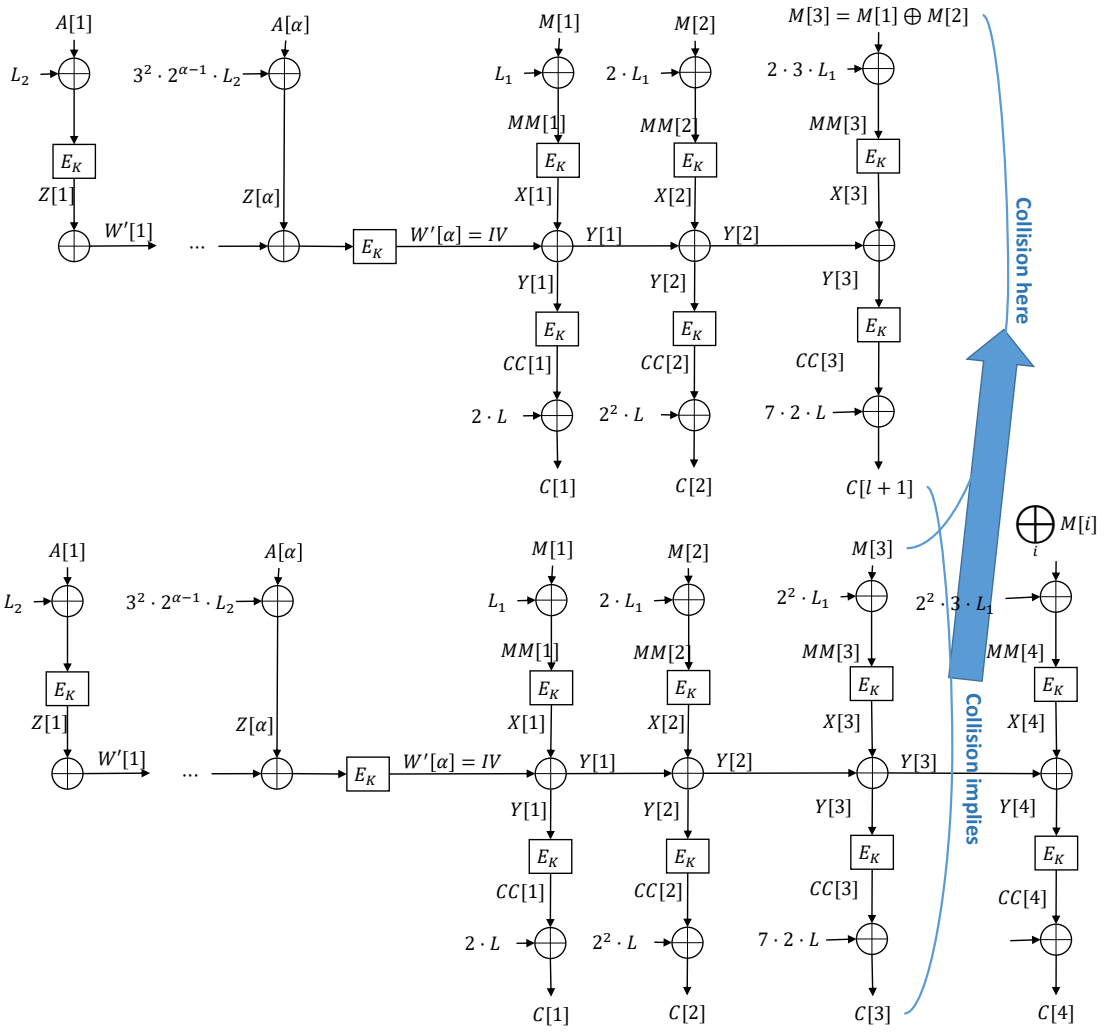

Figure 3.2 Recovering L-parameter under Constant Associated Data Setting

For the second part, mounting universal forgery part, of his attack, he presents three different methods;

1. modifying associated data,
2. modifying message,
3. modifying both associated data and message.

3.1.1. Recovering *L*-parameter under Variable Associated Data Setting

For this procedure, Lu prepares two sets of messages. Lu sets M_1 block to be constant for all messages in both sets. The first set consists of 2^σ messages with AD_1^i, M_1, where $0 < \sigma \leq \frac{n}{2}$ and $AD_1^i = i, \text{for} i = 0, 1, \ldots, 2^{\sigma-1}$. The second set consists of $2^{\varphi-1}$ messages with AD_1^j, M_1, where $0 < \varphi \leq \frac{n}{2}$ and $AD_1^j = j \times (2^{\frac{n}{2}}), \text{for} i = 0, 1, \ldots, 2^{\varphi-1}$. After querying the first set, AD_1^i values are stored in a table indexed by C_1^i. Then the second set is queried, and C_1^j values are checked for matches in the previously stored table. If a match is found between $\omega \in i$ and $\mu \in j$, as depicted in Figure 3.1, an equality relationship can be established between the AD-blocks AD_1^ω and AD_1^μ. The only differentiating blocks between the queries ω and μ are AD_1^ω and AD_1^μ blocks. Therefore, the L parameter can be calculated from this equality. This procedure to recover L requires $2^\sigma + 2^\varphi$ COPA encryption queries, a memory of $n \cdot 2^\sigma$ bits, and a time complexity of 2^φ memory accesses. The success probability of this operation is $\approx 1 - e^{-2^{\sigma+\varphi-n}}$.

3.1.2. Recovering the *L*-parameter under Constant Associated Data Setting

This procedure follows similar steps as in the variable associated data setting. However, in this attack, the associated data is kept constant throughout the entire process, and different messages are queried with the same associated data. The previous attack exploits the difference in the whitening masks of complete and incomplete blocks. The AD is followed by the message; therefore, it is possible to observe the effect of the incomplete block in the previous attack. However, in this attack, after the incomplete block (namely, the last block) is queried, the COPA query ends, and there is no other block that allows the observation of the effect of the incomplete block. This attack follows the procedure below:

1. 2^θ messages of 2 blocks ($M^{(i)} = M_1^{(i)} \| M_2^{(i)}$) are queried through the COPA oracle for ($i = 1, 2, \ldots, 2^\theta$). The obtained ciphertexts are stored according to $C_2^{(i)}$ blocks.

2. Step 1 is repeated until δ messages ($M^{(i_1)}$, $M^{(i_2)}$, $\ldots$, $M^{(i_\delta)}$) having ciphertext with identical second block such that

$$C_2^{(i_1)} = C_2^{(i_2)} = \cdots = C_2^{(i_\delta)}$$

are collected.

3. Two n-bit constants such that

$$\alpha \cdot (2 \cdot 3^2 \cdot L \oplus 2^2 \cdot 3 \cdot L) = \beta \cdot (2^3 \cdot L \oplus 2 \cdot 7 \cdot L)$$

4. 2^φ messages of 3 blocks ($M^{(j)} = M_1^{(j)} \| M_2^{(j)} \| M_3^{(j)}$) are queried through the COPA oracle for ($i = 1, 2, \ldots, 2^\varphi$). All $M^{(j)}$ messages are chosen such that

$$M^{(j)} = (M_1^{(i_1)} \| M_2^{(i_1)} \| M_3^{(j)}).$$

These messages are queried through COPA encryption oracle and $C_3^{(j)}$ blocks are stored. (Note that, the key, the AD and the first two blocks are messages are the same, therefore $C_1^{(j)}$ and $C_2^{(j)}$ blocks remain same as $C_1^{(i_1)}$ and $C_2^{(i_1)}$) blocks, i.e. ($C^{(j)} = (C_1^{(i_1)} \| C_2^{(i_1)} \| C_3^{(j)})$).

5. After the required message-ciphertext sets are prepared, a message-ciphertext pair with the following property is chosen (this matching is illustrated in Figure 3.2):

$$M_3^{(j)} \oplus 2^2 \cdot 3 \cdot L = \bigoplus_{l=1}^{2} M_l^{(i_t)} \oplus 2 \cdot 3^2 \cdot L;$$

$$C_3^{(j)} \oplus 2^3 \cdot L = T^{(i_t)} \oplus 2 \cdot 7 \cdot L.$$

As an efficient way, Lu suggests checking the following equality;

$$\alpha \cdot M_3^{(j)} \oplus \beta \cdot C_3^{(j)} = \alpha \cdot \bigoplus_{l=1}^{2} M_l^{(i_t)} \oplus \beta \cdot T^{(i_t)}.$$

The qualified message-ciphertext pairs are denoted as (M^{ω}, C^{ω}) where $1 \leq \omega \leq 2^{\varphi}$.

6. From the qualified pair, L can be recovered according the equation below;

$$M_3^{(\omega)} \oplus 2^2 \cdot 3 \cdot L = \bigoplus_{l=1}^{2} M_l^{(i_t)} \oplus 2 \cdot 3^2 \cdot L.$$

Lu calculates the probability of at least one δ-tuple satisfying the equation in Step 2 as

$$p_{\delta} \approx 1 - e^{\binom{2^{\theta}}{\delta} \cdot 2^{-n(\delta-1)}}$$

for $\theta \ll n$ and small δ. The probability of the recovered L being correct is calculated as follows;

$$\frac{1}{2} \cdot (1 - e^{-\delta \cdot 2^{\varphi} - n + 1})$$

When these two probabilities combined together, the success probability of the attack becomes;

$$\approx 1 - e^{\binom{2^{\theta}}{\delta} \cdot 2^{-n(\delta-1)}} \cdot \frac{1}{2} \cdot (1 - e^{-\delta \cdot 2^{\varphi} - n + 1})$$

3.1.3. Forgery Attack by Modifying *AD*

To mount the forgery attack, Lu extends the current messages by adding two new blocks. While the AD is being processed in COPA, all message blocks except the last one are encrypted and XORed to each other. The last message block is directly XORed with the resulting intermediate block. Since XOR is an involutive function, if two blocks which become equal after masking added to AD twice, the state value does not change. Lu exploits this property of COPA given that the masking subkey L is already recovered and such block can be easily calculated and mounts the attack shown in Figure 3.3 with the following steps:

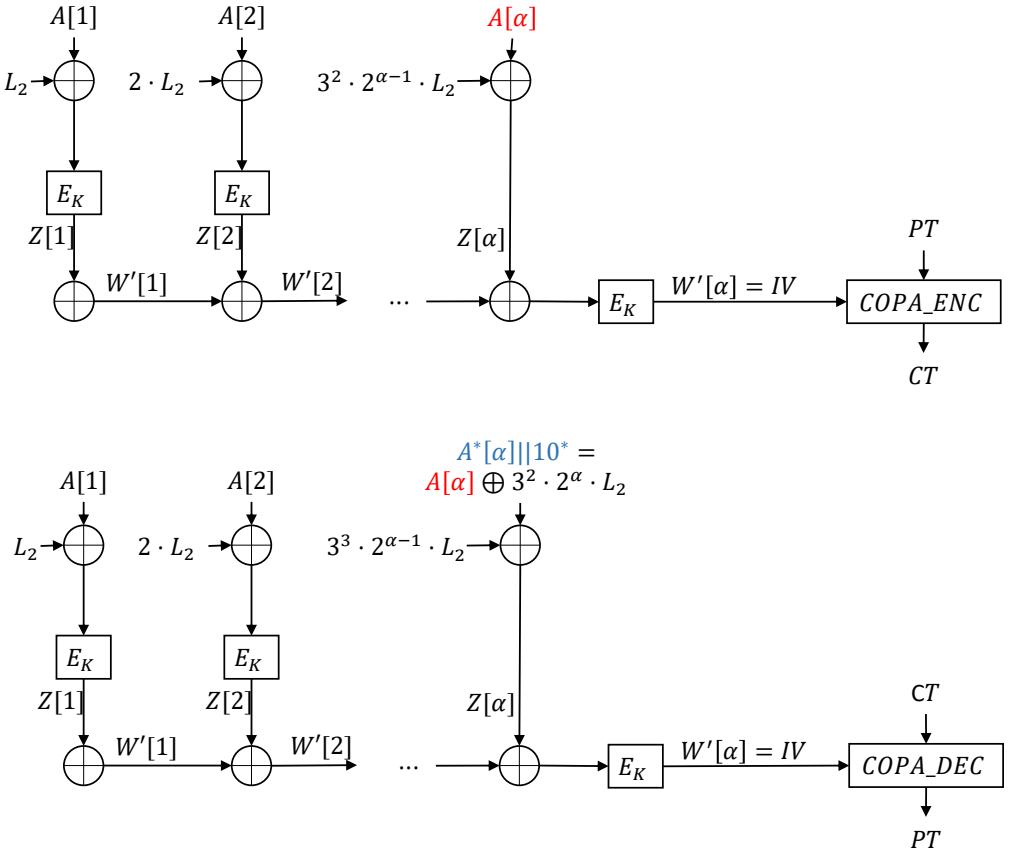

Figure 3.3 Universal Forgery with Modifying Associated Data

1. Query the COPA encryption oracle with the following $AD - PT$ pair:

$$(\widehat{AD}, M) = (AD_1, AD_2, \cdots AD_{\alpha-1}, \widehat{AD_\alpha}, \widehat{AD_\alpha} \oplus 2^{\alpha-1} \cdot 3^4 \cdot L, AD_\alpha \oplus 2^{\alpha-1} \cdot 3^6 \cdot L, M_1, M_2, \cdots, M_l).$$

2. Obtain its $CT(\widehat{C})$ and tag $(\widehat{T})$:

$$\widehat{C} = (\widehat{C}_1, \widehat{C}_2, \cdots, \widehat{C}_l),\ \widehat{T}.$$

3. Output

$$C = (\widehat{C_1}, \widehat{C_2}, \cdots, \widehat{C_l}) \text{ and } T = \widehat{T}$$

as forged ciphertext and tag.

3.1.4. Forgery Attack by Modifying *PT*

In this attack model, Lu applies an extension attack. Since the subkey L is already recovered, for a message;

$$M = M_1, M_2, \cdots, M_l$$

the additional block that will be queried to generate the tag can be calculated easily. In the forgery attack mounted by modifying the PT, Lu calculates this block and appends it to end of to the plaintext to obtain the tag as the $l + 1^{st}$ block of the tampered ciphertext. In this attack, the following steps are followed;

1. The following associated data and plaintext pair is queried;

$$(AD, \widehat{M}) = (AD_1, AD_2, \cdots, AD_\alpha, M_1, M_2, \cdots, M_l, 2^{l-1} \cdot 3 \cdot L \cdot \bigoplus_{i=1}^{l} M_i).$$

2. The related ciphertext $CT = \widehat{C}$ where

$$\widehat{C} = \widehat{C_1}, \widehat{C_2}, \cdots, \widehat{C_l}, \widehat{C_{l+1}}$$

is obtained.

3. The following ciphertext and tag pair is release as the forged ciphertext and tag for (AD, M).

$$C = (\widehat{C}_1, \widehat{C}_2, \cdots, \widehat{C}_l),\ T = \widehat{C}_{l+1} \oplus 2^{l-1} \cdot 3 \cdot L.$$

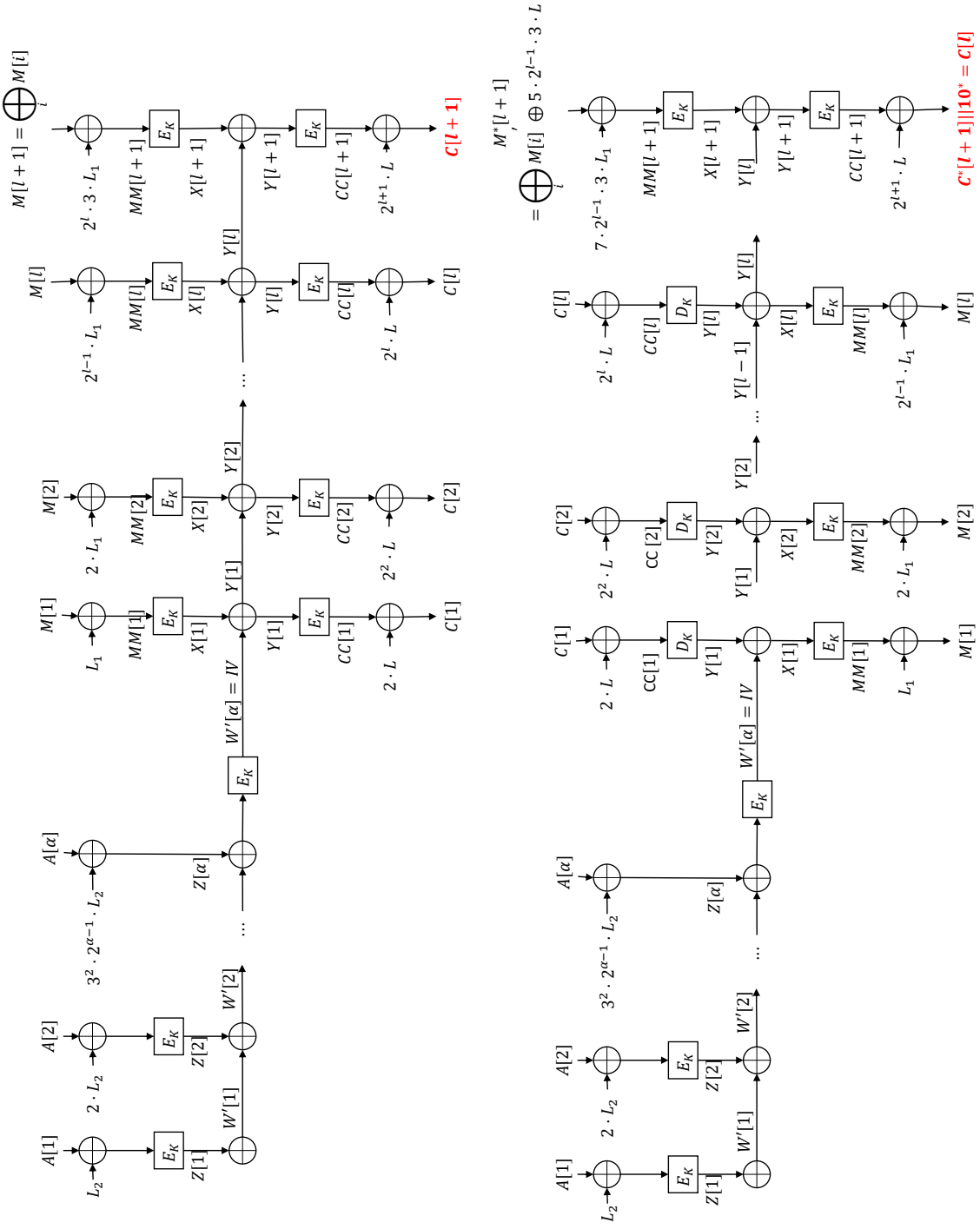

Figure 3.4 Universal Forgery with Modifying Message

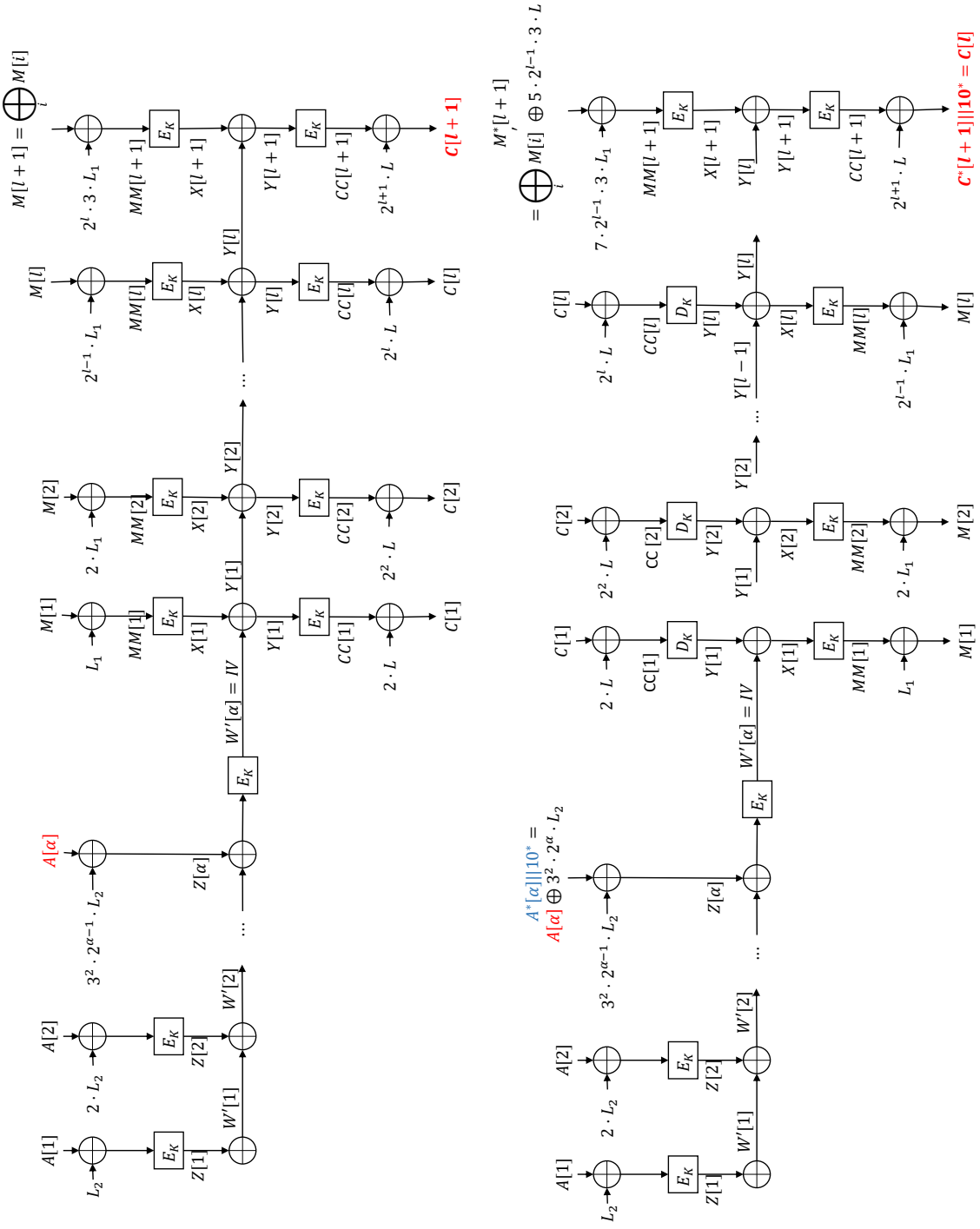

Figure 3.5 Universal Forgery with Modifying Both Associated Data and Message

3.1.5. Forgery Attack by Modifying Both *AD* and *PT*

In the last attack method, the modifications in the previous two attacks are applied at once and the attack is conducted as follows;

1. Query the COPA encryption oracle with the following $AD - PT$ pair:

$$\begin{aligned}(\widehat{AD}, M) \;\; = \;\; & (AD_1, AD_2, \cdots AD_{\alpha-1}, \widehat{AD_\alpha}, \widehat{AD_\alpha} \oplus 2^{\alpha-1} \cdot 3^4 \cdot L, \\ & AD_\alpha \oplus 2^{\alpha-1} \cdot 3^6 \cdot L, M_1, M_2, \cdots, M_l, 2^{l-1} \cdot 3 \cdot L \cdot \bigoplus_{i=1}^{l} M_i).\end{aligned}$$

2. Obtain its $CT(\widehat{C})$:

$$\widehat{C} = \widehat{C_1}, \widehat{C_2}, \cdots, \widehat{C_l}, \widehat{C_{l+1}}.$$

3. Output the following ciphertext and tag pair as the forged ciphertext and tag for (AD, M).

$$C = (\widehat{C}_1, \widehat{C}_2, \cdots, \widehat{C}_l),\ T = \widehat{C}_{l+1} \oplus 2^{l-1} \cdot 3 \cdot L.$$

3.2. Universal Forgery and SDBC Attacks against ELmD

Bay et. al mounts various forgery and key recovery attacks against ELmD cipher in [11]. The key recovery attacks are based on simulation of the decryption query of the underlying cipher. COPA and ELmD ciphers use almost the same structures; therefore, the very same methods are also valid to recover L-parameter. Bay et. al recover the subkey L under variable AD setting. Then, they mount two different universal forgery attack against ELmD. In both attacks, they modify the AD. Besides forgery attacks, they exploit the structure of ELmD to generate multiplicative and differential $PT - CT$ pairs. Subsequently, they use these pairs to build a decryption simulation model for the underlying cipher of ELmD and suggest a chosen ciphertext attack against the underlying cipher choice for AES^6.

3.2.1. Recovering the Subkey L

To recover subkey L, Bay et al. prepares two set of messages where $(D, M) = (D_1, M_1) = (\alpha, M)$ and $(D', M') = (D'_1, M'_1) = (\beta, M)$ where $0 \leq \alpha, \beta \leq 2^{64} - 1$, α is a 64-bit value and β is 128-bit value. Here, the authors exploit additions of different whitening masks for complete and incomplete blocks. The sets for α and β are chosen such that $\exists \alpha, \beta$ such that $\alpha \oplus \beta = \Xi$ for any chosen $\Xi \in \mathbb{F}_{2^{128}}$.

They query the message sets and search for a collision in the first ciphertext blocks $C_1 = C'_1$. The collision in the first block means, the constructed ADs produce the same IV-value, i.e. $IV = IV'$ since $M_1 = M'_1$. The subkey L can be recovered easily by solving the following equality;

$$\begin{aligned} D_1 \oplus 7 \cdot 3 \cdot L &= D'_1 \oplus 2 \cdot 3 \cdot L \\ L &= (5 \cdot 3)^{-1}(\alpha \| 10^{63} \oplus \beta). \end{aligned}$$

Since L is already known, rest of the attacks are explained on the intermediate blocks after whitening.

3.2.2. Universal Forgery Attacks

In their work, Bay et al. introduce two different universal forgery attacks. In the first attack, they differ the last block of the AD from complete to incomplete or vice versa. In the second attack, they prepend the AD with two additional blocks. The first attack for the $AD - PT$ pair $D_0, \cdots, D_d, M_1, \cdots, M_l$ with $|D_d| = 128$ is mounted as follows;

1. Compute
$$D'_d \| 10* = D_d.$$
2. Query
$$D_0, \cdots, D'_d, M_1, \cdots, M_l.$$

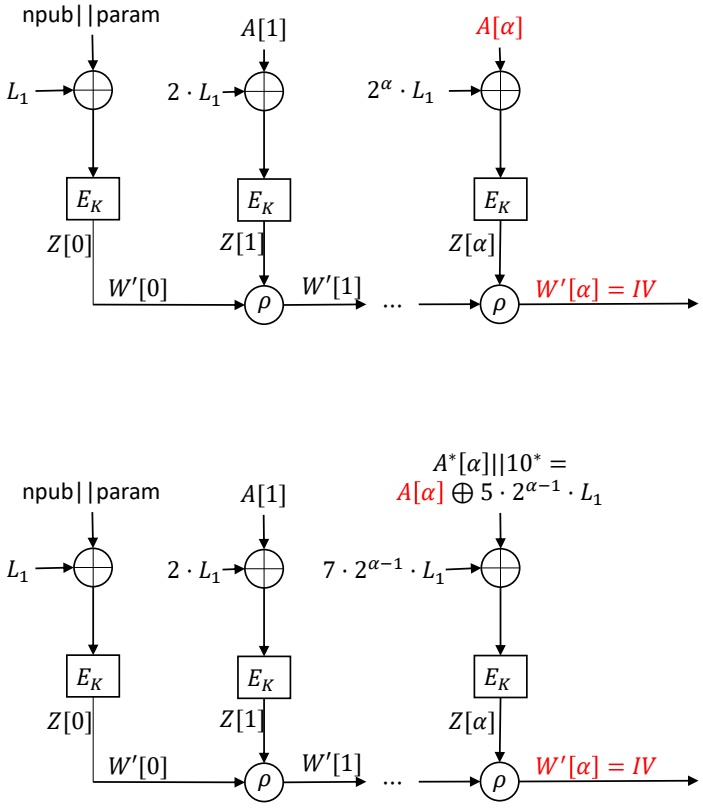

Figure 3.6 1^{st} Universal Forgery Suggested by Bay et al.

3. Output $(\widehat{C}, \widehat{T})$ as the forged CT and tag pair.

The queried associated data produces the same IV as the AD of the target $AD - PT$ pair. The associated data is equal till the $d - 1^{th}$ block; therefore, the intermediate value after the $d - 1^{th}$ block remains the same. The other component of the ρ-operations are also equal to each other because the last blocks are chosen such that after masking, they end up to be equal. The equal blocks produce the same value after encryption and following that the same IV-value. This attack is illustrated in Figure 3.6.

The same attack can be conducted for any incomplete D_d-block. In this version, D'_d block is chosen to be complete block as;

$$D'_d = D_d.$$

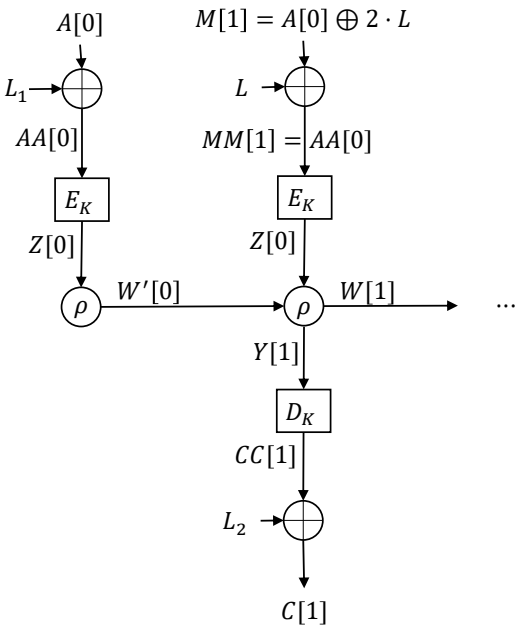

Figure 3.7 1^{st} Query of the 2^{nd} Universal Forgery Suggested by Bay et al.

The first forgery attack is conducted in a single query. The second attack requires two queries. In the first query (See Figure 3.7), a helper block is prepared to reset the intermediate state in the AD to 0^{128}. In the second query (See Figure 3.7), 1^{st} AD-block of the 1^{st} query and the helper block are prepended to AD and the new AD is generated. The steps to mount the attacks are;

1. Query $D_0, M_1 = D_0$.
2. Obtain C_1. (Note that: $C_1 = D_K(2 \cdot E_K(D_0))$
3. Query

$$\begin{aligned} AD' &= D_0 \| C_1 \| AD, \\ M' &= M. \end{aligned}$$

4. Output C' and T' as the forged CT and tag pair.

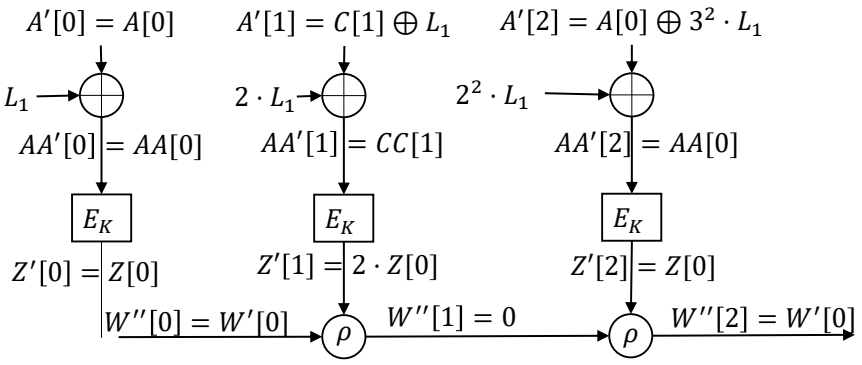

Figure 3.8 2^{nd} Query of the 2^{nd} Universal Forgery Suggested by Bay et al.

In the second query, $W_1' = E_K(D_0)$ and the second AD-block is encrypted to be $AA_2' = 2 \cdot E_K(D_0)$. Consequently;

$$\rho(W_1', AA_2') = 2 \cdot W_1' \oplus AA_2' = 0.$$

3.2.3. Generating 2-Multiplicative Pairs (R_1, R_2) such that $2 \cdot E_K(R_1) = E_K(R_2)$

This attack is theoretically same as the second universal forgery attack. However, the attack is performed using message blocks because the IV is not freely chosen. The first query of the second universal forgery is made to the oracle as is. To obtain R_2 for the message R_1, the following query is made;

$$AD = D_0 \| C_1 \quad (1)$$

$$M = R_1 \| R_1. \quad (2)$$

C_2 is output as $R_2 = D_K(2 \cdot E_K(R_1))$. This query is illustrated in Figure 3.9.

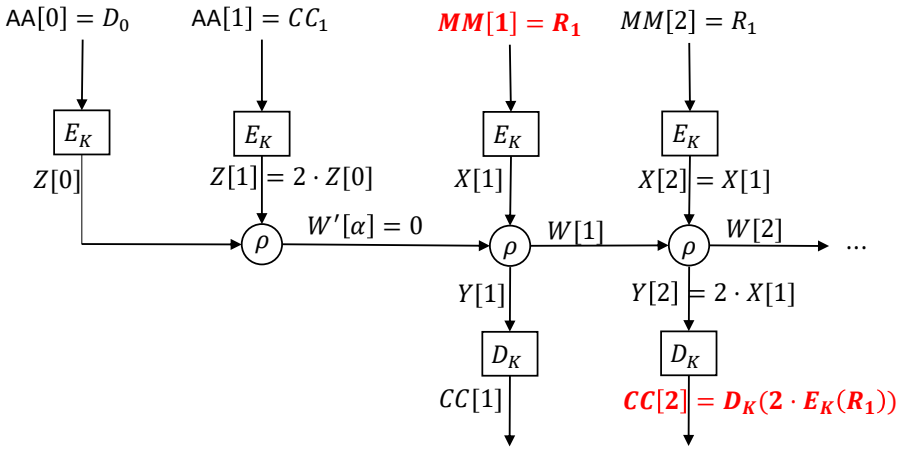

Figure 3.9 2^{nd} Query of Generating 2-Multiplicative Pairs by Bay et al.

3.2.4. Generating μ-Multiplicative Pairs (P_1, P_2) such that $\mu \cdot E_K(P_1) = E_K(P_2)$

In this attack multiplicative pairs are generated in the most general form by using the pairs constructed in Section 3.2.3. A $P_1 - P_2$ pair is found such that $\mu \cdot E_K(P_1) = E_K(P_2)$. Then 129-block PT, composed of P_1 and P_2 blocks, is prepared as shown in Figure 3.10. The intermediate text in the last block becomes;

$$Y_{129} = \mu E(P_1),$$

where $\mu = (3\mu' \oplus 1)$. Here, $\mu' = 3^{-1}(\mu \oplus 1)$ is the decision parameter. According to μ' PT blocks are chosen such that W_{128} becomes $\mu' E(R_1)$. The last block is chosen to be P_1 so $Y_{129} = \mu E(P_1)$ and $CC_{129} = D_K(\mu \cdot E_K(P_1))$ are constructed.

3.2.5. Generating 1-Difference Pairs (S_1, S_2) such that $E_K(S_1) = E_K(S_2) \oplus 1$

To prevent length extension attacks in ELmD, the designers XOR the last block with 1 after the linear-mix function and use this block to generate the tag. However, this operation ends up in a flaw in the algorithm and Bay et al. exploit it to generate 1-difference pairs. How IV can be set to 0^{128} is explained in Section 3.2.3. The operations before the linear mix layers of the AD and PT are the same; therefore, with the same method used in AD the 2^{nd} state

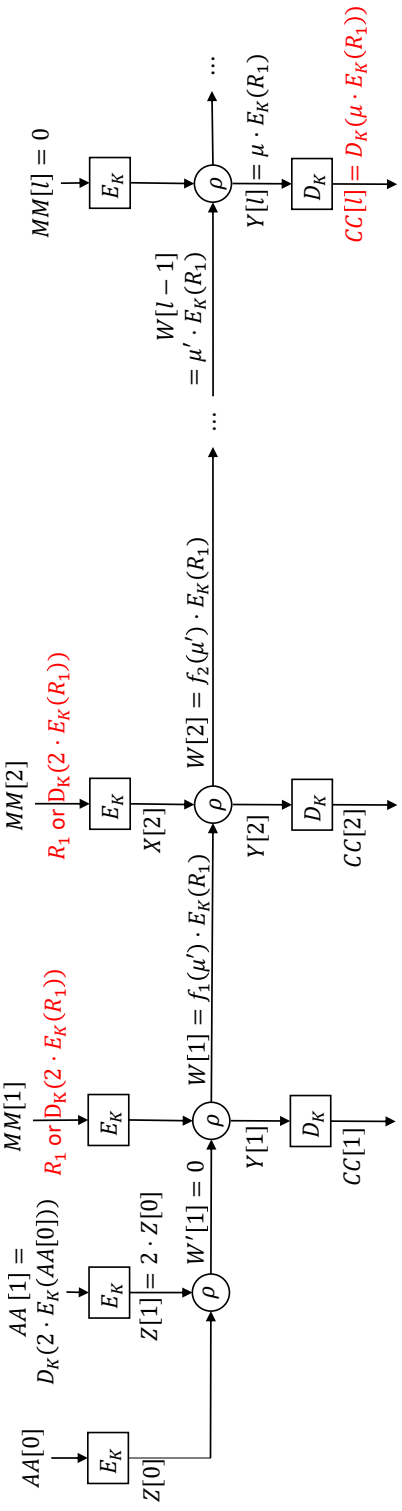

Figure 3.10 Generating μ-Difference Pairs by Bay et al.

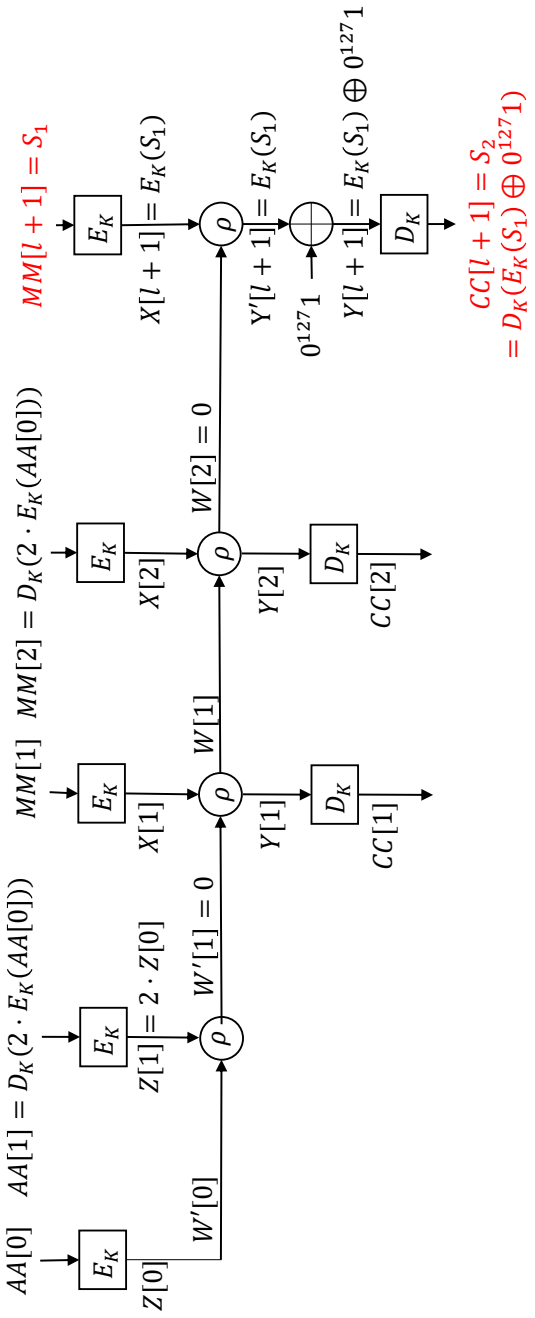

Figure 3.11 Generating 1-Difference Pairs by Bay et al.

can be set to 0^{128}. For a chosen S_1 value as the 3^{rd} PT-block, $S_2 = D_K(E_K(S_1) \oplus 1)$ can be obtained from the ELmD oracle as the 3^{rd} CT-block. This process is illustrated in Figure 3.11.

3.2.6. Generating δ-Difference Pairs (P_1, P_2) such that $E_K(P_1) = E_K(P_2) \oplus \Delta$

After obtaining (S_1, S_2) such that $E_K(S_1) = E_K(S_2) \oplus 1$, ELmD oracle can be exploited to generate (P_1, P_2) such that $E_K(P_1) = E_K(P_2) \oplus \Delta$ for any chosen Δ-value. In this attack, first, the IV is set to 0^{128}. The $3^{-1} \cdot \Delta$-value is used as the decision parameter for the choice of the message blocks. The input block is chosen as S_1 if the corresponding $3^{-1} \cdot \Delta$ bit is 0 and S_2, otherwise. When the constructed message block is queried through the ELmD encryption oracle, the targeted $P_2 = D_K(E_K(P_1) \oplus \Delta)$ is obtain as the 129^{th} CT-block. This process is illustrated in Figure 3.12.

3.2.7. Simulation of Decryption Oracle of the Underlying Block Cipher

In their study, Bay et al. suggest two different ways to simulate the decryption oracle of the underlying block cipher. However, this queries requires a known P_1-$E_K(P_1)$ pair. To mount this attack, they produce two message blocks $R_2 = D_K(E_K(R_1) \oplus 1)$ and $R_3 = D_K(3^{-1} \cdot E_K(R_1))$ by generating 1-difference pair and μ-multiplicative pair attacks. Then they query R_3 and R_2 block as 1^{st} and 2^{nd} PT block and obtain $CC_2 = D_K(0^{127}1)$ value. Using this value as P_1 in the generating μ-multiplicative pair attack, they find any P_2-value for the chosen μ-value.

They modify Demirci-Selcuk attack and get a chosen-ciphertext version of the attack. Then, they use the modified version of the attack with their SDBC model to mount key recovery attack against the ELmD oracle with AES^6 as the underlying block cipher.

3.3. A Semi-Universal Forgery Attack against COLM

In [37], Forler et al. derive a new security approach for integrity of AEAD from an already defined approach, called $INT - CTXT$. Then, they mount $j - IV - CA$ attack various AEAD schemes, including some of the 3^{rd} round candidates of the CAESAR competition. In their work, they mount semi-universal forgery attack against COLM. In their attack, first

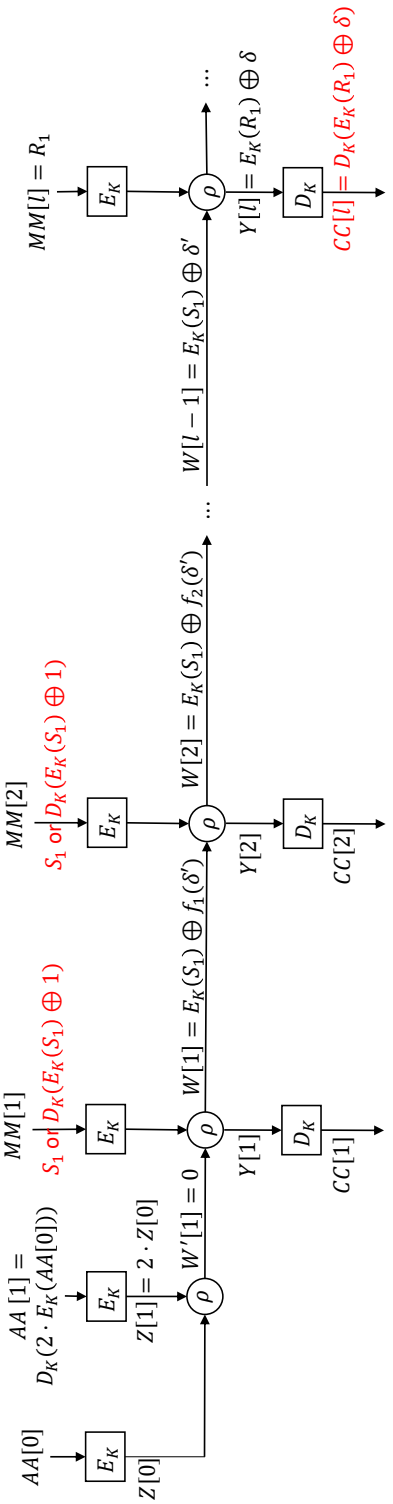

Figure 3.12 Generating δ-Difference Pairs by Bay et al.

they recover the subkey L with the method in Section 3.2.1. In COLM oracle, AD-blocks are mixed to each other by XOR-operation in the linear-mix layer. After the subkey L is recovered, Forler et al. exploits the commutative property of the XOR-operation and output alternative forgeries from each permutation of AD with the same PT after a single query. Therefore, they release $a! - 1$ forgeries from a single query.

3.4. Forgery Attacks against and SEBC of COLM

As the final round of the CAESAR competition approaches, Vaudenay and Vizár analyze and classify the candidates according to their resistance in [12]. They evaluate each of the 15 candidates in terms of forgeries, decryption attacks and key recoveries. In their evaluation, COLM is placed in most second resilient category. Besides the general evaluation and classification, they mount two different L-recovery attack. Both of the attacks are collision-based attacks, the first attack has a low probability of success and the second attack is beyond birthday bound complexity. Moreover, they introduce three distinct types of forgery attack, namely existential forgery, semi-universal forgery, and universal forgery. Finally, they take advantage of structure of COLM to build a SEBC model of COLM.

3.4.1. L-Recovery Attack with Low Probability

In this attack, Vaudenay and Vizár mention that $param$ input in the COLM algorithm is constant and this attack is built on the assumption that 64 bits of a 128-bit value is guessed correctly. They call this value B and assume that the last 64-bit of B is the same as the last 64-bit of $3^2 \cdot L$. For the remaining 64 bits, they conduct a collision attack based on birthday paradox. The attack is mounted with the following steps;

1. Prepare a set of $A_0 \| A_1$'s such that $A_0 = A_1 \oplus B$.
2. Observe a collision in the first CT-block of the queries.
3. Let the colliding blocks be $\widehat{CT_1}$ and $\widetilde{CT_1}$ after the querying the blocks: $(\widehat{N}, param, \widehat{A1}, M)$ and $(\widetilde{N}, param, \widetilde{A1}, M)$.

4. For the colliding blcoks solve the following equation to recover the L-parameter:

$$L = 3^{-2} \cdot ((\widehat{N}\|param) \oplus (\widetilde{N}\|param) \oplus B).$$

This attack works as follows, the collision in the 1^{st} block of CT means that IVs collide, too. Due to the commutative property of the linear-mix function of COLM's AD-process, one can deduce that

$$\begin{aligned} \widehat{AA_0} &= \widetilde{AA_1}, \\ \widetilde{AA_0} &= \widehat{AA_1} \Rightarrow \\ (\widehat{N}\|param \oplus 3 \cdot L) &= (\widetilde{A_1} \oplus 2 \cdot 3 \cdot L) \text{ and} \\ (\widetilde{N}\|param \oplus 3 \cdot L) &= (\widehat{A_1} \oplus 2 \cdot 3 \cdot L). \end{aligned}$$

Since $\widehat{N}\|param \oplus \widehat{A_1} = B$. One can write;

$$\begin{aligned} (\widehat{N}\|param) \oplus (\widetilde{N}\|param) \oplus 3^2 \cdot L &= B \Rightarrow \\ L &= 3^{-2} \cdot ((\widehat{N}\|param) \oplus (\widetilde{N}\|param) \oplus B). \end{aligned}$$

Success probability of this attack is 2^{64} with 2^{32} COLM encryption queries.

3.4.2. L-Recovery Attack Beyond Birthday Bound Complexity

This attack follows the same reasoning as the attack in Section 3.4.1 but unlike the previous attack, it is a nonce misuse attack. For this attack, they build a nonce set and query the each nonce with 2^m different ADs. They prepare 2^{128-2m} batches and within a batch, they keep the nonce the same. The query data can be formulated as follows;

1. Choose 2^{128-2m} nonces,
2. Choose 2^m random two-block AD,

3. Choose a constant M for all queries,

4. Query all $(N^i, A^{i,j}, M)$ for $0 \leq i \leq 2^{128-2m} - 1$ and $0 \leq j \leq m - 1$,

5. Search for collision among the sets in which the same nonce is used i.e. search for $C^{i,\widehat{j}}, T^{i,\widehat{j}}$=$C^{i,\widetilde{j}}, T^{i,\widetilde{j}}$,

6. For the found collision recover the subkey L as in Section 3.4.1 from $A^{i,\widehat{j}}$, $A^{i,\widetilde{j}}$ blocks.

3.4.3. Existential Forgery against COLM

Vaudenay and Vizár generates $AD - PT$ pairs with constant M_1-block. After quering these pais, they search for a collision in the C_1-block. The C_1 depends on IV and M_1. For the given set, a collision C_1 blocks of two different queries implies collision in IV as the M_1 is kept constant. From the $AD - PT$ pairs ($\widehat{N}, \widehat{A}, M_1, \widehat{M_2}$ and $\widetilde{N}, \widetilde{A}, M_1, \widetilde{M_2}$) that output the same C_1 block, the valid messages ($\widehat{N}, \widehat{A}, M_1, \widetilde{M_2}$ and $\widetilde{N}, \widetilde{A}, M_1, \widehat{M_2}$) can be forged.

3.4.4. Semi-universal Forgery against COLM

In this attack, they construct a set of data with random N and A but constant M. Then, they query (N, A, M) until they find a collision such that $C^i \| T^i = C^j \| T^j$. After the collision occur, they query $(N^i, A^i, \widehat{M})$ for any given $\widehat{M}$ and obtain $\widehat{C} \| \widehat{T}$ and forge $(N^j, A^j, \widehat{C}, \widehat{T})$.

3.4.5. Universal Forgery against COLM

In this attack, Vaudenay and Vizár modify A_1 block to $A_2 \oplus 3 \cdot 6 \cdot L$ and A_2 to $A_1 \oplus 3 \cdot 6 \cdot L$ for any given $AD - CT$ pair and release the resulting data as the a forged $AD - CT$ pair.

3.4.6. SEBC Model of COLM

They built their SEBC model in two steps. The first step is a precomputation. In this step, a set of 128 linearly independent CC blocks is aimed to be constructed. A query of 135 blocks

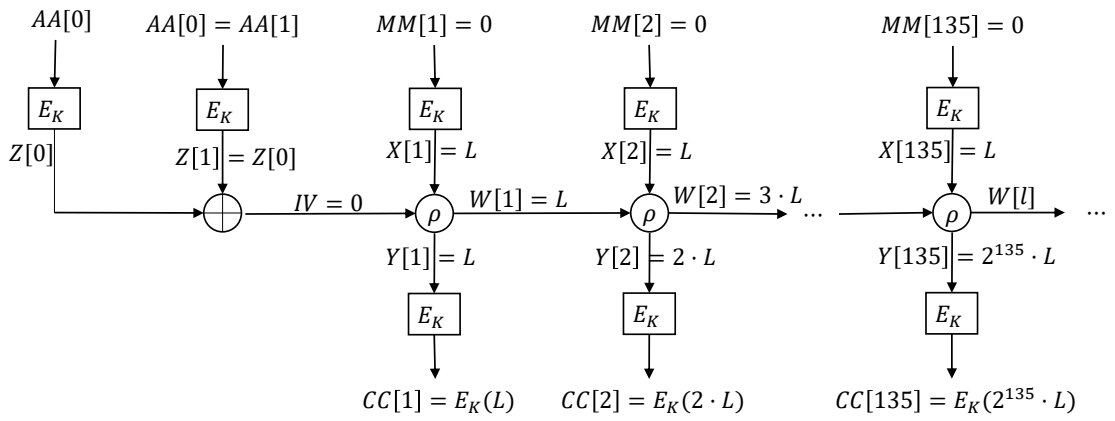

Figure 3.13 1^{st} Query of Vaudenay and Vizár's SEBC Model

(see Figure 3.13) is increase the success probability of this step close to "1". They use a set named as chaining values in the precomputation and one can increase the number of the chaining values until 128 linearly independent CC blocks are obtained. In the second step, they find a subset of precomputed CC blocks whose sum ends up to $3^{-1} \cdot (\beta \oplus L)$ where β is the target to be queried. After the state value is set to 0^{128} by choosing $AA_0 = AA_1$, the remaining AA-blocks are set to blocks related to CC blocks in the calculated subset. Finally MM_1 blocks is set to be 0^{128} and $\alpha = E_K(\beta)$ can be obtained as the MM_1 block. This query is shown in Figure 3.14. The precomputation is formulated as follows;

1. Set $AA_0 = AA_1$,
2. Set $MM_1 = 0^{128}$ for $1 \leq i \leq 135$,
3. Observe that state (chaining) values are $W_i = (2^i - 1) \cdot L$ for $0, 1, \cdots$,
4. Y_i values are $3 \cdot W_{i-1} \oplus X_i = 2^{i-1} \cdot L$ for $1, 2, \cdots$,
5. $CC_i = E_K(Y_i)$ is the set aimed to be constructed in precomputation.

The online query of the attack is formulated as follows;

1. For a given β, calculate $\beta' = 3^{-1} \cdot (\beta \oplus L)$,

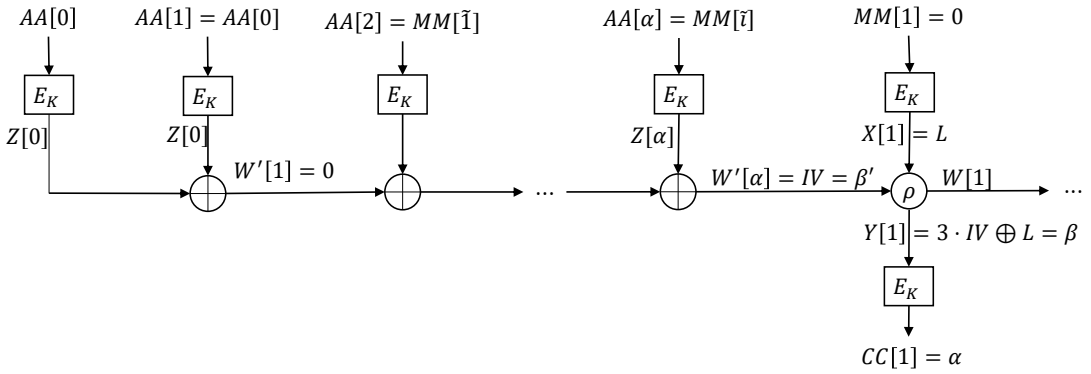

Figure 3.14 2^{nd} Query of Vaudenay and Vizár's SEBC Model

2. From CC_i blocks find a linear solution of β' by Gaussian elimination,
3. Let the index set $\widehat{i} \subseteq i$ gives the solution $\bigoplus_{j \in \widehat{i}} CC_j = \beta'$,
4. Set $AA_0 = AA_1$,
5. Set AA_{j+1} to j^{th} element of the set $\widehat{i}$ for $1 \leq j \leq |\widehat{i}|$,
6. Query $AA_0 \| AA_1 \| AA_2 \| \cdots AA_{|\widehat{i}|+1} \| MM_1 = 0$, for the $MM_{\widehat{i}}$,
7. Observe that $IV = \beta'$ and $Y_1 = \beta$,
8. Obtain $YY_1 = E_K(\beta)$.

Recall that encryption oracle of COLM only uses the underlying block cipher in the encryption direction. As a result, once an adversary is able to query the underlying block cipher of COLM in encryption direction, she can forge CT and tag of any given $AD - PT$ pair.

3.5. DPA Protected Implementation of COLM

As being chosen to final portfolio of CAESAR competition, COLM has also drawn attention of developers. When an encryption schemes without taking related security measures, it might be at risk due to physical attacks aiming to jeopardize the cipher by

attacking its implementation. In [46], Jahanbani et al. discuss application of the threshold implementation (TI) and domain oriented masking (DOM) methods to COLM and verify their countermeasures with t-test over the power traces.

Both TI and DOM are masking schemes aiming to secure operation by separating sensitive data into shares. Designed approaches based TI resist first-order DPA attacks if the scheme is designed with satisfying correctness, non-completeness and uniformity properties. DOM is another masking schemes as secure as TI moreover it can be implemented in less area and with less randomness. After the implementing the cipher with measures against attack, the success of the implementations should be evaluated for this purpose, authors prefer to conduct the first-order t-test instead of DPA. For DPA, the attacker is supposed to have knowledge about the underlying architecture; on the other hand, in t-test, t-statistics are calculated to distinguish two power traces. Although, this method does not lead to key recovery, it shows if two systems can be distinguished from each other.

Protected implementation of COLM can be grouped into two parts;

1. Linear operations: Masking, finite field multiplications, etc,
2. Non-linear operations: S-Boxes.

For the linear operations, repeating the operation as the number of masks is enough; on the other hand, the non-linear parts require complicated methods for masking. In the mask implementation of COLM, the operations not placing in the underlying block cipher AES are all linear operations; therefore, their masked implementations are straightforward. Since the authors aim to make the implementation of a single mask, it is enough to implement all of these operations twice.

The implementation of AES should be further investigated, since it still involves non-linear operations. The TI protected implementation of AES S-Box involves converting the operations in $GF(2^8)$ into $GF(2^4)$ because in $GF(2^8)$ the degree of S-Box is seven, requiring eight shares, and it becomes infeasible. On the other side, implementing the

S-Box operations in $GF(2^4)$ is not straightforward and requires to define fresh random bits. After defining 40 fresh random bits per S-Box, AES S-box can be implemented with a mixture of 2-3 shares and defining three multiplications and a single inversion operation in $GF(2^4)$. For the DOM protected implementation of AES S-Box, also the field should be reduced to $GF(2^2)$. DOM implementation also requires pipelined structure not to reduce the throughput. DOM protected implementation requires 18 fresh random bits per SBox. When the AES implemented in an 8-bit architecture by utilizing a TI/DOM protected SBox, the key and the message are masjed into as follows; $k_1 = m_k$, $k_2 = m_k \oplus k$, $d_1 = m_d$ and $d_2 = m_d \oplus d$, here m_k and m_d are masks for the key k and the data d, respectively. Single AES block encryption takes 246 clocks and 205 clocks in DOM-protected implementation and in TI-protected implementation, respectively.

The authors implement the protected and unprotected designs of COLM for practical benchmark comparison. In the unprotected COLM implementation, $|t|$ value for a non-specific t-test is calculated to be more than 4.5 for the total number of 18000 traces. For the protected setup, t value is calculated for a second order t-test additionally to a non-specific t-test. The authors observed that for 18000 samples, both the TI-protected implementation with hybrid 2-3 shares and DOM-protected implementation with 2 shares pass the t-test. But unfortunately, both TI-protected and DOM-protected implementations fail from the second-order t-test as both implementations are expected to resist against only first-order DPA attacks.

In the study, the authors also note that DOM protected implementation of COLM requires 2.13 times more area to be implemented and produces 1.87 times less throughput than unprotected implementation, TI protected implementation of COLM requires 2.25 times more area to be implemented and produces 1.47 times less throughput than unprotected implementation.

3.6. Pipelined Hardware Implementation of COLM

As the winner of the CAESAR competition, COLM and its variants, COPA and ELmD, have attracted the attention of researchers for implementation methods. In [1], Bossuet et al. implemented COLM, COPA, and ELmD pipelined and compared the performance of these implementations. From the implementation aspect, the authors divide the whole algorithm into the following components;

1. Interface: inputs, outputs and controls,
2. AES: the underlying block cipher (two cores: one for encryption, one for decryption),
3. Encryption: 2 pipelined AES cores,
4. Mult (x2): shift and modular reduction
5. ρ, ρ^{-1}: Mult (x2) and XORs
6. Masking:
   (a) 2 Mults x2
   (b) register
   (c) XORs

Among these, implementation components, the interface operations, mult (x2) function, ρ, ρ^{-1} operations and masking functions are single clock operations and they do not require any strategy. On the other hand, AES and encryption components are complex operations and they are the points where the study focus.

For the main component of the implementation, AES, the authors condider two strategies;

1. Round based: A single round is implemented and iterated 10 times with a seperation at the last round to avoid $MixColumn$ operation. A single block encryption takes ten clocks.

2. Pipelined: All of 10 rounds are cascaded to each other and the next round is fed by the output of the previous round. Although the latency is 10 clocks in pipelined implementation after the first output, it is possible to output in each clock cycle.

The authenticated encryption schemes in the subject family can be paralleled at a certain level. All blocks of the inputs can be computed in parallel until the linear-mix layer and after the linear-mix layer is performed sequentially, the remaining operations can further be performed in parallel. All of the parallel operations involve AES encryption, this means that there can be multiple implementations of AES cores in the design.

On the system level, the authors divide the algorithm in three parts;

1. Associated Data Processing: In this step, three operations;

   - the linear masking,
   - the ECB layer,
   - the output accumulation

   are conducted. The input of the ECB layer are independent to each other, therefore only necessary condition for implementing pipeline architecture is to generate the masking value of each block in each clock cycle.

2. Encryption: Five operations take place in this part;

   - the first masking,
   - the first ECB,
   - the linear mixing,
   - the second ECB,
   - the final masking.

   For an effective solution, the ECB cores required to be implemented in seperate pipeline cores.

AEAD	FFs	LUTs	Slices	Latency (CCs)
COPA	8067	32770	8875	46+(d+l)
ELmD	5677	16959	5163	35+(d+l)
COLM	8243	32593	9076	35+(d+l)

AEAD	Freq. (MHz)	Throughput (Gb/s)	TPA
COPA	265.89	34.03	3.83
ELmD	276.01	35.33	6.84
COLM	260.45	33.34	3.67

Table 3.1 The properties of COPA, ELmD and COLM from [1]

3. Masking and Mixing Function: Whitening masks can be generated in iterative for as a finite field multiplication by 2 to give a valid mask for each clock cycle. Also when necessary, multiple multipliers can be used to generate multiple masking values. Mixing functions are also single clock cycle operations; however, mixing operation for each block has to wait for the process of the all previous blocks. Therefore, they can be implemented simply sequential.

In a block level view, the design is implemented as in Figure 3.15. Being a 10-round cipher, AES core is expected to last 10 clock cycles. The additional clock in the Figure 3.16 is consumed for the input selection of the AES operation. The overall time consumption of the system is illustrated in Figure 3.16. The required time for COLM Encryption and COLM Decryption is nearly equal to each other, the only required difference occur at the stage of tag verification. Due to tag verification the decryption process last 13 clock cycles longer as illustrated in Figure 3.16. Also for COLM Encryption and COLM Decryption operations, AES Encryption and AES Decryption cores should be implemented seperately. For the other operations the same structures can be used in different order. In Table 3.1, implementation properties of COPA, ELmD and COLM are listed for 10-round AES, 128-bit tag length and no intermediate tags settings. When the results on the table are compared, it is easily seen that ELmD has a bigger advantage than COPA and COLM in terms of source usege. This advantage comes from the fact that ELmD encryption and decryption can be built on the same structure unlike to COPA and COLM.

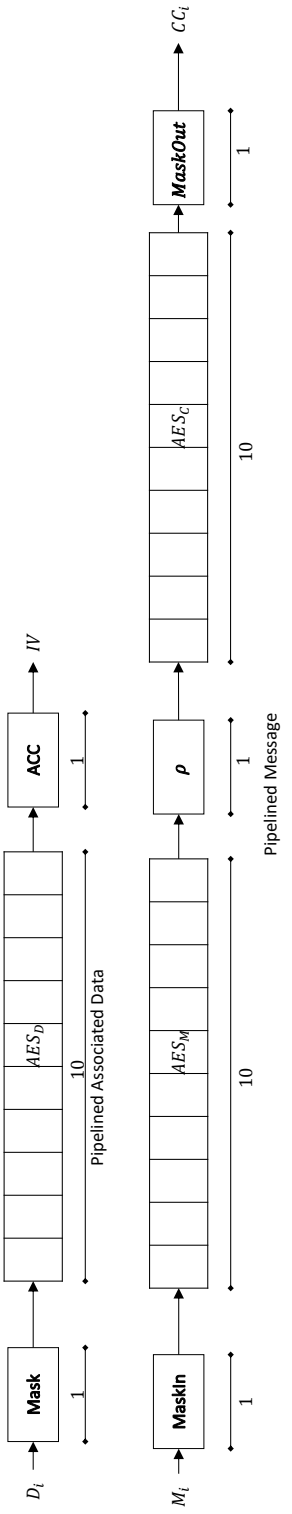

Figure 3.15 Block-Level Pipelined COLM Implementation

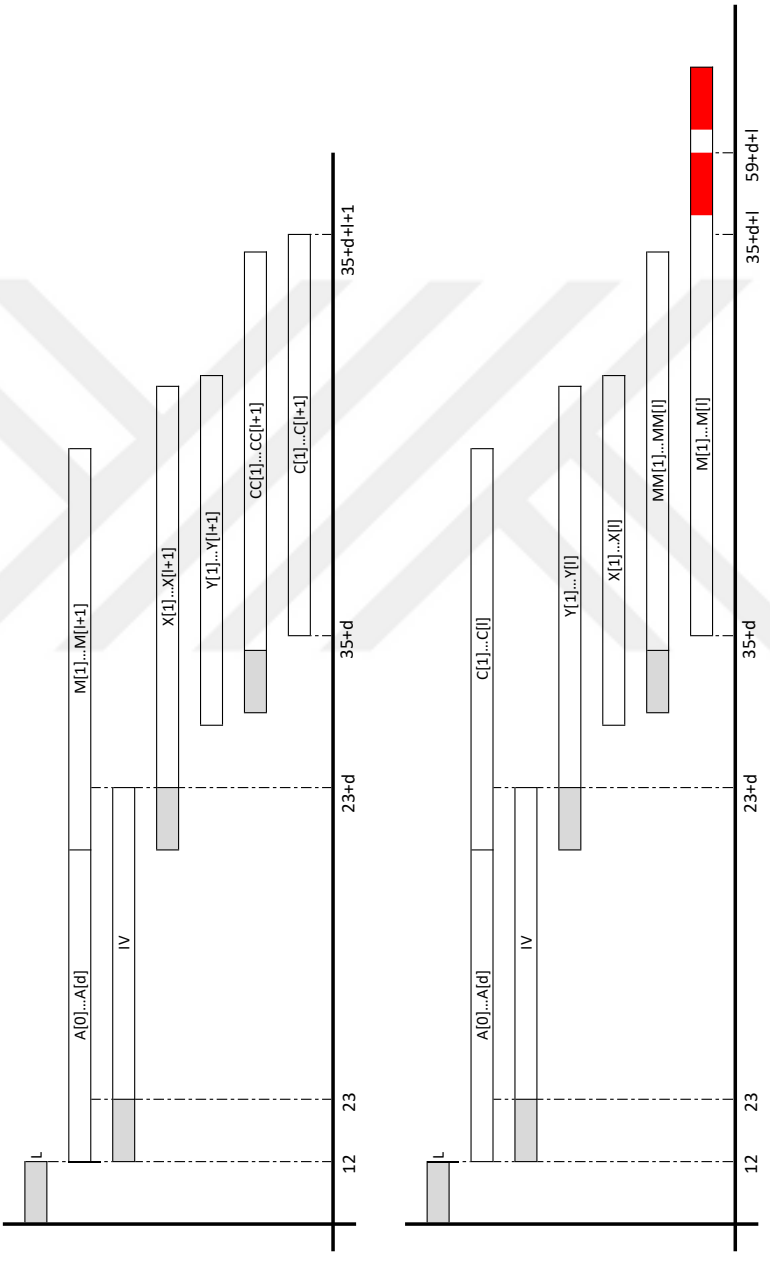

Figure 3.16 Clock Consumption of COLM Encryption/Decryption for d-block AD and l-block Message (The Red Scanned Area is the Additional Time Required for Tag Verification

3.7. Persistent Fault Attack against COLM

3.7.1. Fault Attacks

The fault attack is first introduced by Boneh et al. in [51]. Today, three types of fault attacks are considered in the literature. These attack are;

1. Transient Fault Attack
2. Persistent Fault Attack
3. Permanent Fault Attack

In transient fault attack, the error is injected while the algorithm is running. The effect of transient fault attack disappears in the coming runs of the algorithm. To perform the transient fault attack and observe its effect, the same error should be injected at the same time to same point. Therefore, it might be hard to perform since it requires high level of consistency and synchronization. In permanent fault attack, the error is injected to a cell permanently such EEPROM and either the algorithm or data is changed permanently. The persistent fault attack is in between transient and permanent fault attack. The fault is injected during compilation or synthesis. In this type of attack, the injected fault continues among the encryption processes; on the other hand, when the device is rebooted or the algorithm is recalled from storage, the fault disappears. The persistant fault attack is introduced by Zhang et al. in [52].

3.7.2. Persistent Fault Attack against AES

In [52], Zhang et al. change a single bit in the S-Box of a hardware AES implementation and observe its effect at the output. Cryptographic algorithms have desirable randomness properties when generating data. Therefore; when various data is encrypted in the long run, all possible values are expected to appear with same property. Each byte at the last round of AES can be modelled as in Figure 3.17. As can be inferred from Figure 3.17, after the

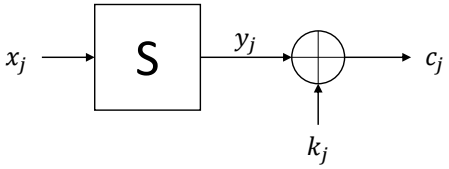

Figure 3.17 Representation of Each Byte in Final Round of AES

error is injected according to PFA model, for each byte, the modified S-Box output (let it be B_{old}) is expected to disappear and its new value (let it be B_{new}) is expected to appear twice as other values. This change in the distribution of S-Box effects the resulting CT as follows; in the j^{th} byte of the output, the value $B_{old} \oplus k_j$ is expected not to appear on the other hand the value $B_{new} \oplus k_j$ is expected to appear twice as other values, where k_j is the j^{th} byte of the last round key.

By considering effect of the corrupted S-Box output, three methods can be used to extract the k_js. These methods are

1. After enough number of erroneous encryption for the missing CT-byte, the following equation can be solved:
$$k_j = B_{old} \oplus CT_{j_{missing}}$$

2. After enough number of erroneous encryption for the appearing CT-byte, the following equation can be used to eliminate impossible key candidates:
$$k_j \neq B_{old} \oplus CT_{j_{missing}}$$

3. After enough number of erroneous encryption for the CT-byte appearing as twice as others, the following equation can be solved to determine the key:
$$k_j = B_{new} \oplus t_{max}$$

The first and second methods are determined methods, contrarily the third method is a probabilistic method. Therefore, the third method requires more erroneous queries than the

first two methods. Another difference between these methods is that in the first two methods, the adversary has to know which S-Box value she changes, on the other hand in the last method she has to know the new value of the modified S-Box.

3.7.3. Mounting PFA against COLM

In case of COLM, the whitening mask is XORed to output of the AES block. Therefore, observing a single CT-block is not enough to mount PFA against COLM. Therefore, the attacker is supposed to use at least two CT-blocks to mount the attack.

In [39], the authors mount the attack in the encryption stage of COLM. In the encryption stage, the whitening mask is derived from the subkey $L_2 = 3^2 \cdot E_K(0)$ and for the i^{th}-block the whitening mask is $2^i \cdot L_2$. Therefore, the i^{th} block of CT can be formulated as;

$$CT_i = B \oplus k \oplus 2^i \cdot L_2,$$

where B is the output of the last SBox, k is the last round key and $2^i \cdot L_2$ is the whitening key for the i^{th} block. From each missing byte of CT_i block the $k \oplus 2^i \cdot L_2 = B_{old} \oplus CT_{i.n}$ is calculated. By applying the same process to the CT_{i+1} output block $k \oplus 2^{i+1} \cdot L_2 = B_{old} \oplus CT_{(i+1).n}$ is calculated.

Let the calculated $k \oplus 2^i$ value be denoted as R_i and the calculated $k \oplus 2^{i+1}$ value be denoted as R_{i+1}. From R_i and R_{i+1} the last round key k can be calculated from the formula:

$$k = 3^{-1} \cdot (2 \cdot R_i \oplus R_{i+1}).$$

3.8. On misuse of nonce-misuse resistance

In [40], Khairallah et al. mounted differential fault attacks against AES-based CAESAR winners. Differential fault attack is a sort of transient fault attack in which the attacker first encrypts the fault-free message then she injects fault at an intermediate point of the cipher

and observes the difference between the output of the fault-free and fault-injected ciphertext. In AEAD schemes, in the decryption stage, the oracle is expected not to release the plaintext unless the tag is verified. Therefore, the authors conduct the attack in the encryption stage as a nonce-misuse attack. The presence of the whitening mask makes the DFA attack more challenging as in the PFA attack in [39]. Consequently, the authors of [40] carry out the attack using two output blocks.

For both blocks, the authors chose the inject the fault before the diffusion layer in Round 8. To observe the effect of the fault, the authors define the Joint Difference Distribution Table ($JDDT$). The purpose of the $JDDT$ is to precompute and store the potential propagation paths of all four affected bytes and to analyze them simultaneously from their joint differential properties. Figure 3.18 illustrates the propagation of the injected fault in Round 8 through the Round 9 and Round 10. As can be observed from the Figure 3.18, the injected fault propagates deterministically through linear layer of Round 8. After the probabilistic change of the difference in the S-layer of Round 9, the result propagates further through Round 9 and after Round 10 the difference ends up 4 groups of 4 bytes each having $\approx 2^8$ candidates for the value $A_i = K_{10} \oplus WM_i$. In total there will be $\approx 2^{32}$ candidates for the A_i value on average. However, the WM_i is also not known, therefore, it is impossible to retrieve information from single block. When the fault is injected into two consecutive blocks, the attacker obtains the following two equations:

$$A_i = K_{10} \oplus WM_i,$$

$$A_{i+1} = K_{10} \oplus WM_{i+1},$$

here $WM_i = 2 \cdot WM_{i+1}$.

As it can be seen from the Figure 3.18, for each of the 4 groups, the equations similar to below (equations for the 3^{rd} column as an example) can be written for the difference in each

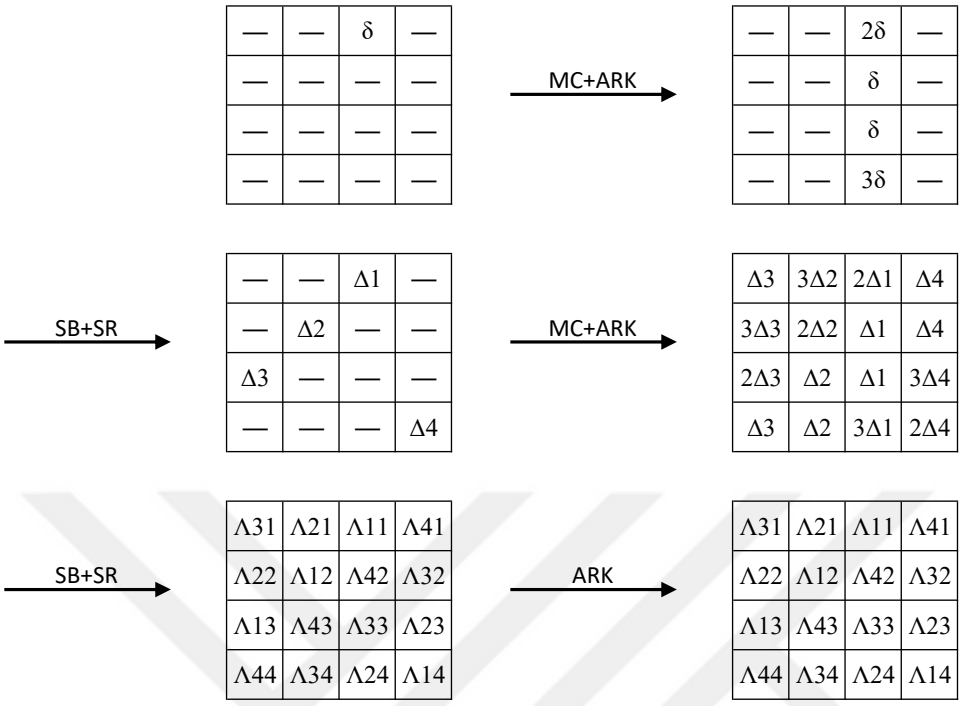

Figure 3.18 Illustration of Adaptive Differential Fault Analysis

block:

$$\begin{aligned} A[31:0] &= K_{10}[127:120]\|K_{10}[23:16]\|K_{10}[47:40]\|K_{10}[71:64] \\ &\oplus\ WM_i[127:120]\|WM_i[23:16]\|WM_i[47:40]\|WM_i[71:64] \quad (3) \\ B[31:0] &= K_{10}[127:120]\|K_{10}[23:16]\|K_{10}[47:40]\|K_{10}[71:64] \\ &\oplus\ WM_i[126:119]\|WM_i[22:15]\|WM_i[46:39]\|WM_i[70:63] \quad (4) \end{aligned}$$

From Equation 3 and Equation 4, 32 equations can be written of 36 variables for each group. Besides, for each group 2^{16} equation systems can be written resulting in 2^{20} candidates in total. When the results of the four groups are combined with each other, the attacker ends up with 2^{80} full key candidates. It is not practical identify the key among these amount of candidates. Therefore, the authors decide to mount the attack with injecting the third fault to the next block. As a result of the injected third fault, the authors arrive at the following

equation:

$$\begin{aligned} C[31:0] &= K_{10}[127:120] \| K_{10}[23:16] \| K_{10}[47:40] \| K_{10}[71:64] \\ &\oplus WM_i[125:118] \| WM_i[21:14] \| WM_i[45:38] \| WM_i[69:61] \end{aligned} \quad (5)$$

When Equation 4 and Equation 5 are considered together under the assumption that false positives are distributed uniformly, it is expected that number of key candidates reduce to 2^{32}. Furthermore, the candidate keys can be filtered out by considering Equation 3 and Equation 5. After the final filter, it is expected that only one or zero incorrect key candidates remains for each group. It is expected to have 16 key candidates for the full key. These remaining key candidates can be eliminated by brute-force and the correct key can be recovered.

When the authors verified their analysis experimentally in software, they achieved to recover the correct key. However, the final intersection set involves more than 1 suggestions, due to the selection of the free variables, the suggested solution was equal to the correct key. Authors also note that with the help of the $JDDT$ table, the time complexity of the analysis is $O(nlogn)$, where $n \approx 2^{20}$.

3.9. RUP Attacks against COLM

In [38], Datta et al. investigate the release of unverified plaintext integrity of COLM. When designing COLM, the designers decided to XOR all input blocks to calculate the $M[l]$-block despite the antecedents of COLM. In this study, Datta et al. study a special version of COLM in which this modification is omitted.

In the study, they figure out that ELmD/ELmE structured authenticated encryption schemes' resistances against RUP attacks strongly depend on the linear mixing function ρ. Therefore, in their work, the linear mixing function of COLM is analyzed in its general case as;

$$\rho(X, W) = (Y, W') = (a \cdot X \oplus b \cdot W, c \cdot X \oplus d \cdot W).$$

((a, b, c, d) values are chosen as $(1, 1, 1, 1)$ in COPA and $(1, 3, 1, 2)$ in ELmD and COLM.)

In the study, the following three attack settings are considered;

- XOR as linear mixing function under nonce-respecting setting
- a ρ alike function as linear mixing under nonce-ignorant setting
- a ρ alike function linear mixing under nonce-respecting setting

3.9.1. Nonce-Respecting INT-RUP Attack for XOR-Mixing

In this setting, they show that an attacker is able to compromise the integrity of the cipher only within a single encryption query and 2 decryption queries.

In the attack scenario, the adversary $\mathcal{A}$ first makes the following query;

$$(C_1^*, \dots C_l^*, T^*) = \mathcal{E}_K(N^*, A^*, M_1^*, \dots M_l^*).$$

From this query, $\mathcal{A}$ targets to construct a new ciphertext

$$(N^*, A^*, C_1, \dots C_l, T^*)$$

such that the following equalities match:

$$C_l = C_l^*, \tag{6}$$

$$\bigoplus_{i=1}^{l} M_i = \bigoplus_{i=1}^{l} M_i^*. \tag{7}$$

For the Adversary $\mathcal{A}$, it is enough to find ciphertext matching only these inequalities, because, as it can be seen from Figure 2.1, the last XOR-operation is built on the following equality:

$$D_K(C_l^* \oplus 2^l L) \oplus D_K(T^* \oplus 2^{l-1}7L) = E_K(\bigoplus_{i=1}^{l} M_i^* \oplus 2^{l-1}3^2 L). \quad (8)$$

When the attacker achieves to construct a PT-CT pair satisfying the equalities 6 and 7, the constructed PT-CT pair naturally will satisfy the equality 8.

The authors demonstrate how to construct such a PT-CT pair with the following steps:

- Choose $l/2$, n-bit strings C_i^0, C_i^1 such that C_i^0, C_i^1 and C_i^* are mutually distinct for $i \in 1, 3, \ldots, l-1$ and define the following CTs:

$$C^0 = C_1^0 C_2^* C_3^0 \ldots C_{l-2}^* C_{l-1}^0 C_l^*$$

$$C^1 = C_1^1 C_2^* C_3^1 \ldots C_{l-2}^* C_{l-1}^1 C_l^*$$

- In this step, the authors make the RUP assumption and presume the following PTs leak; even though, they are not verified.

$$M_1^{*,0} M_2^{0,*} M_3^{*,0} \ldots M_{l-1}^{*,0} M_l^{0,*} \leftarrow \mathcal{D}_K(N^*, A^*, C^0),$$

$$M_1^{*,1} M_2^{1,*} M_3^{*,1} \ldots M_{l-1}^{*,1} M_l^{1,*} \leftarrow \mathcal{D}_K(N^*, A^*, C^1),$$

  As explained in Section 2.7.1, M_i block can be calculated from C_{l-1} and C_l blocks.

- From the leaked M-blocks a valid but illegitimate plaintext can be constructed holding the following equation:

$$\bigoplus_{i=1}^{l} M_i^{b_{i-1},b_i} = \bigoplus_{i=1}^{l} M_i^*$$

  This equality can be solved with probability $1 - 2^{n-l/2}$ by Gaussian elimination.

- Finally, the following forgery ciphertext can be output:

$$(N^*, A^*, C_1^{b_1} \ldots C_{l-1}^{b_{l-1}} C_l^{b_l}, T^*)$$

3.9.2. Nonce-Misuse INT-RUP Attack for Generalized XOR-Mixing

In this attack, the authors generalizes the formulation of ρ and lose the ease coming from the XOR-operation. Therefore, to generate a collision in an intermediate state, they have to make additional encryption queries with the same nonce. In this attack, they aim to find a intermediate block independent of the some previous known blocks. To mount the attack, they make the following queries (in the queries tags are not shown);

$$C_1^0 C_2^0 C_3^0 \leftarrow \mathcal{E}_K(N, A, M_1^0 \star \star),$$

$$M_1^1 M_2^1 M_3^1 \leftarrow \mathcal{D}_K(N, A, \star C_2^0 \star),$$

$$M_1^2 M_2^2 M_3^2 \leftarrow \mathcal{D}_K(N, A, \star \star C_3^0),$$

$$C_1^3 C_2^3 C_3^3 \leftarrow \mathcal{E}_K(N, A, \star M_2^1 M_3^2),$$

($\star$ represents any n-bit value). After these queries, the authors observe that the intermediate block W_3^3 of the last query is independent of the previously chosen and known M_1^0 block. They show this by expanding the formula of W_3^3 block till W_0.

$$W_3^3 = c \cdot X_3^3 \oplus cd \cdot X_2^3 \oplus cd^2 \cdot X_1^3 \oplus d^3 \cdot W_0 \tag{9}$$

(Notice that, the last term is W_0, because N and A are the same in all four queries).

In the last query, the second and the third blocks are chosen from the result of the first query to make $X_2^3 = X_2^1$ and $X_3^3 = X_3^2$. In a similar manner, the second block of the second query and the third block of the third query are chosen from the result of the first query. Therefore, the equalities $Y_2^1 = Y_2^0$ and $Y_3^2 = Y_3^0$ also hold. This yields to

$$X_3^3 = X_3^2 = a^{-1} \cdot (Y_3^2 \oplus b \cdot W_2^2) = a^{-1} \cdot (Y_3^0 \oplus b \cdot W_2^2),$$

$$X_2^3 = X_2^1 = a^{-1} \cdot (Y_2^1 \oplus b \cdot W_1^1) = a^{-1} \cdot (Y_2^0 \oplus b \cdot W_1^1),$$

$$W_3^3 = a^{-1}c \cdot (Y_3^0 \oplus d \cdot Y_2^0) \oplus a^{-1}bc \cdot W_2^2 \oplus a^{-1}bcd \cdot W_1^1 \oplus cd^2 \cdot X_1^3 \oplus d^3 \cdot W_0,$$

$$W_3^3 = c \cdot X_3^0 \oplus a^{-1}c(bc \oplus ad) \cdot X_2^0 \oplus a^{-1}bc \cdot W_2^2 \oplus a^{-1}bcd \cdot W_1^1 \oplus cd^2 \cdot X_1^3 \oplus d^3 \cdot W_0.$$

In the last equation, all components of the W_3^3 are independent of M_1^0.

In the next step, the authors construct two different 3-block messages $M_1M_2M_3 \neq M_1'M_2'M_3'$ whose final states satisfy $W_3 = W_3'$ to two distinct messages with $M_1^0 \neq M_1^{0'}$, $M_2^0 = M_2^{0'}$ and $M_3^0 = M_3^{0'}$. The resulting messages are

$$M_1M_2M_3 := M_1^3M_2^1M_3^2$$

$$M_1'M_2'M_3' := M_1^3M_2^{1'}M_3^{2'}.$$

In the last step of their attacks, the authors show that the adversary $\mathcal{A}$ can make $4l/3 + 1$ encryption queries and $4l/3 + 1$ decryption queries, each of length at most l blocks to forge the output $(N, A, \overline{C_1}^{b_1} \dots \overline{C_\mu}^{b_\mu}, T)$ with probability $p \geq 1 - 2^{n-l/3}$, where $l, n \geq 1$ and $l \geq 3n$. They start the construction by generalizing the collision condition at W_0 to each W_{3l} by applying their observation in Equation 9 to each 3-block chunks. After constructing two different n-chunk messages blocks by applying their method to construct message blocks having $W_{3l} = W_{3l}'$. They show that an output can be forged with probability $1 - 2^{n-\mu}$ due to the following equation:

$$\bigoplus_{i=1}^{\mu} M_{3i-2}^{b_i} \oplus M_{3i-1}^{b_i} \oplus M_{3i}^{b_i} = \bigoplus_{i=1}^{l} M_i.$$

Satisfying this equation is necessary for the tag T to hold.

3.9.3. Nonce-Respecting INT-RUP Attack for Generalized XOR-Mixing

In the last part of their studies, the authors construct a method to forge a ciphertext for any ρ-function in nonce-respecting setting. The attack conditions are stringent, the required length of the forged ciphertext is longer than the previous attacks. The message is required

to be at least n^2 blocks, where n is the block length. Since the this attack is mounted in nonce-respecting setting, and in an encryption query interfering the nonce may result in unrecoverable randomness in the query, the authors avoid the encryption queries and build the attack from the decryption queries. To begin with, the authors make the two following unauthenticated queries:

$$M_1^0 \dots M_{n+1}^0 \leftarrow \mathcal{D}_K(N, A, C_1^0 C_1^0 \dots C_1^0),$$

$$M_1^1 \dots M_{n+1}^1 \leftarrow \mathcal{D}_K(N, A, C_1^1 C_1^0 \dots C_1^0),$$

From these queries, the authors show that a valid plaintext-ciphertext pair can be constructed such that

$$C_1^2 \dots C_{n+1}^2 \leftarrow \mathcal{E}_K(N, A, M_1^{2_1} \dots M_{n+1}^{2_{n+1}})$$

where

$$2_i = \begin{cases} 0 & i \in I_0 \\ 1 & i \in I_1 \end{cases}$$

and there is collision in the $n+1^{st}$ intermediate values in the second (W_{n+1}^1) and third (W_{n+1}^2) query. To construct the solution, the authors also append the value 1 in the set I_0. Then, they write the $n+1^{st}$ intermediate value of the third query (W_{n+1}^2) in terms of the encrypted plaintext blocks X_i's such as;

$$W_{n+1}^2 = d^{n+1} W_0 \oplus c d^n X_1^{2_1} \oplus \dots \oplus c d X_n^n \oplus c X_{n+1}^{2_{n+1}}$$

Then, they rewrite the same equation in terms of Y-intermediate values (the intermediate values to be encrypted to calculate the CT-blocks).

$$\begin{aligned}
W_{n+1}^2 &= d^{n+1}W_0 \\
&\oplus\ a^{-1}cd^n(Y_1^{2_1} \oplus bW_0) \\
&\oplus\ a^{-1}cd^{n-1}(Y_1^{2_2} \oplus a^{-1}bcY_1^{2_2}beW_0) \\
&\oplus\ \cdots \\
&\oplus\ a^{-1}cd(Y_n^{2n} \oplus a^{-1}bcY_{n-1}^{2n} \oplus \cdots \oplus a^{-1}bce^{n-2}Y_1^{2_n} \oplus be^{n-1}W_0 \\
&\oplus\ a^{-1}c(Y_{n+1}^{2n+1} \oplus a^{-1}bcY_n^{2n+1} \oplus \cdots \oplus a^{-1}bce^{n-1}Y_1^{2_{n+1}} \oplus be^nW_0
\end{aligned}$$

In this equation, $e = a^{-1}bc \oplus d$ and $a, b, c, d \neq 0$. In the next step, the equation is arranged in the form:

$$\begin{aligned}
W_{n+1}^2 &= \bigoplus_{i=2}^{n+1}(\bigoplus_{j=0}^{n-i} a^{-2}bc^2d^je^{n-i-j} \oplus a^{-1}cd^{n+1-j})Y_i^0 \\
&\oplus\ \Xi^0 \cdot Y_1^0 \oplus \Xi^1 \cdot Y_1^1 \\
&\oplus\ (a^{-1}bce^n \oplus a^{-1}bcde^{n-1} \oplus \cdots \oplus a^{-1}bcd^n \oplus d^{n+1})W_0.
\end{aligned}$$

Here,

$$\begin{aligned}
\Xi^0 &:= \bigoplus_{i \in I_0 \setminus \{1\}} a^{-2}bc^2d^{n+1-i}e^{i-2} \oplus a^{-1}cd^n, \\
\Xi^1 &:= \bigoplus_{i \in I_1} a^{-2}bc^2d^{n+1-i}e^{i-2} \\
&= \Xi^0 \oplus (\bigoplus_{j=0}^{n-1} a^{-2}bc^2d^je^{n-1-j} \oplus a^{-1}cd^n).
\end{aligned}$$

Note that, the last term of the Ξ^0 equation ($a^{-1}cd^n$) comes from the excluded element ($\{1\}$) of the set I_0.

Following the same steps W^1_{n+1} can be written as:

$$\begin{aligned} W^1_{n+1} &= \bigoplus_{i=2}^{n+1}(\bigoplus_{j=0}^{n-i} a^{-2}bc^2d^je^{n-i-j} \oplus a^{-1}cd^{n+1-j})Y^0_i \\ &\oplus \ (\bigoplus_{j=0}^{n-i} a^{-2}bc^2d^je^{n-1-j} \oplus a^{-1}cd^n)Y^1_i \\ &\oplus \ (a^{-1}bce^n \oplus a^{-1}bcde^{n-1} \oplus \cdots \oplus a^{-1}bcd^n \oplus d^{n+1})W_0. \end{aligned}$$

When the two intermediate values W^1_{n+1} and W^2_{n+1} XORed to each other, the following result is obtained;

$$W^1_{n+1} \oplus W^2_{n+1} = \Xi^0 \cdot (Y^0_1 \oplus Y^1_1).$$

In the next step, they define a new variable $f = ed^{-1}$ and write the following equation:

$$\begin{aligned} ac^{-1}d^{-n}\Xi^0 &= \bigoplus_{i \in I_0 \setminus \{1\}} a^{-1}bcd^{-1}(a^{-1}bcd^{-1} \oplus 1)^{i-2} \oplus 1 \\ &= \bigoplus_{i \in I_0 \setminus \{1\}} (f \oplus 1)f^{i-2} \oplus 1. \end{aligned}$$

The rest of the solution reduces to showing there exists a subset $I'_0 \subseteq \{0, \ldots, n-1\}$ such that

$$\bigoplus_{i \in I'_0} f^i = (f \oplus 1)^{-1}.$$

This subset gives the solution for a collision in the intermediate values W^1_{n+1} and W^2_{n+1}.

The above operations show that in the nonce-respecting setting, for the generalized ρ-function, it is possible to find a collision between W_{n+1} intermediate blocks of two RUPs.

In Section 3.9.2, it has already been explained that to forge a ciphertext with probability $p \geq 1 - 2^{n-l/(n+1}$, the adversary is required the find l colliding intermediate states, where $l \geq (n+1)n$. Therefore, one can conclude to result that for generalized ρ-function, in the nonce-respecting scenario the adversary $\mathcal{A}$ has to make 2 decryption queries length of $n+1$ blocks to find a collision for each intermediate values. As it is necessary to find n collisions

to forge the ciphertext, the adversary has to make $2(n+1)$ decryption queries of length from $n+1$ to $n(n+1)$, accordingly. Then the forged ciphertext

$$(N, A, \overline{C_1}^{b_1} \dots \overline{C_\mu}^{b_\mu}, T)$$

can be output.

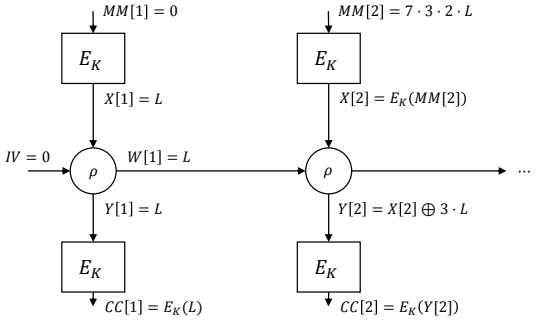

Figure 4.1 Encryption Phase of Obtaining $S = D_K(0)$

4. How to Recover $S = D_K(0)$ through an Existential Forgery

Our principal goal is to recover $S = D_K(0)$, the decryption of the zero-vector for the underlying block cipher of COLM, to mount both the tag guessing and plaintext recovery attacks. Furthermore, it is a milestone as being the first step of the simulation models of both the encryption (see Section 5.2) and the decryption (see Section 5.1) oracles of the underlying block cipher. To recover S, we need to make decryption query of the AEAD-scheme. We further extend the decryption query to a universal decryption query in Section 6, Section 7 and Section 8. Recall that it is much more difficult to query an AEAD structure in decryption direction since one needs the valid tag for the query. To the best of our knowledge, this is the first work making an existential forgery that discloses plaintext-ciphertext pair from the underlying block cipher of an authenticated EME/EMD scheme. We remind that the simulation of underlying block cipher attacks in [11] and [12] make encryption queries of the AEAD-schemes, ELmD and COLM, respectively.

Theorem 4.1 states how to recover $S = D_K(0)$ with an existential forgery attack. Figure 4.1 and Figure 4.2 depict how to recover $S = D_K(0)$. For simplicity, we take $MM[1] = 0$ in Figure 4.1 and the tag verification is illustrated by using $\widetilde{MM'}[3]$ in Figure 4.2. The proof of Theorem 4.1 uses Lemma 4.2, Proposition 4.3 and Theorem 4.4. Therefore, we give the proof as a separate section.

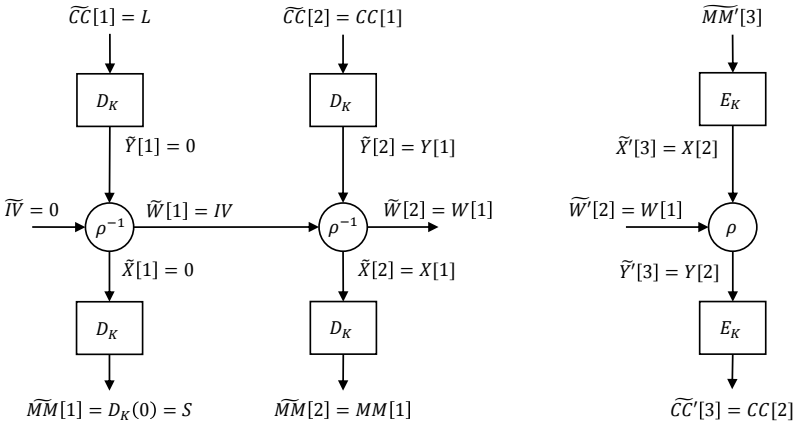

Figure 4.2 Decryption Phase of Obtaining $S = D_K(0)$. Note that the tag is verified since $\widetilde{MM'}[3] = MM[1] \oplus 7 \cdot 3 \cdot 2 \cdot L = MM[2]$.

Theorem 4.1. *Let* $CC[1]\|CC[2]$ *be the* CT *of* $AA[0]\|AA[1]\|MM[1]\|MM[2]$ *where* $AA[0] = AA[1]$ *and* $MM[1] \oplus MM[2] = 7 \cdot 3 \cdot 2 \cdot L$. *Then* $\widetilde{CT} = L\|CC[1]$ *is a valid ciphertext with* $AA[0]\|AA[1]$ *as* $\widetilde{AD}$ *and* $CC[2]$ *as the tag. Moreover, the first block of its plaintext is* $\widetilde{MM}[1] = S = D_K(0)$.

4.1. Proof of Theorem 4.1

Theorem 4.1 has two claims. The former is that $\widetilde{CT}$ is a valid ciphertext and the latter is $S = D_K(0) = \widetilde{MM}[1]$.

We introduce three statements to prove these claims. The first statement is Lemma 4.2 where we show a method to satisfy $\widetilde{W}[l-1] = IV$. In Proposition 4.3, we introduce how to produce the tag of a set of $\widetilde{CT}$s under the condition $\widetilde{W}[l-1] = IV$. We combine Lemma 4.2 and Proposition 4.3 to produce a forged $\widetilde{CT}$ with a valid tag in Theorem 4.4.

Lemma 4.2 exploits a property of the inverse of the linear operator, ρ^{-1}, to satisfy $\widetilde{W}[l-1] = IV$.

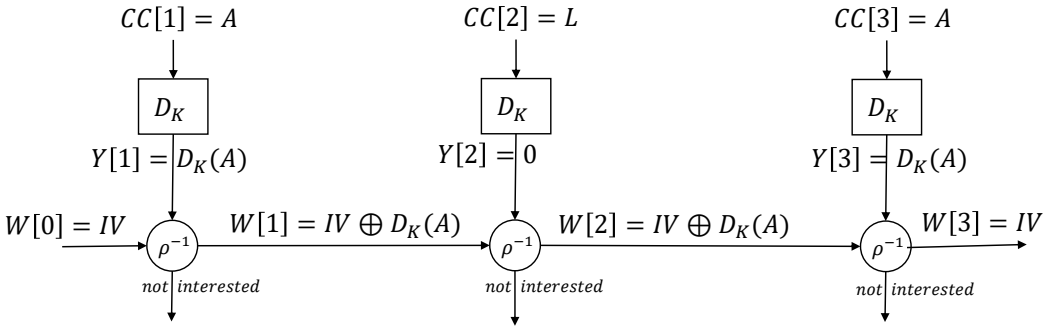

Figure 4.3 An Example of Lemma 4.2, $CC[2] = L$ doesn't have any effect on state value and $CC[3] = A$ compensates effect of $CC[1] = A$.

Lemma 4.2. *For any l-block CT whose initial value is IV and whose $CC[i]$ blocks take any value, except $L = E_K(0)$, even number of times for $1 \leq i < l$ then $W[l-1] = IV$. (An example is illustrated in Figure 4.3)*

Proof. Let CT be a ciphertext of l blocks such that its $CC[i]$ blocks take any value, except $L = E_K(0)$, even number of times for $1 \leq i < l$. That is, for any given $i < l$ if $CC[i] \neq L$ then $\exists j \neq i$ such that $CC[i] = CC[j]$ and the number of $j < l$ satisfying this equality is odd. This simply implies that the $Y[i]$ blocks take a value except $D_K(L) = 0$ even number of times for $1 \leq i < l$. On the other hand, $W[l-1] = \bigoplus_{i=1}^{l-1} Y[i] \bigoplus IV$. Hence one can deduce $W[l-1] = IV$ since nonzero $Y[i]$s will vanish pairwise. □

The following statement provides us a method to produce the tag of a set of $\widetilde{CT}$s where $\widetilde{W}[l-1] = IV$ by using the first two CT blocks of a chosen PT of at least two blocks.

Proposition 4.3. *Let a given plaintext PT be encrypted with an initial value IV and the first two blocks of the produced CT be $CC[1]$ and $CC[2]$. Assume $MM[1] \oplus MM[2] = 7 \cdot 3 \cdot 2^{l-1} \cdot L$. Then any forged l-block $\widetilde{CT}$ whose $\widetilde{W}[l-1] = IV$ and $\widetilde{CC}[l] = CC[1]$ has the valid tag $\widetilde{CC}[l+1] = CC[2]$. (See Figure 4.4.)*

Proof. Let a plaintext PT be given such that $MM[1] \oplus MM[2] = 7 \cdot 3 \cdot 2^{l-1} \cdot L$. Assume $CC[1]$ and $CC[2]$ are the first two blocks of the produced CT by encrypting the PT with

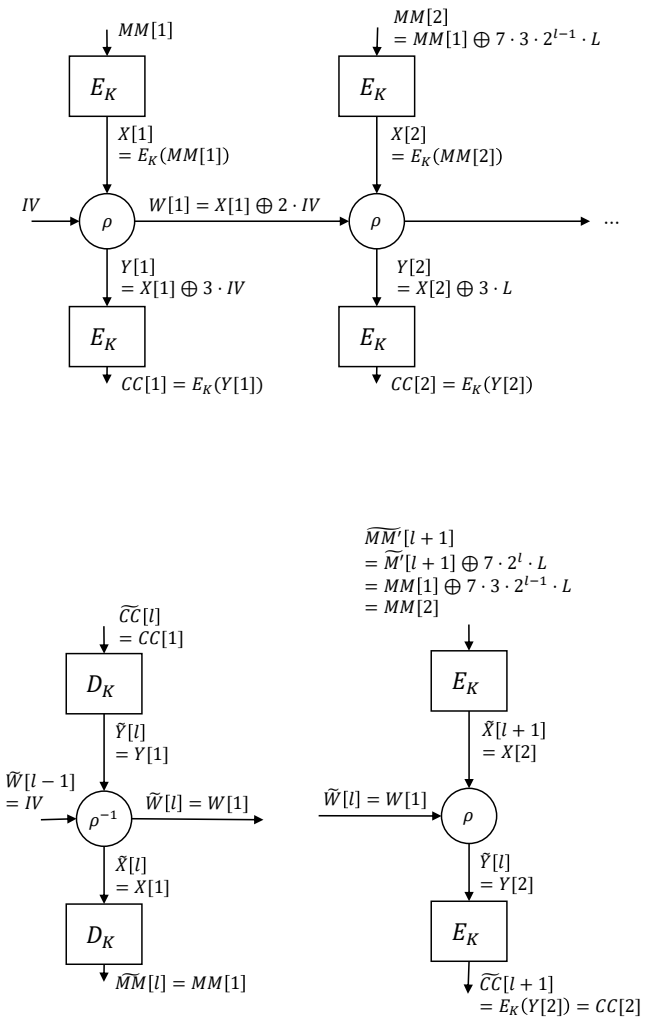

Figure 4.4 Illustration of Proposition 4.3

IV. Let a forged l-block $\widetilde{CT}$ have $\widetilde{CC}[l] = CC[1]$ and $\widetilde{CC}[l+1] = CC[2]$. Observe that

$$\begin{aligned}\widetilde{MM}[l] &= D_K(D_K(\widetilde{CC}[l]) \oplus 3 \cdot \widetilde{W}[l-1]) \\ &= D_K(D_K(CC[1]) \oplus 3 \cdot IV) \\ &= MM[1]\end{aligned}$$

if $\widetilde{W}[l-1] = IV$. Similarly $\widetilde{MM}[l+1] = MM[2]$. Therefore the tag is valid since $MM[1] \oplus MM[2] = 7 \cdot 3 \cdot 2^{l-1} \cdot L$. □

We can combine Lemma 4.2 and Proposition 4.3 to produce a forged $\widetilde{CT}$ with the valid tag which is stated by Theorem 4.4.

Theorem 4.4. *Let a given plaintext PT be encrypted with the initial value IV and the first two blocks of the produced CT be $CC[1]$ and $CC[2]$. Assume $MM[1] \oplus MM[2] = 7 \cdot 3 \cdot 2^{l-1} \cdot L$. For any forged l-block $\widetilde{CT}$ whose $\widetilde{CC}[i]$ blocks take any value, except $L = E_K(0)$, even number of times for $1 \leq i < l$, if $\widetilde{CC}[l] = CC[1]$ and $\widetilde{CC}[l+1] = CC[2]$, then $\widetilde{CT}$ with IV is a valid ciphertext.*

Proof. For an encrypted plaintext PT, assume $MM[1] \oplus MM[2] = 7 \cdot 3 \cdot 2^{l-1} \cdot L$. Let the first two blocks of its ciphertext be $CC[1]$ and $CC[2]$. Let $\widetilde{CT}$ be a l-block forged ciphertext such that its $\widetilde{CC}[i]$ blocks take any value, except $L = E_K(0)$, even number of times for $1 \leq i < l$. Then, we have $\widetilde{W}[l-1] = IV$ by Lemma 4.2. Therefore, $\widetilde{CT}$ is a valid ciphertext by Proposition 4.3. □

Now, we can give the proof of Theorem 4.1.

Proof of Theorem 4.1. Let $CC[1]\|CC[2]$ be the CT of

$$AA[0]\|AA[1]\|MM[1]\|MM[2]$$

where $AA[0] = AA[1]$ and $MM[1] \oplus MM[2] = 7 \cdot 3 \cdot 2 \cdot L$. Then, the forged ciphertext $\widetilde{CT} = L\|CC[1]$ is valid with $AD = AA[0]\|AA[1]$ and the tag $CC[2]$ by Theorem 4.4. Note that $IV = E_K(AA[0]) \oplus E_K(AA[1]) = 0$, $\widetilde{Y}[1] = D_K(L) = 0$ and $\widetilde{X}[1] = \widetilde{Y}[1] \oplus 3 \cdot IV = 0$. Hence, we obtain the first block $\widetilde{MM}[1] = D_K(0) = S$ by decrypting $\widetilde{CT}$. □

4.2. A Method to Produce ADs Having the same IV

Similar to Lemma 4.2, we can manipulate any AD by inserting certain values by utilizing $\widetilde{MM}[1] = D_K(0) = S$ outcome of Theorem 4.1. Lemma 4.5 states this manipulation precisely. Note that instead of the whitening mask L, the S value is used as in Lemma 4.2.

Lemma 4.5. *For any given AD, inserting any number of the S value or any even number of other values into the AD will not change IV.*

Proof. Let an $AD = AA[0]\| \ldots \|AA[a]$ be given with the blocks, its $IV = \bigoplus_{i=0}^{a} X[i]$ where $X[i] = E_K(AA[i])$. Inserting the S value into any block position of AD will result $IV = \bigoplus_{i=0}^{a} X[i] \bigoplus E_K(S) = IV$. Similarly, inserting a different value, V twice into any two block positions of AD will result

$$IV = \bigoplus_{i=0}^{a} X[i] \bigoplus E_K(V) \bigoplus E_K(V) = IV.$$

Therefore, by induction, inserting any number of S or any even number of other values into the AD will not change its IV. □

Remark 4.6. Lemma 4.5 gives us a method to produce several other ADs having the same IV as that of a given AD. Therefore, in all our attacks, we can choose different associated data without changing IV while querying the encryption/decryption oracles of COLM. For the sake of simplicity, we query plaintexts or ciphertexts without changing the AD throughout the paper just to mean that we use the same IV. It is straightforward that we can easily produce different associated data, when necessary, for each query by Lemma 4.5.

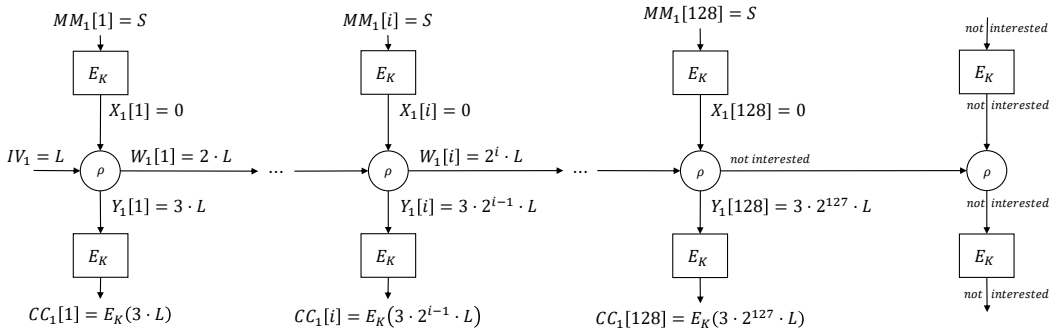

Figure 5.1 Constructing $E_K(2^i \cdot \gamma)$ for $\gamma = 3 \cdot L$ (see Figure 5.2 for setting $IV_1 = L$).

5. Simulation Models of Encryption/Decryption Oracles of the Underlying Block Cipher

In this section, we show how to simulate both the encryption and the decryption oracles of the underlying block cipher (SEBC/SDBC) for any given input. We show that the COLM oracle can be used to query $\alpha = E_K(\beta)$ and $\beta = D_K(\alpha)$ for any given β and α, respectively. For both these tasks, We use the $S = D_K(0)$ value besides the whitening mask L. Recall that $S = D_K(0)$ can be recovered by Theorem 4.1.

5.1. Simulation Model of the Decryption Oracle of the Underlying Block Cipher: Computing $\beta = D_K(\alpha)$

In this section, we introduce a simulation model of the decryption oracle of the underlying block cipher (SDBC) for any given input, i.e. how to compute $\beta = D_K(\alpha)$ for any given α.

Both simulation models of Bay et al. in [11] and of Vaudenay and Vizár in [12] are standard encryption queries of an AEAD scheme, ELmD and COLM, respectively. We query COLM, the AEAD, in decryption direction, in our attack. An AEAD shouldn't release the plaintext without verifying the tag. Therefore, we mount a forgery against COLM as part of this attack.

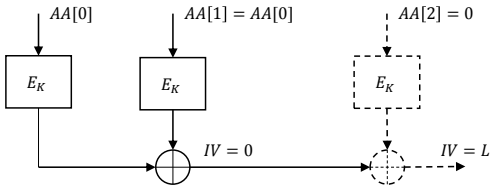

Figure 5.2 Setting $IV = 0$ or $IV = L$ ($IV = 0$ when $AD = AA[0]\|AA[1]$, and $IV = L$ when $AD = AA[0]\|AA[1]\|AA[2]$)

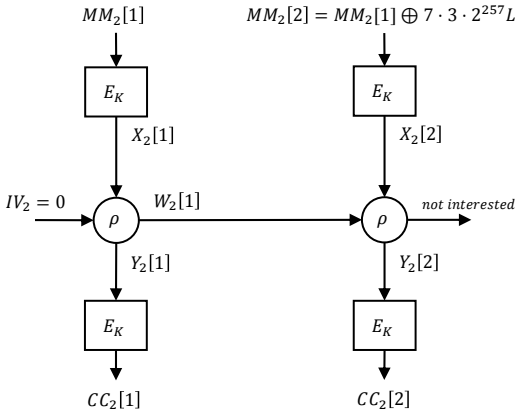

Figure 5.3 Generating Valid Tag

We begin our attack with the query of the COLM encryption twice as a precomputation to forge a valid tag for a chosen set of ciphertexts.

In the first query, the set of $\{E_K(\gamma), E_K(2 \cdot \gamma), \cdots, E_K(2^{127} \cdot \gamma)\}$ is constructed for an arbitrary non-zero γ where both γ and $D_K(3^{-1} \cdot \gamma)$ are known. We choose $\gamma = 3 \cdot L$ and set $IV_1 = L$. Then, encrypting $MM_1[i] = S$ for $i = 1, \ldots, 128$, we get $CC_1[i] = E_K(2^{i-1} \cdot 3 \cdot L)$, by Lemma 5.1 (see Figure 5.1). Taking $AA_1[1] = AA_1[0]$ and $AA_1[2] = 0$ yield to $IV_1 = L$ (see Figure 5.2). The process of constructing the set $\{E_K(3 \cdot L), E_K(2 \cdot 3 \cdot L), \cdots, E_K(2^{127} \cdot 3 \cdot L)\}$ is given in Algorithm 1.

Lemma 5.1. *Let* $AA[0] = AA[1]$ *and* $AA[2] = D_K(3^{-1}\gamma)$. *Then the corresponding* CT *of the* PT *where* $MM[i] = S$ *for* $i = 1, \ldots, 128$ *will result in* $CC[i] = E_K(2^{i-1} \cdot \gamma)$ *for* $i = 1, \ldots, 128$.

Proof. Let $AA[0] = AA[1]$ and $AA[2] = D_K(3^{-1}\gamma)$. Then $IV = W[0] = 3^{-1} \cdot \gamma$. For the PT if $MM[i] = S$ then $X[i] = 0$, for $i = 1, \ldots, 128$. Therefore, the inputs of the linear operator ρ will be $W[i-1] = 3^{-1} \cdot 2^{i-1} \cdot \gamma$ and $X[i] = 0$. So the output $Y[i]$ will be $2^{i-1} \cdot \gamma$, for $i = 1, \ldots, 128$. That is, $CC[i] = E_K(Y[i]) = E_K(2^{i-1} \cdot \gamma)$. □

Algorithm 1 Constructing the Set of $E_K(3 \cdot L)$, $E_K(2 \cdot 3 \cdot L)$, …, $E_K(2^{127} \cdot 3 \cdot L)$

procedure CONSTRUCT THE SET
  $IV \leftarrow L$
  for $i \leftarrow 1, 128$ **do**
    $MM[i] \leftarrow S$
  end for
  $(CT, tag) \leftarrow \mathcal{E}_K(AD, PT)$
  return CC ▷ $CC = CC[1]||CC[2]]|| \cdots ||CC[128]$
end procedure

We query another PT of at least 2 blocks during the offline phase to construct a tag which is valid for any 258-block CTs whose CC blocks appear even number of times. This query is depicted in Figure 5.3. The first two blocks of the resulting CT, which we denote ($CC_2[1]$, $CC_2[2]$), will be used for the verification according to Theorem 4.4 during the online phase.

In the decryption query, the COLM oracle is forged and $\beta = D_K(\alpha)$ for the given α is obtained. The explicit statement and the pseudocode of this process are given in Theorem 5.2 and in Algorithm 2, respectively.

Theorem 5.2. *Let $CC[1]$ and $CC[2]$ be the first two CT blocks of a PT with $MM[1] = MM[2] \oplus 7 \cdot 3 \cdot 2^{257} \cdot L$ encrypted with the 2-block AD where $AA[0] = AA[1]$. Assign $\alpha_d = (3 \cdot \gamma)^{-1} \cdot \alpha$ for a given value α. Let*

$$\widetilde{CC}[i] = \widetilde{CC}[i + 129] \quad = \quad \begin{cases} E_K(2^{128-i} \cdot \gamma) & \textit{if the } i^{th} \textit{ MSB} \\ & \textit{(Most Significant Bit) of } \alpha_d = 1, \\ L & \textit{otherwise}, \end{cases}$$

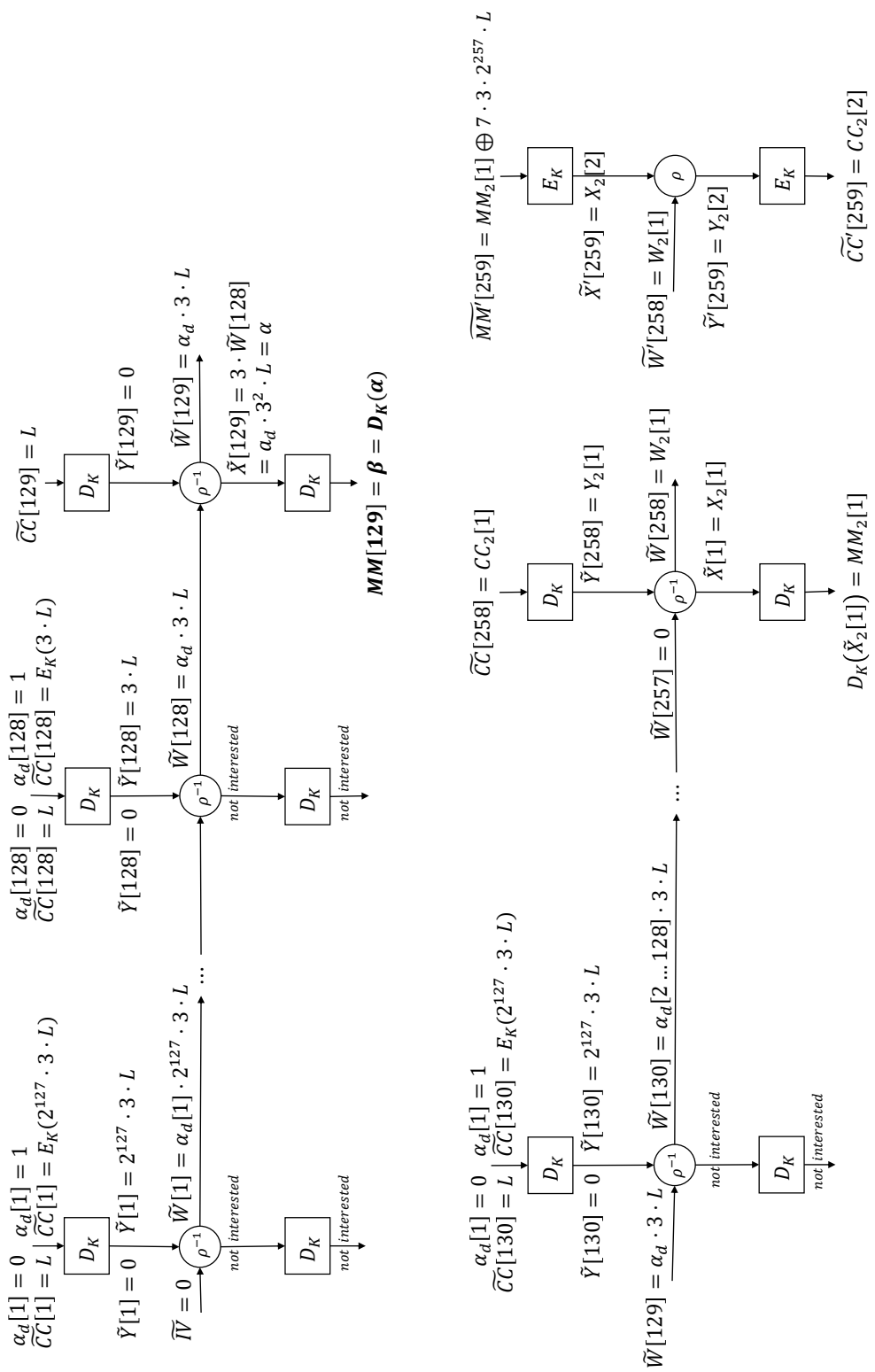

Figure 5.4 Forging COLM Algorithm for Simulating the Decryption Oracle of Underlying Block Cipher. The upper part depicts producing $\beta = D_K(\alpha)$. The lower part depicts verifying the tag.

for $i = 1, \ldots, 128$ *and* $\widetilde{CC}[129] = L$. *Let* $\widetilde{CC}[258] = CC[1]$ *and* $\widetilde{CC}[259] = CC[2]$. *Then, the* $\widetilde{CT} = \widetilde{CC}[1] \| \ldots \| \widetilde{CC}[259]$ *will be a valid ciphertext with the 2-block* AD *where* $AA[0] = AA[1]$ *and* $MM[129]$ *will be equal to* $D_K(\alpha)$.

Proof. Let a given plaintext PT be encrypted with the initial value $IV = 0$ and the first two blocks of the produced CT be $CC[1]$ and $CC[2]$. Assume $MM[1] \oplus MM[2] = 7 \cdot 3 \cdot 2^{257} \cdot L$. For any forged $\widetilde{CT}$ of 258 blocks whose $\widetilde{CC}[i]$ blocks take any value, except $L = E_K(0)$, even number of times for $1 \leq i < 258$, we have

$$\widetilde{W}[257] = \bigoplus_{i=1}^{128} \left(\widetilde{Y}[i] \oplus \widetilde{Y}[i+129] \right) \bigoplus D_K(L) \bigoplus IV = \bigoplus_{i=1}^{128} 0 \bigoplus 0 \bigoplus 0 = 0$$

by Lemma 4.2. By setting the last two blocks as $\widetilde{CC}[258] = CC[1]$ and $\widetilde{CC}[259] = CC[2]$, the tag is verified by Proposition 4.3.

The decision parameter $\alpha_d = (3 \cdot \gamma)^{-1} \cdot \alpha$ is calculated. Both $\widetilde{CC}[i]$ and $\widetilde{CC}[i+129]$ are set to $E_K(2^{128-i} \cdot \gamma)$, if i^{th} MSB of $\alpha_d = 1$, or set to L, otherwise and $\widetilde{CC}[129]$ is set to L.

$$\widetilde{W}[128] = \bigoplus_{i=1}^{128} \widetilde{Y}[i] \bigoplus IV = \bigoplus_{i=1}^{128} \alpha_d[i] \cdot 2^{128-i} \cdot \gamma = \alpha_d \cdot \gamma$$

and hence

$$\widetilde{X}[129] = 3 \cdot \widetilde{W}[128] \bigoplus \widetilde{Y}[129] = \alpha_d \cdot 3\gamma \bigoplus D_K(L) = \alpha$$

which yields $\widetilde{MM}[129] = \beta = D_K(\alpha)$ (see Figure 5.4). □

The online cost of this attack is a single COLM decryption query for an input consisting of a 2-block AD and a 259-block CT including the tag.

Algorithm 2 Forging COLM to Decrypt $D_K(\alpha)$

procedure $D_K(\alpha)$
  $\alpha_d \leftarrow (3^2 \cdot L)^{-1} \cdot \alpha$
  $\widetilde{AA}[1] \leftarrow \widetilde{AA}[0]$
  for $i \leftarrow 1, 128$ **do** ▷ 1 is the MSB, 128 is the LSB (Least Significant Bit)
    if $\alpha_d[i] = 1$ **then**
      $\widetilde{CC}[i] \leftarrow E_K(2^{128-i} \cdot 3 \cdot L)$
    else
      $\widetilde{CC}[i] \leftarrow L$
    end if
    $\widetilde{CC}[i + 129] \leftarrow \widetilde{CC}[i]$
  end for
  $\widetilde{CC}[129] \leftarrow L$
  $\widetilde{CC}[258] \leftarrow CC[1]$
  $\widetilde{CC}[259] \leftarrow CC[2]$
  $\widetilde{CC} = \widetilde{CC}[1]||\cdots||\widetilde{CC}[259]$
  $\widetilde{PT} \leftarrow \mathcal{D}_K(\widetilde{AD}, \widetilde{CT})$
  return $\widetilde{MM}[129]$ ▷ $\widetilde{MM} = \widetilde{MM}[1]|\widetilde{MM}[2]]|| \cdots ||\widetilde{MM}[258]$
end procedure

5.2. Simulation Model of the Encryption Oracle of the Underlying Block Cipher: Computing $\alpha = E_K(\beta)$

For a given β, we construct a convenient (AD, PT) such that the last CT block will be the required α value. The explicit statement is given in Theorem 5.3. We always choose AD so that $IV = 0$ independent of the given β. For example AD can be taken as two blocks such that $AA[0] = AA[1]$, then it is clear that $IV = 0$ (see Figure 5.2).

Theorem 5.3. *Let a PT be encrypted with an AD of two blocks such that $AA[1] = AA[0]$. Let $\beta_d = (3 \cdot L)^{-1} \cdot \beta$ for a given β. Take*

$$MM[i] = \begin{cases} S & \text{if the } i^{th} \text{ MSB of } \beta_d = 0, \\ 0 & \text{otherwise}, \end{cases}$$

for $i = 1, \ldots, 128$. Let $MM[129] = S$. Then the last block of the corresponding CT, $CC[129]$, equals $\alpha = E_K(\beta)$.

Proof. For any given β, we define the decision parameter $\beta_d = (3 \cdot L)^{-1} \cdot \beta$. Then, take $MM[i] = S$ if the i^{th} MSB of β_d is zero and $MM[i] = 0$, otherwise. This leads to

$$W[i] = 2 \cdot W[i-1] \oplus \beta_d[i] \cdot L.$$

Let $\beta_d = \beta_d[1]\beta_d[2] \cdots \beta_d[128]$ where $\beta_d[1]$ is the MSB and $\beta_d[128]$ is the LSB of β_d. Then,

$$W[128] = \beta_d \cdot L = \oplus_{i=1}^{128} \beta_d[i] \cdot 2^{128-i} \cdot L$$

as depicted in Figure 5.5. On the other hand, $X[129] = 0$ since $MM[129] = S$. Hence

$$Y[129] = 3 \cdot W[128] \oplus X[129] = 3 \cdot \beta_d \cdot L.$$

As a result,

$$CC[129] = E_K[\beta] = \alpha.$$

□

Algorithm 3 gives the pseudocode of how to construct the convenient PT for given β. Then querying $AD\|PT$, the last block of the corresponding CT will be α by Theorem 5.3 (see Figure 5.5). Let's remark that this is a different method for SEBC of COLM from Vaudenay's and Vizár's method in [12]. In [12], the authors precompute the series $E_K((2^n-1)L)$, where $0 \leq n \leq 134$ and arrange an arbitrary length AD to obtain $Y[1] = \beta$. In this attack, we precompute only $D_K(0)$ and arrange a 129-block PT to obtain $Y[129] = \beta$.

Algorithm 3 Computing $\alpha = E_K(\beta)$ for a given β

procedure $E_K(\beta)$
  $AA[1] \leftarrow AA[0]$
  $\beta_d \leftarrow (3 \cdot L)^{-1} \cdot \beta$
  for i $from$ 1 to 128 **do**
    if $\beta_d[i] = 0$ **then** ▷ 1 is the MSB, 128 is the LSB
      $MM[i] \leftarrow S$
    else
      $MM[i] \leftarrow 0$
    end if
  end for
  $MM[129] \leftarrow S$
  $(CT, tag) \leftarrow \mathcal{E}_K(AD, PT)$
  return $CC[129]$ ▷ $\alpha = E_K(\beta)$ is $CC[129]$
end procedure

The cost of this attack is a single COLM encryption query for an input consisting of a 2-block AD and a 259-block PT.

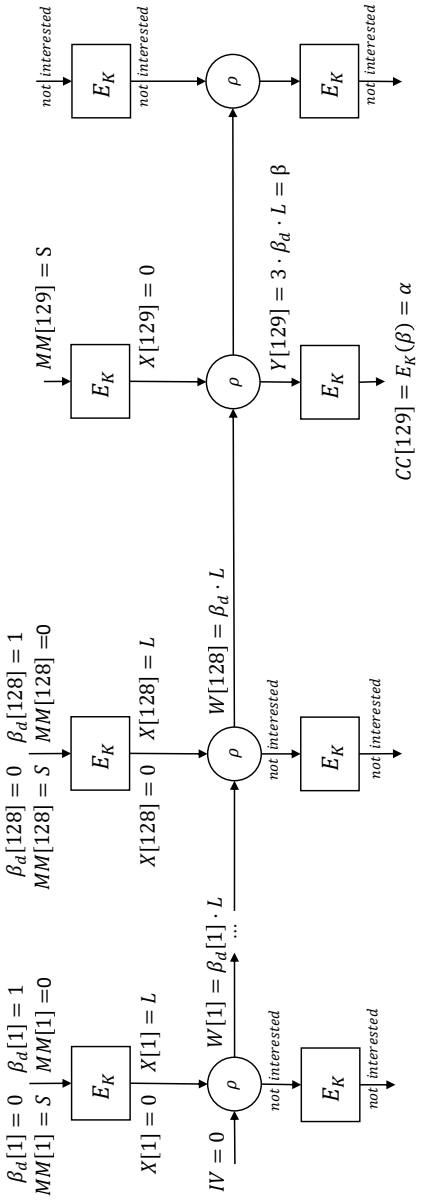

Figure 5.5 Simulation of the Encryption Oracle of the Underlying Block Cipher via COLM Oracle

6. Universal Forgery, Tag Guessing and Plaintext Recovery Attacks

In a universal forgery attack, the attacker is given any (AD, PT) pair and required to compute the corresponding CT and the tag. The attacker can query the encryption and decryption oracles of COLM for any data except for the given PT with any AD or the given AD with any PT.

Each ciphertext has two different tags in COLM, one for the full last block case and one for the incomplete last block case. In a tag guessing attack, the attacker is given any (AD, CT) pair and required to compute one of these two tags. The attacker can query any data except for the given CT with any AD or the given AD with any CT in the encryption and decryption oracles of COLM.

In a plaintext recovery attack, the attacker is given any (AD, CT) pair and required to compute the corresponding PT. The attacker can query the encryption and the decryption oracles of COLM for anything except for the given AD or CT even with different ciphertexts or associated data, respectively. Notice that the challenging pair (AD, CT) can be given with the valid tag in another version of plaintext recovery attacks. But we adopt the former definition in this section and in Section 7. We introduce also a plaintext recovery attack for the latter definition in Section 8.

All of the attacks above can be deduced easily from the SEBC and SDBC introduced in Section 5. An attacker who can run both SEBC and SDBC has the same capability as a legitimate user. Therefore, in such a scenario the attacker can produce the CT and its tag for any given PT and also decrypt any given CT even if its tag is missing. In the universal forgery attack, the state values are obtained by querying the MM blocks as in Section 5.2; on the other hand, in the tag guessing and plaintext recovery attacks the state values are obtained by querying the CC blocks as in Section 5.1.

In the universal forgery attack, first of all, the $MM[l]$ and the $MM[l+1]$ blocks are prepared as explained in Section 4. All Z, X and CC blocks in an encryption process of COLM are the outputs of the underlying block cipher encryption. Therefore, we use the SEBC to recover all Z, X and CC blocks from the AA, the MM, and the Y blocks respectively as in Section 5.2. The AA and the MM blocks are given while the Y blocks are computed via the linear combinations of the Z and the X blocks. Hence, we can produce the CT of any given PT by the SEBC. We query COLM $a + 2 \cdot l + 2$ times, where a is the number of the AD-blocks, and l is the number of the PT-blocks.

The tag guessing attack and the plaintext recovery attack are similar to our universal forgery attack. In both attacks, the SDBC introduced in Section 5.1 is also used and the AD-blocks are encrypted through the SEBC to obtain the Z blocks as explained in Section 5.2 and the IV is calculated via XORing the Z blocks. Similarly, the CT-blocks are decrypted through the SDBC to recover the Y-blocks. The W and the X-blocks are calculated via the inverse linear mix. All X blocks are decrypted through the SDBC to recover the PT-blocks. Furthermore, in the tag guessing attack, $MM'[l+1]$ is computed. It is $MM[l] \oplus 7 \cdot 3 \cdot 2^{l-1} \cdot L$ for the PTs whose last block is full. $CC'[l+1]$ is recovered by using $MM'[l+1]$ and $W[l]$ through the SEBC. Let's remark that we don't need the whole PT to obtain the length of its last block. We query COLM $a + l + 3$ times for the tag guessing and $a + 2 \cdot l + 2$ times for plaintext recovery, where a is the number of the AD blocks, and l is the number of the PT blocks.

7. Another Plaintext Recovery without Tag

In this section, we introduce another plaintext recovery attack where we can decrypt any ciphertext without its tag. In this attack, we use Lemma 4.5, Lemma 4.2, Proposition 4.3 and Theorem 4.4.

Let an (AD, CT) pair be a given ciphertext with its associated data. It is possible to mount an attack to decrypt this CT even if the tag is not known. Let

$$CT = CC[1] \| \dots \| CC[l].$$

Then, query a message of two blocks satisfying Equation 10 with the associated data AD:

$$\widehat{MM}[1] \oplus \widehat{MM}[2] \;\; = \;\; 7 \cdot 3 \cdot 2^{2l} \cdot L. \tag{10}$$

Let the first two blocks of the output be $\widehat{CC}[1]$ and $\widehat{CC}[2]$. To forge a $\widetilde{CT}$, set $\widetilde{CC}[2l+1] = \widehat{CC}[1]$, $\widetilde{CC}[2l+2] = \widehat{CC}[2]$ and choose the other $\widetilde{CC}[i]$'s such that

$$\widetilde{CC}[i] = \widetilde{CC}[i+l] = CC[i] \text{ for } 1 \leq i \leq l.$$

Observe that

$$\widetilde{W}[2l] = \bigoplus_{i=1}^{l} \left(D_K(\widetilde{CC}[i]) \bigoplus D_K(\widetilde{CC}[i+l]) \right) \bigoplus IV = IV. \tag{11}$$

Then the ($\widetilde{AD}$, $\widetilde{CC}$) pair will be an authenticated ciphertext with the tag $\widetilde{CC}[2l+2]$ (see Figure 7.1). Indeed, Equation 10 and Equation 11 together yield $\widehat{CC}[2] = \widetilde{CC'}[2l+2]$ by Lemma 4.2 and Proposition 4.3. Therefore, $\widetilde{CC}[2l+2] = \widehat{CC}[2]$ is a valid tag by Theorem 4.4.

It is clear that, the first l blocks of the plaintext of the forged ciphertext ($\widetilde{AD}$, $\widetilde{CT}$) will be the plaintext of the given ciphertext (AD, CT).

Note that $\widetilde{CC}[2l+2] = \widehat{CC}[2]$ is the valid tag for any forged ($\widetilde{AD}$, $\widetilde{CT}$) pair where $\widetilde{IV} = IV$ (see Lemma 4.5), $\widetilde{CC}[2l+1] = \widehat{CC}[1]$ and $\widetilde{CC}[i]$ blocks take any value, except $L = E_K(0)$, even number of times for $1 \leq i \leq 2l$. Therefore, the plaintext recovery attack can be converted to an existential forgery of $2l + 1$-block such $\widetilde{CT}$s (see Lemma 4.2).

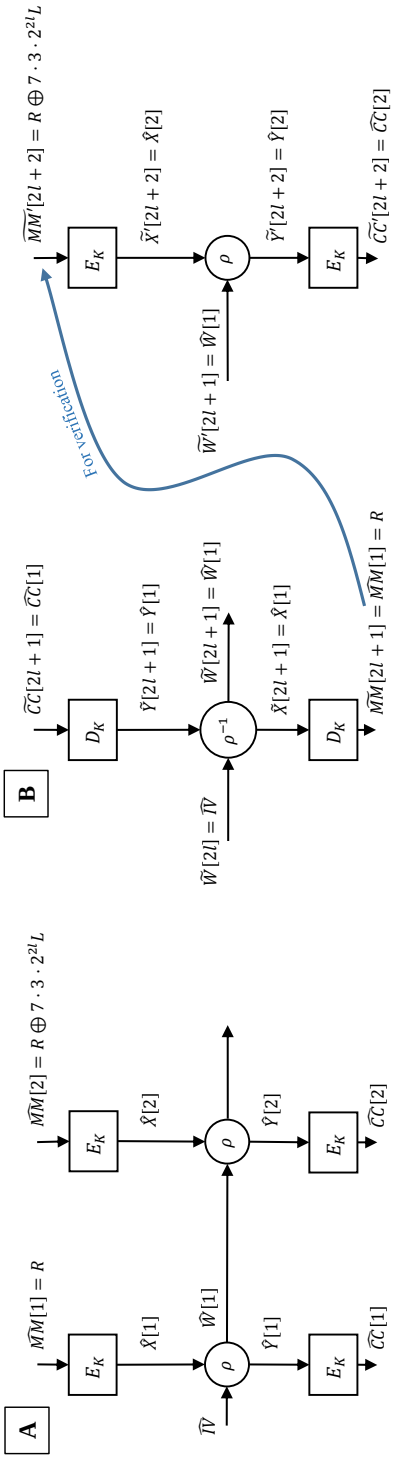

Figure 7.1 Plaintext Recovery Attack in Section 7. In Part A: Two PT blocks with a special difference are queried to obtain $\widehat{CC}[1]$ and $\widehat{CC}[2]$. In Part B: $\widetilde{CC}[2l+1] = \widehat{CC}[1]$ and $\widetilde{CC}[2l+2] = \widehat{CC}[2]$ are verified through the special difference of $\widehat{MM}[1]$ and $\widehat{MM}[2]$ depicted in Part A.

8. Permuting CT-Blocks

In this section, we introduce a new property for COLM. The CT blocks can be permuted without any effect on the tag value. This results from a property of the linear operator of COLM explained in Proposition 8.2. This property can be exploited to mount a plaintext recovery attack and a universal forgery attack. During the encryption, a state value $W[i]$ depends on all $X[j]$-blocks, $1 \leq j \leq i$, through the linear operation ρ. Hence $W[i]$ can be written as a linear combination of all $X[j]$s along with the IV, namely $W[i] = \bigoplus_{j=1}^{i} 2^{i-j} \cdot X[j] \bigoplus 2^i \cdot IV$. Observe that each $X[j]$ has a different coefficient in this combination. Therefore, reordering the $MM[j]$ blocks changes the state value $W[i]$. However, this is not true for the decryption process. Lemma 8.1 states this property of COLM.

Lemma 8.1. *For a given l-block ciphertext CT produced by COLM, the $\widetilde{CT}$ obtained by swapping any two blocks $CC[i]$ and $CC[j]$ for $i, j < l$ is still valid with the same tag $CC[l+1]$.*

Proof. Let a ciphertext $CT = CC[1]||\cdots||CC[l]$ with its tag $CC[l+1]$ be given with the initialization value IV. During the decryption, the state value $W[l]$ can be written as the accumulation of all the $Y[j]$ values where $1 \leq j \leq l$ as $W[l] = \bigoplus_{j=1}^{l} Y[j] \bigoplus IV$ since $W[k] = W[k-1] \oplus Y[k]$ for any $1 \leq k \leq l$. By swapping $CC[i]$ and $CC[j]$, we swap $Y[i]$ and $Y[j]$ for $i, j < l$. The new CT has the following Y values $\widetilde{Y}[k] = Y[k]$ for any $k \neq i, j$; $\widetilde{Y}[i] = Y[j]$ and $\widetilde{Y}[j] = Y[i]$. Hence,

$$\widetilde{W}[l-1] = \bigoplus_{j=1}^{l-1} \widetilde{Y}[j] \bigoplus IV = \bigoplus_{j=1}^{l-1} Y[j] \bigoplus IV = W[l-1]$$

and

$$\widetilde{W}[l] = \bigoplus_{j=1}^{l} \widetilde{Y}[j] \bigoplus IV = \bigoplus_{j=1}^{l} Y[j] \bigoplus IV = W[l].$$

Therefore, the checksum of the PT blocks $MM[l]$ remains same and hence the new ciphertext $\widetilde{CC}$ is valid with the tag $CC[l+1]$. □

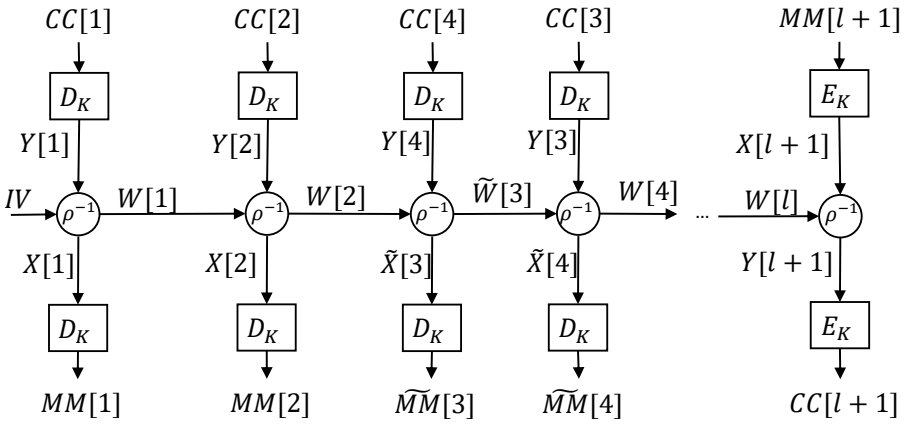

Figure 8.1 First Query to Recover PT

We can generalize the statement in Lemma 8.1 by applying arbitrary permutation instead of a swap operation.

Proposition 8.2. *For a given l-block ciphertext CT, the $\widetilde{CT}$ obtained by permuting the $CC[i]$ blocks for $i < l$ is still valid with the same tag $\widetilde{CC}[l+1] = CC[l+1]$.*

Proof. Let CT be a given ciphertext consists of $CC[1]\| \cdots \|CC[l]$ with its tag $CC[l+1]$ and the initialization value IV. Let the new ciphertext $\widetilde{CT}$ be obtained by permuting the $CC[i]$ blocks for $i < l$. On the other hand, any permutation can be written as the combination of two-cycles (swap operations). Hence, this new ciphertext $\widetilde{CT}$ can be obtained by swapping certain pairs of the CC blocks. However, after each swap the tag value remains same by Lemma 8.1. Therefore, the same tag $CC[l+1]$ is valid for $\widetilde{CT}$. □

Proposition 8.2 can be used to mount a plaintext recovery attack of any given (AD, CT) pair where $CT = CC[1]\|CC[2]\|CC[3]\|CC[4]\| \ldots \|tag$ of at least 5 blocks. One can recover the plaintext by querying

$$\mathcal{D}_K(AD, CC[1]\|CC[2]\|CC[4]\|CC[3]\| \ldots \|tag) \text{(See Figure 8.1)}$$
$$\mathcal{D}_K(AD, CC[2]\|CC[1]\|CC[3]\|CC[4]\| \ldots \|tag) \text{(See Figure 8.2)}$$

with an associated data AD and obtaining

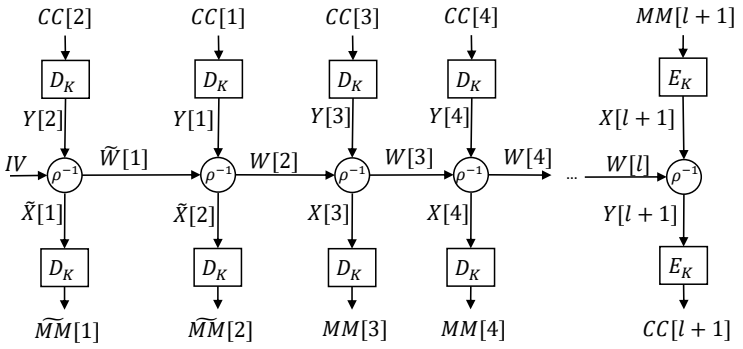

Figure 8.2 Second Query to Recover PT

$$MM[1]\|MM[2]\|\widetilde{MM}[3]\|\widetilde{MM}[4]\|\dots$$
$$\widetilde{MM}[1]\|\widetilde{MM}[2]\|MM[3]\|MM[4]\|\dots$$

respectively and then combining the first two blocks of the former plaintext with the rest of the latter plaintext.

Proposition 8.2 can be used also to produce the CT and the tag of a given $PT = MM[1]\|MM[2]\|MM[3]\|\dots$ by querying the oracle as follows:

$$CC[1]\|CC[2]\|\widetilde{CC}[3]\|\widetilde{tag}, \leftarrow \mathcal{E}_K(AD, MM[1]\|MM[2]\|\widetilde{MM}[3])$$
$$\widetilde{MM}[1]\|\widetilde{MM}[2]\|\widetilde{MM}[3], \leftarrow \mathcal{D}_K(AD, CC[2]\|CC[1]\|\widetilde{CC}[3]\|\widetilde{tag})$$
$$CC[2]\|CC[1]\|CC[3]\|\dots\|tag. \leftarrow \mathcal{E}_K(AD, \widetilde{MM}[1]\|\widetilde{MM}[2]\|MM[3]\|\dots)$$

with an associated data AD. For illustration of these queries refer to Figure 8.3, Figure 8.4, Figure 8.5, respectively.

The third block of the target message is modified to be an arbitrary $\widetilde{MM}[3]$ value and the new message is encrypted to $CC[1]\|CC[2]\|\widetilde{CC}[3]\|\widetilde{tag}$, whose first two blocks will be the original CT blocks. The oracle decrypts this CT when we swap the first two blocks since the tag is still valid by Lemma 8.1. The first two $\widetilde{MM}$ blocks will result in random values ($\widetilde{MM}[1]$ and $\widetilde{MM}[2]$) but observe that $\widetilde{W}[2]$ will be equal to $W[2]$. Therefore, querying this $\widetilde{MM}[1]\|\widetilde{MM}[2]\|MM[3]\|\dots$ leads to $CC[2]\|CC[1]\|CC[3]\|\dots\|tag$ from which the original CT can be obtained by just swapping the first two blocks.

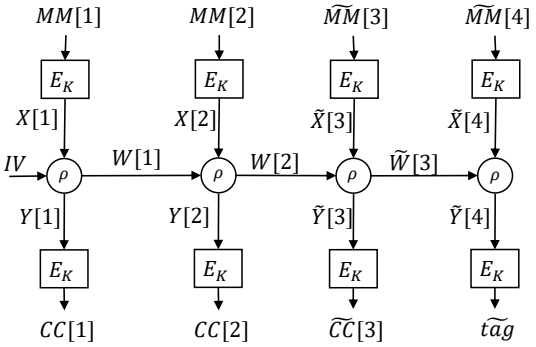

Figure 8.3 First Query to Produce CT

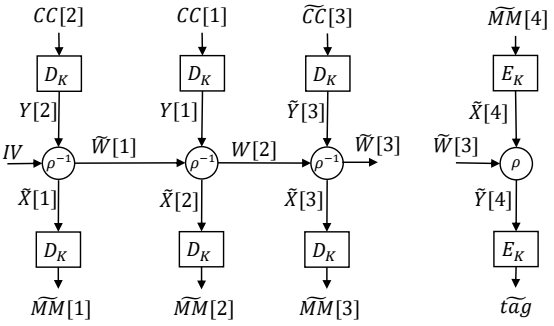

Figure 8.4 Second Query to Produce CT

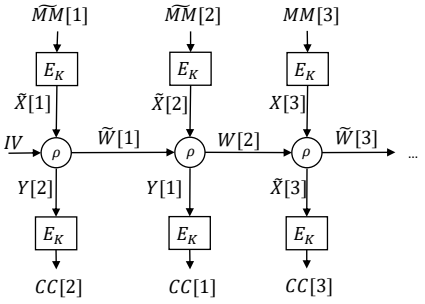

Figure 8.5 Third Query to Produce CT

$MM[l]$, $MM[l+1]$ and $W[l-1]$ parameters are used for the tag verification. Permuting the first $(l-1)$ CT-blocks will not affect these parameters by Proposition 8.2, resulting in $(l-1)!$ existential forgeries.

Table 9.1 Some Key Recovery Attacks against AES-128. MitM: Meet-in-the-Middle, ID: Impossible Differential

Attack	Rounds	Data	Query	Time	Reference
MitM	7	2^{97} CPA	2^{105}	2^{99}	[53]
ID	7	2^{105} CPA	2^{113}	$2^{106.88}$	[54]
Biclique	10	2^{88} CCA	2^{97}	$2^{126.18}$	[55]

9. Key Recovery

In this Section, we introduce how to use COLM oracle to collect necessary data for mounting attacks against (reduced-round) AES-128.

There are several attacks against reduced-round AES-128. In general, these attacks require large amount of data, as either CPA or CCA. The question is how to collect data for an AES key used in the COLM encryption. In principle, any weakness of AES isn't supposed to be exploited in COLM oracle even in nonce misuse scenario since COLM is chosen to the defense-in-depth final portfolio of the CAESAR competition. However, as explained in Section 5, we can simulate AES encryption and decryption oracles for any given input, enabling us to collect data required for an attack against (reduced-round) AES-128.

We give three examples of combining these oracles (see Section 5) with published CPA/CCA attacks against reduced-round AES-128, a meet-in-the-middle (MitM) attack [53], an impossible differential (ID) attack [54] and a biclique attack [55]. The baseline complexities of these published attacks are listed in Table 9.1. In Section 5.2, we show that a single COLM query of a 132-block AD-PT is enough to get the AES encryption of the any given input. Therefore, to collect the data for MitM and ID attacks, we need to make $262 \times 2^{97} \approx 2^{105}$ and $262 \times 2^{105} \approx 2^{113}$ AES-128 queries in COLM, respectively. In Section 5.1, we show that a single COLM query of a 261-block AD-CT is enough to get the AES decryption of any given input. Therefore, to collect the data for biclique attack, we make $520 \times 2^{88} \approx 2^{97}$ AES-128 queries in COLM. Recall that we need 2^{130} AES-128 encryptions ($L = E_K(0)$, $Z[0] = E_K(AA[0])$, $X[1] = E_K(MM[1])$, $CC[1] = E_K(Y[1])$ for each key candidate) to

mount a brute force attack against COLM. Therefore, these attacks compared to brute force attacks are 4 times more time efficient in COLM oracle scenario than in AES oracle scenario.

10. Conclusions and Suggestions

In this thesis, we showed that COLM is as secure as the secrecy of its whitening parameter $L = E_K(0)$. We mounted plaintext recovery attacks, a tag guessing attack, several forgery attacks, and successfully built an SEBC and an SDBC with polynomial time complexity once the L parameter is known. To the best of our knowledge, this thesis introduces the first simulation model where an EME/EMD type AEAD is queried in the decryption direction, enabling the building of both an SEBC and an SDBC together. It should be noted that an attacker in possession of both an SEBC and an SDBC can generate any plaintext or ciphertext like a legitimate user with key access. Generalizing our methods for mounting plaintext recovery and tag guessing attacks to an arbitrarily formed authenticated EME/EMD scheme remains an open problem.

EME algorithms have generally a whitening mask L defined by designers. In authenticated encryption, algorithms are supposed to have the security level of $\min(|key|, |block|)$ against some attacks such as plaintext recovery and tag guessing. Both the key and the block sizes are 128 bits in COLM. So, the security level of recovering L or the EME structure must be at least 128-bit. Forler et al. illustrate that L can be recovered in 2^{65} queries [37]. In this thesis, we show that the EME structure in COLM doesn't satisfy 128-bit security against plaintext recovery and tag guessing attacks. That is, neither the confidentiality of L nor EME of COLM satisfies 128-bit security.

We recommend that both recovering L and the EME structure should have the security level of $\min(|key|, |block|)$ for an authenticated EME scheme designed in defense-in-depth approach. Accordingly, if one of them fails to provide this security level, the authenticated EME scheme still remains secure.

We suggest some modifications for COLM to prevent our attacks. We propose a simple but unconventional suggestion to fulfill the security requirement for recovering L, by producing it with another key deduced from the main key by a slight modification. Even if L is recovered (note that Lu's method is still valid), it can't be exploited in our attacks in

Algorithm 3, Algorithm 1 and Algorithm 2 since L is produced by a different key and it does not give a plaintext-ciphertext pair for the underlying block cipher. Our next suggestion is to modify the ρ-operation. Note that $W[i] = W[i-1] \oplus Y[i]$ in the ρ^{-1} operation and we exploit this property in Lemma 4.2, Lemma 8.1, Proposition 8.2, Theorem 4.1, Theorem 4.4, Theorem 5.3 and Algorithm 2. Hence, in the computation of $W[i]$ the coefficients of $W[i-1]$ inputs shouldn't be 1 in both ρ and ρ^{-1}. This criterion should also be adopted for $W'[i]$. As an example; $W'[i] = 3 \cdot W'[i-1] \oplus A[i]$ for processing AD, and $W[i] = 4 \cdot W[i-1] \oplus X[i]$, $Y[i] = 6 \cdot W[i-1] \oplus X[i]$ for processing PT (resulting in $W[i] = 2 \cdot W[i-1] \oplus Y[i]$, $X[i] = 6 \cdot W[i-1] \oplus Y[i]$) can be used. Our last suggestion is to use different coefficients for the M blocks in the summation to compute the $M[l]$ and the $M[l+1]$ blocks so that one cannot isolate the last two blocks when verifying the tag. This suggestion will render Proposition 4.3, Proposition 8.2, Theorem 4.1, Theorem 4.4 and Theorem 5.2 invalid. Lastly, these suggestions are to prevent our attacks and COLM is still required to be analyzed with these amendments.

REFERENCES

[1] Lilian Bossuet, Cuauhtemoc Mancillas-López, and Brisbane Ovilla-Martínez. Pipelined hardware implementation of copa, elmd, and colm. *IEEE Transactions on Computers*, 69(10):1533–1543, **2020**. doi:10.1109/TC.2020.2977031.

[2] Elaine Barker and Nicky Mouha. Recommendation for the triple data encryption algorithm (tdea) block cipher. Technical report, Gaithersburg, MD, United States, **2017**.

[3] National Institute of Standards and Technology. Advanced encryption standard. *NIST FIPS PUB 197*, **2001**.

[4] Phillip Rogaway. Authenticated-encryption with associated-data. In *Proceedings of the 9th ACM Conference on Computer and Communications Security*, pages 98–107. **2002**.

[5] Lawrence Bassham, Çağdaş Çalık, Kerry McKay, and Meltem Sönmez Turan. Submission requirements and evaluation criteria for the lightweight cryptography standardization process. *US National Institute of Standards and Technology*, **2018**.

[6] Morris Dworkin. Sha-3 standard: Permutation-based hash and extendable-output functions, **2015**. doi:https://doi.org/10.6028/NIST.FIPS.202.

[7] Cryptographic competitions.

[8] Elena Andreeva, Andrey Bogdanov, Nilanjan Datta, Atul Luykx, Bart Mennink, Mridul Nandi, Elmar Tischhauser, and Kan Yasuda. COLM v1, **2016**. `https://competitions.cr.yp.to/round3/colmv1.pdf`.

[9] L. Bossuet, N. Datta, C. Mancillas-López, and M. Nandi. ELmD: A pipelineable authenticated encryption and its hardware implementation. *IEEE Transactions on Computers*, 65(11):3318–3331, **2016**. doi:10.1109/TC.2016.2529618.

[10] Elena Andreeva, Andrey Bogdanov, Atul Luykx, Bart Mennink, Elmar Tischhauser, and Kan Yasuda. Parallelizable and authenticated online ciphers. Cryptology ePrint Archive, Report 2013/790, **2013**. https://eprint.iacr.org/2013/790.

[11] Asli Bay, Oguzhan Ersoy, and Ferhat Karakoç. Universal forgery and key recovery attacks on elmd authenticated encryption algorithm. In Jung Hee Cheon and Tsuyoshi Takagi, editors, *Advances in Cryptology - ASIACRYPT 2016 - 22nd International Conference on the Theory and Application of Cryptology and Information Security, Hanoi, Vietnam, December 4-8, 2016, Proceedings, Part I*, volume 10031 of *Lecture Notes in Computer Science*, pages 354–368. **2016**. doi:10.1007/978-3-662-53887-6_13.

[12] Serge Vaudenay and Damian Vizár. Can caesar beat galois? - robustness of CAESAR candidates against nonce reusing and high data complexity attacks. In Bart Preneel and Frederik Vercauteren, editors, *Applied Cryptography and Network Security - 16th International Conference, ACNS 2018, Leuven, Belgium, July 2-4, 2018, Proceedings*, volume 10892 of *Lecture Notes in Computer Science*, pages 476–494. Springer, **2018**. doi:10.1007/978-3-319-93387-0_25.

[13] Kerckhoffs Auguste. La cryptographie militaire. *Journal des sciences militaires*, IX:5–38, **1883**.

[14] Claude Shannon. *A Mathematical Theory of Cryptography*. **1945**.

[15] E. N. Gilbert, F. J. MacWilliams, and N. J. A. Sloane. Codes which detect deception. *Bell System Technical Journal*, 53(3):405–424, **1974**. doi:https://doi.org/10.1002/j.1538-7305.1974.tb02751.x.

[16] Roger M. Needham and Michael D. Schroeder. Using encryption for authentication in large networks of computers. *Commun. ACM*, 21(12):993–999, **1978**. ISSN 0001-0782. doi:10.1145/359657.359659.

[17] W. Diffie and M.E. Hellman. Privacy and authentication: An introduction to cryptography. *Proceedings of the IEEE*, 67(3):397–427, **1979**. doi:10.1109/PROC.1979.11256.

[18] Gustavus J. Simmons. Authentication theory/coding theory. In George Robert Blakley and David Chaum, editors, *Advances in Cryptology*, pages 411–431. Springer Berlin Heidelberg, Berlin, Heidelberg, **1985**.

[19] W. Diffie and M. Hellman. New directions in cryptography. *IEEE Trans. Inf. Theor.*, 22(6):644–654, **2006**. ISSN 0018-9448. doi:10.1109/TIT.1976.1055638.

[20] R. L. Rivest, A. Shamir, and L. Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Commun. ACM*, 21(2):120–126, **1978**. ISSN 0001-0782. doi:10.1145/359340.359342.

[21] Geoffroy Couteau. *Zero-knowledge proofs for secure computation*. Theses, Université Paris sciences et lettres, **2017**.

[22] Henk CA Van Tilborg and Sushil Jajodia. *Encyclopedia of cryptography and security*. Springer Science & Business Media, **2014**.

[23] Eric Rescorla. The Transport Layer Security (TLS) Protocol Version 1.3. RFC 8446, **2018**. doi:10.17487/RFC8446.

[24] Chris M. Lonvick and Tatu Ylonen. The Secure Shell (SSH) Transport Layer Protocol. RFC 4253, **2006**. doi:10.17487/RFC4253.

[25] Sheila Frankel and Suresh Krishnan. IP Security (IPsec) and Internet Key Exchange (IKE) Document Roadmap. RFC 6071, **2011**. doi:10.17487/RFC6071.

[26] Nadhem J Al Fardan and Kenneth G Paterson. Lucky thirteen: Breaking the tls and dtls record protocols. In *2013 IEEE symposium on security and privacy*, pages 526–540. IEEE, **2013**.

[27] Announcement of the CAESAR finalists. https://cr.yp.to/talks/2018.03.05/slides-djb-20180305-caesar-4x3.pdf. Accessed: 2020-05-25.

[28] Mihir Bellare and Chanathip Namprempre. Authenticated encryption: Relations among notions and analysis of the generic composition paradigm. *Journal of cryptology*, 21(4):469–491, **2008**.

[29] Tetsu Iwata. New blockcipher modes of operation with beyond the birthday bound security. In *Fast Software Encryption: 13th International Workshop, FSE 2006, Graz, Austria, March 15-17, 2006, Revised Selected Papers 13*, pages 310–327. Springer, **2006**.

[30] Phillip Rogaway and Thomas Shrimpton. A provable-security treatment of the key-wrap problem. In Serge Vaudenay, editor, *Advances in Cryptology - EUROCRYPT 2006*, pages 373–390. Springer Berlin Heidelberg, Berlin, Heidelberg, **2006**. ISBN 978-3-540-34547-3.

[31] Jean-Philippe Aumasson, Philipp Jovanovic, and Samuel Neves. Norx: Parallel and scalable aead. In Mirosław Kutyłowski and Jaideep Vaidya, editors, *Computer Security - ESORICS 2014*, pages 19–36. Springer International Publishing, Cham, **2014**. ISBN 978-3-319-11212-1.

[32] Viet Tung Hoang, Jonathan Katz, and Alex J Malozemoff. Automated analysis and synthesis of authenticated encryption schemes. In *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*, pages 84–95. **2015**.

[33] Lei Zhang, Susanna Spinsante, Chaojing Tang, and Ennio Gambi. Application and performance analysis of various aead techniques for space telecommand authentication. *IEEE Transactions on Wireless Communications*, 8(1):308–319, **2009**.

[34] Idris Ahmed, Anne James, and Dhananjay Singh. Critical analysis of counter mode with cipher block chain message authentication mode protocol—ccmp. *Security and communication Networks*, 7(2):293–308, **2014**.

[35] David A. McGrew and John Viega. The security and performance of the galois/counter mode of operation (full version). Cryptology ePrint Archive, Paper 2004/193, **2004**. `https://eprint.iacr.org/2004/193`.

[36] Jiqiang Lu. Almost universal forgery attacks on the copa and marble authenticated encryption algorithms. In *Proceedings of the 2017 ACM on Asia Conference on Computer and Communications Security*, ASIA CCS '17, page 789–799. Association for Computing Machinery, New York, NY, USA, **2017**. ISBN 9781450349444. doi:10.1145/3052973.3052981.

[37] Christian Forler, Eik List, Stefan Lucks, and Jakob Wenzel. Reforgeability of authenticated encryption schemes. In Josef Pieprzyk and Suriadi Suriadi, editors, *Information Security and Privacy - 22nd Australasian Conference, ACISP 2017, Auckland, New Zealand, July 3-5, 2017, Proceedings, Part II*, volume 10343 of *Lecture Notes in Computer Science*, pages 19–37. Springer, **2017**. doi:10.1007/978-3-319-59870-3_2.

[38] Nilanjan Datta, Atul Luykx, Bart Mennink, and Mridul Nandi. Understanding RUP integrity of COLM. *IACR Trans. Symmetric Cryptol.*, 2017(2):143–161, **2017**. doi:10.13154/tosc.v2017.i2.143-161.

[39] M. Gruber, M. Probst, and M. Tempelmeier. Persistent fault analysis of OCB, DEOXYS and COLM. In *2019 Workshop on Fault Diagnosis and Tolerance in Cryptography (FDTC)*, pages 17–24. **2019**.

[40] M. Khairallah, S. Bhasin, and A. Chattopadhyay. On misuse of nonce-misuse resistance : Adapting differential fault attacks on (few) CAESAR winners. In *2019 IEEE 8th International Workshop on Advances in Sensors and Interfaces (IWASI)*, pages 189–193. **2019**.

[41] Yu Sasaki. Improved related-tweakey boomerang attacks on deoxys-bc. In Antoine Joux, Abderrahmane Nitaj, and Tajjeeddine Rachidi, editors, *Progress in Cryptology - AFRICACRYPT 2018 - 10th International Conference on Cryptology in Africa, Marrakesh, Morocco, May 7-9, 2018, Proceedings*, volume 10831 of *Lecture Notes in Computer Science*, pages 87–106. Springer, **2018**. doi:10.1007/978-3-319-89339-6_6.

[42] F. Moazami, A.R. Mehrdad, and H. Soleimany. Impossible differential cryptanalysis on Deoxys-BC-256. *The ISC International Journal of Information Security*, 10(2):93–105, **2018**. ISSN 2008-2045. doi:10.22042/isecure.2018.114245.405.

[43] Maria Eichlseder, Marcel Nageler, and Robert Primas. Analyzing the linear keystream biases in AEGIS. *IACR Transactions on Symmetric Cryptology*, 2019(4):348–368, **2020**. doi:10.13154/tosc.v2019.i4.348-368.

[44] Tomer Ashur, Maria Eichlseder, Martin M. Lauridsen, Gaëtan Leurent, Brice Minaud, Yann Rotella, Yu Sasaki, and Benoît Viguier. Cryptanalysis of MORUS. In Thomas Peyrin and Steven D. Galbraith, editors, *Advances in Cryptology - ASIACRYPT 2018 - 24th International Conference on the Theory and Application of Cryptology and Information Security, Brisbane, QLD, Australia, December 2-6, 2018, Proceedings, Part II*, volume 11273 of *Lecture Notes in Computer Science*, pages 35–64. Springer, **2018**. doi:10.1007/978-3-030-03329-3_2.

[45] Prakash Dey, Raghvendra Singh Rohit, and Avishek Adhikari. Full key recovery of acorn with a single fault. *Journal of Information Security and Applications*, 29:57–64, **2016**. ISSN 2214-2126. doi:https://doi.org/10.1016/j.jisa.2016.03.003.

[46] M. Jahanbani, Z. Norozi, and N. Bagheri. DPA protected implementation of OCB and COLM authenticated ciphers. *IEEE Access*, 7:139815–139826, **2019**.

[47] M. Tempelmeier, F. De Santis, G. Sigl, and J. Kaps. The CAESAR-API in the real world — towards a fair evaluation of hardware CAESAR candidates. In *2018 IEEE International Symposium on Hardware Oriented Security and Trust (HOST)*, pages 73–80. **2018**.

[48] M. Katsaiti and N. Sklavos. Implementation efficiency and alternations, on CAESAR finalists: AEGIS approach. In *2018 IEEE 16th Intl Conf on Dependable, Autonomic and Secure Computing, 16th Intl Conf on Pervasive Intelligence and Computing, 4th Intl Conf on Big Data Intelligence and Computing and Cyber Science and Technology Congress(DASC/PiCom/DataCom/CyberSciTech)*, pages 661–665. **2018**.

[49] A. Abbas, H. Mostafa, and A. N. Mohieldin. Low area and low power implementation for CAESAR authenticated ciphers. In *2018 New Generation of CAS (NGCAS)*, pages 49–52. **2018**.

[50] F. Farahmand, W. Diehl, A. Abdulgadir, J. Kaps, and K. Gaj. Improved lightweight implementations of CAESAR authenticated ciphers. In *2018 IEEE 26th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*, pages 29–36. **2018**.

[51] Dan Boneh, Richard Demillo, and Richard Lipton. On the importance of eliminating errors in cryptographic computations. *Journal of Cryptology*, 14, **1999**. doi:10.1007/s001450010016.

[52] Fan Zhang, Xiaoxuan Lou, Xinjie Zhao, Shivam Bhasin, Wei He, Ruyi Ding, Samiya Qureshi, and Kui Ren. Persistent fault analysis on block ciphers. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, 2018(3):150–172, **2018**. doi:10.13154/tches.v2018.i3.150-172.

[53] 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**. doi:10.1007/978-3-642-38348-9_23.

[54] Christina Boura, Virginie Lallemand, María Naya-Plasencia, and Valentin Suder. Making the impossible possible. *J. Cryptology*, 31:101–133, **2018**. doi:10.1007/s00145-016-9251-7.

[55] 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**. doi:10.1007/978-3-642-25385-0_19.