MARCOS ANTONIO SIMPLICIO JUNIOR

MESSAGE AUTHENTICATION ALGORITHMS FOR WIRELESS SENSOR NETWORKS

ALGORITMOS DE AUTENTICAÇÃO DE MENSAGENS PARA REDES DE SENSORES

Tese apresentada à Escola Politécnica da Universidade de São Paulo para obtenção do Título de Doutor em Engenharia Elétrica.

São Paulo
2010

MARCOS ANTONIO SIMPLICIO JUNIOR

MESSAGE AUTHENTICATION ALGORITHMS FOR WIRELESS SENSOR NETWORKS

ALGORITMOS DE AUTENTICAÇÃO DE MENSAGENS PARA REDES DE SENSORES

Tese apresentada à Escola Politécnica da Universidade de São Paulo para obtenção do Título de Doutor em Engenharia Elétrica.

Área de Concentração:

Sistemas Digitais

Orientador:

Paulo Sérgio Licciardi M. Barreto

São Paulo
2010

AGRADECIMENTOS

Primeiramente, agradeço a duas mulheres muito importantes na minha vida: minha mãe, Maria Suely M. Simplício, e à minha quase-esposa, Débora Arantes (Simplício). A vocês o meu muito obrigado por continuarem me aturando e apoiando em todos os momentos, mesmo que muitas vezes tenham se perguntado quando é que eu iria terminar de vez a faculdade ☺

Além disto, agradeço imensamente ao meu orientador e amigo Paulo S. L. M. Barreto, sempre disponível para longas conversas sobre segurança e criptografia (ou simplesmente sobre Jornada nas Estrelas, tipos de cerveja, ou qualquer outro assunto que viesse à mente), as quais costumavam terminar com um "E então, já podemos marcar a defesa?". Muito obrigado pelo apoio, pela amizade, e pelo exemplo.

Também não posso deixar de agradecer ao FDTE (Fundação Para o Desenvolvimento Tecnológico da Engenharia) e à Ericsson pelo auxílio financeiro, à Prof. Tereza Carvalho, amiga e coordenadora de diversos projetos no LARC dos quais participei, e à toda a equipe do LARC e da Ericsson (em especial ao Mats Näslund) pelas inspiradas e bem humoradas discussões.

Meus sinceros agradecimentos a todos vocês.

~ · ~ · ~ · ~ · ~ · ~ · ~ · ~ · ~

First, I need to express my gratitude to two women with a very important role in my life: my mother, Maria Suely M. Simplício, and my almost-wife, Débora Arantes (Simplício). Thank you for supporting and bearing with me in all moments, even when asking themselves when (or if) I would finally finish my studies ☺

Moreover, I am also immensely grateful to my advisor and friend Paulo S. L. M. Barreto, always available for a long chat about security and cryptography – or simply about Star Trek, types of beer, or any other subject crossing the mind – which would usually end with the phrase "so, let's choose the date for your defense?". Thank you very much for the support, friendship and example.

Finally, I would like to express my gratitude to the FDTE (Fundação Para o Desenvolvimento Tecnológico da Engenharia – Foundation for the Technological Development of the Engineering) and Ericsson for sponsoring my research, to Prof. Tereza Carvalho, friend and coordinator in the projects I have participated at LARC, and to all LARC and Ericsson members (especially Mats Näslund) for the inspired and pleasant conversations.

I am sincerely grateful to all of you.

RESUMO

Prover segurança às informações trafegadas nos mais diversos tipos de redes é algo essencial. Entretanto, redes altamente dependentes de dispositivos com recursos limitados (como sensores, *tokens* e *smart cards*) apresentam um desafio importante: a reduzida disponibilidade de memória e energia destes dispositivos, bem como sua baixa capacidade de processamento, dificultam a utilização de diversos algoritmos criptográficos considerados seguros atualmente. Este é o caso não apenas de cifras simétricas, que proveem confidencialidade aos dados, mas também de MACs (*Message Authentication Code*, ou "Código de Autenticação de Mensagem"), que garantem sua integridade e autenticidade. De fato, algumas propostas recentes de cifras de bloco dedicadas a plataformas limitadas (e.g., o CURUPIRA-2) proveem segurança e desempenho em um nível mais adequado a este tipo de cenário do que soluções tradicionais. Seguindo uma linha semelhante, o presente trabalho concentra-se no projeto e análise MACs leves e seguros voltados a cenários com recursos limitados, com foco especial em redes de sensores sem fio (RSSF).

MARVIN é o nome do algoritmo de MAC proposto neste trabalho. MARVIN adota a estrutura ALRED, que reutiliza porções de código de uma cifra de bloco subjacente e, assim, introduz um reduzido impacto em termos de ocupação de memória. Este algoritmo apresenta uma estrutura bastante flexível e é altamente paralelizável, permitindo diversas otimizações em função dos recursos disponíveis na plataforma alvo. Como vantagem adicional, MARVIN pode ser usado tanto em cenários que necessitam apenas da autenticação de mensagens quanto em esquemas de AEAD (*Authenticated-Encryption with Associated Data*, ou "Encriptação Autenticada com Dados Associados"), que aliam encriptação e autenticação. O esquema de AEAD proposto neste trabalho, denominado LETTERSOUP, explora as características da estrutura do MARVIN e adota uma cifra de bloco operando no modo LFSRC (*Linear Feedback Shift Register Counter*, ou "Contador-Registrador de Deslocamento Linear com Retroalimentação"). Além da especificação de ambos os algoritmos, este documento apresenta uma análise detalhada da segurança e desempenho dos mesmos em alguns cenários representativos.

Palavras-chave: Criptografia. Redes de Sensores Sem Fio. Redes com recursos limitados. Códigos de Autenticação de Mensagens (MACs). Encriptação Autenticada com Dados Associados (AEAD).

ABSTRACT

Security is an important concern in any modern network. However, networks that are highly dependent on constrained devices (such as sensors, tokens and smart cards) impose a difficult challenge: their reduced availability of memory, processing power and (specially) energy hinders the deployment of many modern cryptographic algorithms known to be secure. This inconvenience affects not only the deployment of symmetric ciphers, which provide data confidentiality, but also Message Authentication Codes (MACs), used to attest the messages' integrity and authenticity. Due to the existence of dedicated block ciphers whose performance and security are adequate for use in resource-constrained scenarios (e.g., the CURUPIRA-2), the focus of this document is on the design and analysis of message authentication algorithms. Our goal is to develop a secure and lightweight solution for deployment on resource constrained scenarios, with especial focus on Wireless Sensor Networks (WSNs).

MARVIN is the name of the MAC algorithm proposed in this document. MARVIN adopts the ALRED structure, allowing it to reuse parts of an underlying block cipher machinery; as a result, MARVIN's implementation builds on the cipher's efficiency and introduces little impact in terms of memory occupation. Moreover, this algorithm presents a flexible and highly parallelizable structure, allowing many implementation optimizations depending on the resources available on the target platform. MARVIN can be used not only as an authentication-only function, but also in an Authenticated-Encryption with Associated Data (AEAD) scheme, combining authentication and encryption. In this document, we define a new AEAD proposal called LETTERSOUP, which is based on the LFSRC (Linear Feedback Shift Register Counter) mode of operation and builds on MARVIN. Together with the specification of both algorithms, we provide a detailed security analysis and evaluate their performance in some representative scenarios.

Keywords: Cryptography. Wireless Sensor Networks. Constrained Platforms. Message Authentication Codes (MACs). Authenticated-Encryption with Associated Data (AEAD).

RÉSUMÉ

Assurer la sécurité des données est un sujet de grande importance dans les réseaux modernes. Toutefois, réseaux fortement dépendants de dispositifs de capacité réduite (comme des senseurs, badges et cartes à puce) impose un grand défi: leurs limitations en termes de disponibilité de mémoire, vitesse du traitement des données et (surtout) énergie entrave l'utilisation de plusieurs solutions cryptographiques modernes. Cette difficulté ne concerne pas seulement les algorithmes de chiffrement, lesquels assure la confidentialité des données, mais aussi les MACs (*Message Authentication Codes* ou "Codes d'Authentification de Messages"), utilisés pour attester l'intégrité et l'authenticité des messages. Récemment, un certain nombre d'algorithmes de chiffrement par bloc dédiés à plateformes limitées ont été proposés, et quelques-uns parmi eux offrent une performance et sécurité adéquats pour l'utilisation dans ce genre de scénario (e.g., le CURUPIRA-2). Pour cette raison, dans ce document on s'intéresse au projet et à l'analyse d'algorithmes pour l'authentification de messages. Notre but est de développer des solutions que soient au même temps sûrs et suffisamment légères pour l'utilisation dans les réseaux de senseurs sans fil et d'autres scénarios si restreints.

L'algorithme de MAC proposé dans ce document a été nommé MARVIN. Ce nouveau MAC adopte la construction ALRED, laquelle permet à MARVIN de réutiliser quelque parties d'un algorithme de chiffrement par bloc sous-jacent ; de cette façon, l'implémentation de MARVIN requiert une quantité réduite de mémoire. En plus, la structure de l'algorithme proposé est très flexible et entièrement parallélisable, ce que permet l'adoption de différentes stratégies d'optimisation, en consonance avec les ressources disponibles dans la plateforme ciblé. Comme avantage supplémentaire, MARVIN peut être utilisé indépendamment (simplement pour authentifier des messages) ou dans un mode de Authentification Chiffrée avec Données Associées (*Authenticated-Encryption with Associated Data* — AEAD), lequel allie authentification et chiffrement. En profitant de la structure interne de Marvin. Ici, on propose un nouveau AEAD dédié à des plateformes restreintes, lequel harmonise MARVIN et le mode d'opération LFSRC (*Linear Feedback Shift Register Counter*). Finalement, on analyse en détail la sécurité des algorithmes résultants, ainsi que leur performance dans des scénarios représentatifs.

Mots-clés: Cryptographie. Réseaux de Senseurs Sans Fil. Réseaux de ressources limitées. Codes d'Authentification de Message (MACs). Authentification Chiffrée avec Données Associées (AEAD).

CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF ACRONYMS

AE Authenticated-Encryption

AEAD Authenticated-Encryption with Associated Data

AEM Advanced Encryption Mode — deprecated name of OCB

AES Advanced Encryption Standard — block cipher

AP Access Point

CBC Cipher Block Chaining — mode of operation

CBC-MAC Cipher Block Chaining MAC — MAC algorithm

CCM Counter with CBC-MAC — AEAD algorithm

CMAC Cipher-based MAC — MAC algorithm

CRC Cyclic Redundancy Check

CS Cipher-State — AE scheme

CTR Counter — mode of operation

DES Data Encryption Standard — block cipher

EAX Encrypt-then-Authenticate-then-Translate — AEAD algorithm

ECB Electronic CodeBook — mode of operation

ECC Elliptic Curve Cryptography

EaM Encrypt-and-MAC

EtM Encrypt-then-MAC

GCM Galois/Counter Mode — AEAD algorithm

GF Galois Field (or Finite Field)

GHASH Galois Hash — the universal hash function used by GCM

GMAC Galois Message Authentication Code — MAC algorithm

GPL GNU General Public License

Hashed-MAC Hashed Message Authentication Code — MAC algorithm

IDE Integrated Development Environment

IP Internet Protocol

IV Initialization Vector

JDK Java Development Kit

LFSRC Linear Feedback Shift Register Counter — mode of operation

LUT Look-Up Table

MAC Message Authentication Code

MD Message Digest — family of hash algorithms

MDS Maximal Distance Separable

MEDP Maximum Expected Differential Probability

MtE MAC-then-Encrypt

NIST National Institute of Standards and Technology

OCB Offset CodeBook — AEAD scheme

OMAC1 One-Key CBC-MAC version 1(also known as CMAC) — MAC algorithm

PC Personal Computer

PMAC1 Parallelizable MAC version 1 — MAC algorithm

PRF Pseudo-Random Function

RAM Random Access Memory

RC4 Rivest Cipher 4 — stream cipher

RSA Rivest, Shamir, & Adleman — asymmetric encryption algorithm

SCT Square-Complete Transform

SHA Secure Hash Algorithm — family of hash algorithms

SNEP Secure Network Encryption Protocol

SPINS Security Protocols for Sensor Networks

SPN Substitution-Permutation Network

TCP Transmission Control Protocol

WSN Wireless Sensor Network

XOR Exclusive OR

1 INTRODUCTION

Wireless networks can be defined as a collection of mobile and/or fixed nodes that are not physically interconnected, forming a dynamic topology. Basically, these networks can be classified as *infra-structured* or *ad-hoc*. In infra-structured networks, the nodes have direct access to Access Points (APs), which are typically connected to a central administration in a wired network (e.g., Ethernet networks) with higher connection speeds than those achieved by the wireless network itself; for this reason, all communication between the nodes of the network pass through the APs. Ad-hoc networks, on the other hand, are independent from any infrastructure or administrative entity; in this case, the nodes that build the network also play the role of routers, retransmitting packets from neighboring nodes that are unable to communicate directly due to their limited radio range.

Recent advances in wireless communication have created many new prominent areas of research. Among them, one that has received much attention is Wireless Sensor Networks (WSNs). WSNs are an especial type of ad-hoc network, composed by a large number of tiny, cheap and highly resource constrained sensor nodes, known as *motes* (AKYILDIZ et al., 2002; YICK; MUKHERJEE; GHOSAL, 2008)[1]. These sensors can gather and process data from the environment (e.g., mechanical, thermal, biological, chemical, and optical readings), enabling the development of applications such as environment and habitat monitoring, surveillance, support for logistics, indoor climate control, structural monitoring, health care and emergency response, as well as military

[1] *Mote*: n.; A very small particle, as of floating dust; anything proverbially small.

operations (ARAMPATZIS; LYGEROS; MANESIS, 2005; ALEMDAR; IBNKAHLA, 2007).

Sensors used in WSNs are typically battery-powered, which has motivated considerable research efforts on the development of energy-aware protocols. Some notable examples of data link layer protocols are described in (YAHYA; BEN-OTHMAN, 2009). There are also protocols for data collection, such as diffusion (INTANAGONWIWAT; GOVINDAN; ESTRIN, 2000) and SPIN (KULIK; HEINZELMAN; BALAKRISHNAN, 2002), data aggregation (SOLIS; OBRACZKA, 2004), topology control (CERPA; ESTRIN, 2002), and so on. In general, the main goal driving the design of these protocols is to optimize network communications. This happens because transmission in WSNs usually consume considerably more energy than computation — e.g., 1 bit transmitted may require the power equivalent to executing 800-1000 instructions (KARLOF; SASTRY; WAGNER, 2004). Therefore, this becomes an effective strategy to save energy and thus extend the network's lifetime.

On the other hand, since data in WSNs is transmitted over the air and many applications monitor sensitive information, the deployment of security mechanisms is also essential. Such mechanisms are used to prevent unauthorized users from eavesdropping the information trasmited or introducing malicious data into the network; without them, private information could be made available without the approval of legitimate users, or the system could take mistaken actions due to the invalid data.

A main challenge to the deployment of security in WSNs is that battery-powered sensor networks impose several constraints on the cryptographic algorithms that can be effectively used. Commercial motes usually have 8-128 KiB of code memory, 4-10 KiB of data memory (RAM) and are equipped with 8- or 16-bit processors operating at 4-16 MHz. Some examples are shown in Table 1. Moreover, messages exchanged in these applications are frequently small: the lenght of a typical packet for monitoring and control applications is expected to be between 30 and 60 bytes (CORDEIRO; AGRAWAL, 2006; KARENOS; KALOGERAKI, 2006). Finally, the batteries typically used

as energy supply in WSNs are such that they can be depleted in about 72 hours if the mote constantly operates in active mode (ZHANG et al., 2004). In this context, complex all-purpose algorithms not only take longer to run but also consume more energy, bandwidth and memory, which motivates the research for more efficient alternatives.

Table 1: Comparison of some commercial motes: MICAz (CROSSBOW, 2008b), Mica2 (CROSSBOW, 2008a), FireFly (NANO-RK, 2007), TelosB (CROSSBOW, 2008c) and the sensor used in the Smart Dust project (HILL et al., 2000)

	Processor	Code Memory	RAM	Bandwidth
Smart Dust	8-bit, 4 MHz	8 KiB	0.5 KiB	10 kbps
MICAz	8-bit, 7.3 MHz	128 KiB	4 KiB	250 kbps
Mica2	8-bit, 7.3 MHz	128 KiB	4 KiB	38.4 kbps
FireFly	8-bit, 7.3 MHz	128 KiB	8 KiB	250 kbps
TelosB	16-bit, 8 MHz	48 KiB	10 KiB	250 kbps

1.1 Motivation

To date, many architectures have been proposed to provide security in WSNs. One of the most popular is TinySec (KARLOF; SASTRY; WAGNER, 2004), which offers link layer security (authenticity, integrity and confidentiality) to TinyOS (LEVIS et al., 2004), the *de facto* standard operating system for sensor networks; this solution has two modes of operation, one that provides data authentication only (called *TinySec-Auth*) and another that combines encryption and authentication (*TinySec-AE*). Another prominent solution is the Secure Network Encryption Protocol (SNEP), the component of the Security Protocols for Sensor Networks (SPINS) (PERRIG et al., 2001) responsible for data confidentiality, two-party data authentication, and data freshness. More recently, the MiniSec (LUK et al., 2007) and Sensec (LI et al., 2005) architectures have been proposed for providing similar security services, but promising a lower energy consumption.

Despite their careful design, an important issue with these solutions concerns the cryptographic algorithms they adopt: most of times, general-purpose solutions are suggested, though they are not as optimal as possible in such constrained scenarios. For

example, TinySec has chosen Skipjack (NSA, 1998) as default block cipher due to its good performance; however, since this cipher uses relatively small (80-bit) keys and 31 out of its 32 rounds can be successfully cryptanalyzed (BIHAM; BIRYUKOV; SHAMIR, 1999), it presents a very low margin of security.

In order to address this issue, special-purpose cryptographic solutions have been developed. Recently proposed block ciphers include PRESENT (BOGDANOV et al., 2007), HIGHT (HONG et al., 2006), and CURUPIRA (SIMPLICIO, 2008; BARRETO; SIMPLICIO, 2007), which aim to combine security and performance in constrained platforms. Indeed, CURUPIRA's security analysis (BARRETO; SIMPLICIO, 2007; NAKAHARA, 2008) shows that up to 7 out of 10 (or more) rounds can be successfully cryptanalyzed; at the same time, the second version of CURUPIRA, named CURUPIRA-2, achieves a better performance than Skipjack in most of the constrained platforms on which it has been tested (SIMPLICIO et al., 2008).

Motivated by these results in the field of confidentiality, it is reasonable to consider further optimizations on another crucial subject: message authentication. Most security architectures for WSNs (including TinySec, SNEP/SPINS and ZigBee (ZIGBEE, 2005)) address this issue by deploying some MAC algorithm[2] derived from the all-purpose Cipher Block Chaining MAC (CBC-MAC). With a carefully chosen variation, CBC-MAC is indeed a secure solution (BELLARE; KILIAN; ROGAWAY, 2000). Nonetheless, its performance is an important concern, since it requires a full encryption per authenticated block and is strictly sequential (i.e., it cannot be parallelized). Furthermore, each of these variants usually displays further (though more subtle) performance issues. The one adopted in TinySec, for example, encrypts the message-length and XORs it with the first plaintext block, a strategy having the following drawbacks: the message size must be known before the tag computation starts (hence, the algorithm is not online (PETRANK; RACKOFF, 2000)), and one additional encryption is needed for

[2] Throughout this document, we will use the abbreviation "MAC" to indicate "Message Authentication Code", not the Media Access Control address (commonly known as MAC-address).

each different message length (computing a single constant is not enough).

Another possible approach for providing message authentication is to adopt an Authenticated-Encryption with Associated Data (AEAD) scheme, i.e., a solution that provides encryption, authentication or both at the same time. This is the strategy of MiniSec, for example, which adopts the Offset CodeBook (OCB) (KROVETZ; ROGAWAY, 2005) as its authentication and encryption mechanism. However, OCB also has some limitations: it presents a reasonably high cost of processing plaintext data (one full encryption per block); it requires the implementation of the underlying block cipher's encryption and decryption algorithms instead of only the former (affecting the overall memory usage); it does not allow the verification of the authentication tags prior to its decryption (i.e., all messages must be decrypted, even fake ones); finally, OCB is encumbered with patents, hindering its broad adoption.

Therefore, in this scenario where there is no absolute winner, the development of lightweight, compact, flexible and unpatented message authentication mechanisms becomes of great interest.

1.2 Goals

Our goal in this research is to specify and develop cryptographic algorithms for message authentication that are, at the same time, well adapted to constrained platforms and flexible enough to take advantage of extra resources available in more powerful platforms. More specifically, we aim to develop a MAC function and an AEAD scheme that are not only better adapted to WSNs — and similarly constrained scenarios — than similar algorithms currently available, providing an adequate level of security at low cost (in terms of memory, processing and energy overhead), but also to explore modern computers' features such as parallelism. In fact, even if most sensors are unable to take advantage of parallelizability due to the absence of multiple

computing units, this is also an interesting feature in WSNs when one has interest in deploying a fast hardware implementation, or when powerful nodes are also part of the sensor network — e.g., a multi-core base station that processes many packets larger than those usual due to data aggregation mechanisms (PATEL; VENKATESAN; WEINER, 2007). Therefore, although we focus on constrained platforms along this document, and especially in WSNs, the algorithms proposed should be useful in a wide range of applications and scenarios.

Finally, the security of the proposed algorithms shall be proved by means of formal analysis, and their performance shall be evaluated both in resourceful devices (e.g., a modern computer) and in constrained platforms (such as motes).

1.3 Methodology

Considering the intrinsic restrictions of the sensors, the construction of message authentication algorithms requires the study of suitable techniques for designing efficient and secure solutions. This work focuses mainly on techniques that reuse parts of an underlying block cipher machinery, resulting in algorithms with reduced memory footprint. At the same time, we aim to add parallelizability to the resulting structure, thus creating a more flexible solution as previously stated. Finally, we study how we can merge encryption and authentication in a hybrid AEAD scheme.

Due to the need of concrete security proofs, in our analysis we adopt the so-called *Game-Playing technique* (BELLARE; ROGAWAY, 2004; SHOUP, 2004), a widely accepted method for verifying the security of cryptographic solutions.

Finally, the performance evaluation of the proposed algorithms is both theoretical — we evaluate the expected amount of computation and memory used by the proposed solutions — and practical — we develop a comparative benchmark involving ours and similar solutions, in some relevant platforms. In order to allow a fair comparison, we

use similar coding techniques in such comparative benchmark.

1.4 Original Contributions

The main contributions of this thesis are: (1) we develop a novel and flexible Message Authentication Code (MAC) named MARVIN, which is especially adapted for constrained networks such as WSNs; (2) we develop a novel Authenticated-Encryption with Associated Data (AEAD) scheme named LETTERSOUP, which gracefully combines features from MARVIN and the Linear Feedback Shift Register Counter (LFSRC) mode to achieve a compact and efficient solution; (3) we provide a security analysis of the proposed solutions, showing that both MARVIN and LETTERSOUP are secure as long as the underlying block cipher adopted is also secure; (4) we present the benchmark results of ours and widely used message authentication solutions (both MACs and AEAD schemes) in some relevant platforms, showing that MARVIN and LETTERSOUP are flexible algorithms and have a high potential for deployment in WSNs.

1.5 Document Organization

This document is organized in chapters. Chapter 2 describes some basic concepts, including the elements used in cryptography for providing data and communication security. Chapter 3 presents some design techniques for building message authentication algorithms, evaluating the suitability for use in WSNs, and thus launching the knowledge base needed in Chapter 4, which introduces MARVIN and LETTERSOUP algorithms. The security of both solutions is formally analyzed in Chapter 5. Then we evaluate MARVIN's and LETTERSOUP's performance in Chapter 6, which includes our benchmark results in some representative platforms. Finally, we conclude our discussion and present our future work in Chapter 7.

2 CRYPTOGRAPHY: CONCEPTS AND NOTATION

In order to allow a better comprehension of the concepts explored in this document, it is necessary to clearly understand some basic concepts involved in the area of Cryptography. This is the main goal of this chapter, which covers the basic services provided by cryptographic functions and how they can be implemented. More specifically, this chapter summarizes the characteristics and the utilization of Symmetric and Asymmetric Ciphers, Hash-Functions, Message Authentication Codes (MACs) and Authenticated-Encryption with Associated Data (AEAD) schemes. In this manner, we aim to provide enough background for the (more detailed) discussion about MACs and AEAD schemes that will be covered in the chapters that follows. Additionally, this chapter also presents the basic notation used along this document. Since this notation is quite standard, readers that have some familiarity with the above-mentioned concepts are can probably proceed to the next chapter. On the other hand, for readers looking for a mor detailed discussion on the cryptographic solutions present in the following and on other cryptography-related subjects, we recommend the reading of (MENEZES; OORSCHOT; VANSTONE, 1999).

2.1 Security Services

The term Cryptography (from the Greek: *kryptós*, “hidden”, and *gráphein*, “writing”) is historically associated to the art of “hiding information”, which can be interpreted as the capability to provide confidentiality to some (probably important) infor-

mation. However, nowadays the term Cryptography refers to a wider range of security services, listed below (MENEZES; OORSCHOT; VANSTONE, 1999):

- *Confidentiality*: assurance that the messages exchanged between the communicating parties can be correctly interpreted only by authorized users, i.e., that only them can access the information contained in the transmitted data.

- *Integrity*: the possibility of verifying an unauthorized alteration of the data. It is important to note that this service does not prevent the data modification, but simply allows users to detect such event.

- *Authentication*: the process of reliably verifying the identity of someone or something.

- *Non-repudiation*: assurance that someone cannot deny something, such as the transmission, reception or possession of some information.

Another service that is usually required in secure architectures is *availability*, which refers to the system's capability to keep running in good conditions despite the number of users accessing (or attacking) it. However, this is not a service directly offered by cryptography, though the deployment of above-mentioned cryptographic services can aid in this task (e.g., filtering unauthorized users by means of authentication). Indeed, this is an example that shows that the security of communications and systems usually requires a careful combination of some (cryptographic and non-cryptographic) services.

Nowadays, many types of cryptographic algorithms are used for providing these services. In the next sections, we will briefly discuss some of the cryptographic functions that are most commonly used and which type of service can be achieved with them.

2.2 Ciphers

Ciphers are reversible functions used for providing confidentiality. During the process known as *encryption*, a cipher transforms a plaintext M (legible data) into a ciphertext C (meaningless data) by applying a key K_e; in the inverse process, known as *decryption*, C is turned into M again by means of a key K_d. More formally, we can define the encryption and decryption processes as follows:

Definition 1. *Let $\mathcal{M}$ be a set of plaintexts, $\mathcal{C}$ a set of ciphertexts and $\mathcal{K}$ a set of keys. An encryption function E associates to each pair $(M, K_e) \in \mathcal{M} \times \mathcal{K}$ an element $C = E(M, K_e) \in \mathcal{C}$. The corresponding decryption function E^{-1}, together with the decryption key K_d, associates to the pair $(C, K_d) \in \mathcal{C} \times \mathcal{K}$ the same element $M \in \mathcal{M}$, in such a manner that $E^{-1}(E(M, K_e), K_d) = M$.*

The knowledge of the secret decryption key K_d is essential for gaining access to the information hidden in a ciphertext created using key K_e. When a cipher takes the same key for both encryption and decryption (i.e., $K_e = K_d = K$), it is called a *symmetric cipher*; when different keys are used, it is called an *asymmetric cipher*.

Along this document, we write either $E_K(M)$ or $E(M, K)$ to denote the encryption of some message M using cipher E under a k-bit long key K. Analogously, we write $E_K^{-1}(M)$ or $E^{-1}(M, K)$ to denote the decryption process of cipher E under the k-bit long key K.

2.2.1 Symmetric Ciphers

Many symmetric ciphers operate only over messages of fixed size, i.e., both M and C must always have a pre-defined length n (e.g., 128 bits). Ciphers having this characteristic are called *block ciphers* and the value n is called the *block size*.

Whenever the size of the plaintext M (denoted $|M|$) is different from n, especial

techniques must be employed in order to allow its encryption by an n-bit block cipher:

- *Padding*: Padding techniques are used when the plaintext M is smaller than the cipher's block size n. Their operation is simple: some bits are added to M, in such a manner that its final size becomes n. A commonly used technique is the "NIST padding" (NIST, 2001b)[1] , which concatenates $\{10^*\}$ to the message, i.e., a single bit '1' followed by enough bits '0'.

- *Modes of Operation*: Messages that are larger than the block size must be handled according to some *mode of operation*, which defines how each message block is processed. One could simply encrypt each block separately, as in the Electronic CodeBook (ECB) mode (BELLOVIN; BLAZE, 2001), but a more commonly adopted mode is the Cipher Block Chaining (CBC). In this mode, the plaintext M is broken into n-bit blocks $M_0 \| M_1 \| \ldots \| M_t$; each block M_i is XORed with the previously encrypted block C_{i-1} and then encrypted, yielding C_i, as illustrated in Figure 1. Since the first block M_0 has no predecessor, it is XORed with a special block called the *Initialization Vector (IV)*, which is a public parameter (e.g., a block filled with '0's). Padding mechanisms are usually used in case the last block, M_t, is incomplete (in Figure 1, the NIST padding is depicted).

Stream Ciphers, on the other hand, do not operate on fixed-size data blocks, but rather produce a sequence of pseudo-random bits that is combined with the message by means of simple operations, such as bitwise Exclusive-OR (denoted XOR or $\oplus$). This pseudo-random sequence is always key-dependent but, in some stream ciphers, it depends also on the input plain message and/or on the ciphertext previously generated. Since there is no restriction on the size of the message to be processed, stream ciphers do not require padding mechanisms or the adoption of a mode of operation.

[1] NIST stands for the U.S. "National Institute of Standards and Technology".

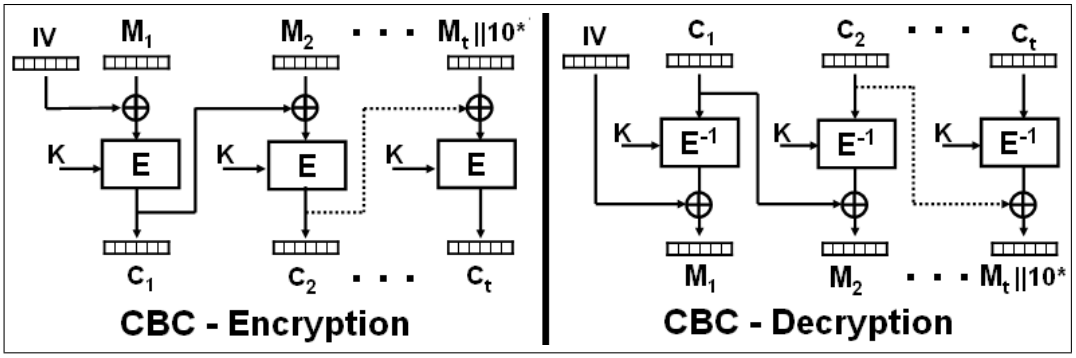

Figure 1: Encryption and decryption with the Cipher Block Chaining (CBC) mode of operation.

The design of secure stream ciphers is usually considered a very difficult task. For this reason, the usual way of achieving their characteristic "stream behavior" is to use block ciphers in some mode of operation that allows the generation of pseudo-random bit-streams. A simple example is the *Counter Mode* (also called CTR), illustrated in Figure 2. In this mode, the bit-stream is generated by encrypting successive values of a counter, initialized with some IV. In the example of Figure 2, we show the case where $IV = 0$ (or, more precisely, $[0]_n$, the n-bit representation of 0). It is important to note that the CTR mode uses only the encryption function E_K; hence, the decryption function E_K^{-1} does not need to be implemented when this mode is adopted.

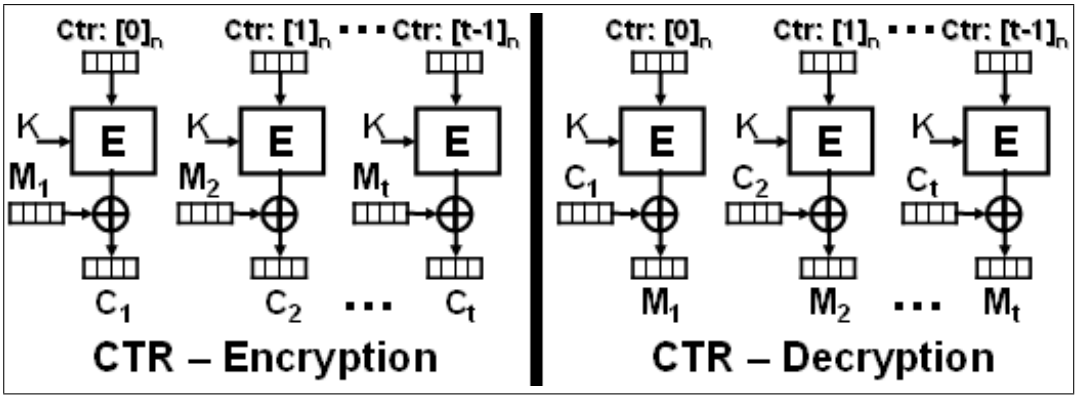

Figure 2: Encryption and decryption with the Counter (CTR) mode of operation.

Many block and stream ciphers adopt an iterative structure (in *rounds*), meaning that the encryption and decryption operations can be decomposed in smaller, identical sub-operations called *round functions*. The input and output blocks of a round function

are called *states*. Hence, the data to be processed passes $r > 0$ times through the round functions, each of which taking a *round subkey*. These subkeys are generated from the initial key K through the process known as *key schedule*. The exact number of rounds r depends on the cipher design (e.g., Skipjack (NSA, 1998) takes 32 rounds) and, possibly, on the key size (e.g., CURUPIRA takes 10, 14 and 18 rounds for, respectively, 96-, 144- and 192-bit keys). In order to illustrate the idea, consider the following example: the encryption operation in a 3-round block cipher having ρ as its round function would look like $E_K : C = E(M, K) = \rho(\rho(\rho(M, K_1), K_2), K_3)$, while the decryption would take the form $E^{-1} : M = E^{-1}(C, K) = \rho^{-1}(\rho^{-1}(\rho^{-1}(C, K_3), K_2), K_1)$ (here, K_i represents the subkey at the i^{th} round).

2.2.2 Asymmetric Ciphers

Before 1976, when the asymmetric cryptography was first introduced in the civil domain (DIFFIE; HELLMAN, 1976)[2] , data confidentiality was provided only by symmetric ciphers. The shortcoming in this case was that, before the users were able to establish a secure communication, they were obliged to share a secret key, which needed to be distributed through a secure channel (e.g., in a face-to-face meeting).

This is not the case with asymmetric cryptography, in which each user receives two keys: one is made public (and is called a *public key*), while the other one is kept secret (the *private key*). In this manner, if Alice wants to send a message M that is intended to Bob only, Alice can encrypt the message with Bob's public key K_B^u and send the resulting ciphertext $C = E(M, K_B^u)$ through any channel (including insecure ones); since Bob is the only one who knows his own private key K_B^r, he is the only one able to decrypt C and recover the correct plaintext message $M = E^{-1}(C, K_B^r)$. Therefore, in these schemes, two users can communicate privately even if there is no prior shared

[2] Allegedly, the asymmetric cryptography was known by the militaries before 1976 (WILLIAMSON, 1974), but it was only in 1976 that it was presented to the public in general by the British cryptographers Diffie and Hellman (DIFFIE; HELLMAN, 1976).

secret between them. However, this greater versatility comes with a price: asymmetric algorithms are typically much slower and require larger keys than symmetric algorithms for similar security levels. For this reason, asymmetric algorithms are usually employed at the beginning of the communication for protecting the transmission of a symmetric key, which is the one used after this first *key-agreement* step. Furthermore, asymmetric ciphers can also be used with other purposes, such as the creation of digital signatures, as discussed in section 2.4.

2.2.3 Examples

Table 2 lists some symmetric and asymmetric algorithms that can be used for data encryption, as well as some of their characteristics.

Table 2: Examples of symmetric and asymmetric encryption algorithms: DES (NIST, 1977), AES (NIST, 2001a), Skipjack (NSA, 1998), CURUPIRA (BARRETO; SIMPLICIO, 2007; SIMPLICIO et al., 2008), RC4 (FLUHRER; MANTIN; SHAMIR, 2001), RSA (RIVEST; SHAMIR; ADELMAN, 1977) and ElGamal (GAMAL, 1985).

Algorithm	Type	Comments
DES	Symmetric Block Cipher	*Data Encryption Standard*. The former encryption standard (up to 2001). Its use is not recommended anymore due to its reduced security level. Block: 64 bits; Key: 56 bits.
AES	Symmetric Block Cipher	*Advanced Encryption Standard*. The current encryption standard (since 2001). Block: 128 bits; Key: 128, 192 or 256 bits.
Skipjack	Symmetric Block Cipher	The default cipher adopted by TinySec (KARLOF; SASTRY; WAGNER, 2004). Block: 64 bits; Key: 80 bits.
CURUPIRA	Symmetric Block Cipher	A lightweight cipher for constrained networks. Block: 96 bits; Key: 96, 144, or 192 bits.
RC4	Symmetric Stream Cipher	One of the most widely used stream cipher. Designed by Ron Rivest in 1987, but never officially published. After it was reverse-engineered, many security flaws became public (FLUHRER; MANTIN; SHAMIR, 2001). Key: usually 40 to 256 bits.
RSA	Asymmetric	An asymmetric algorithm whose security is based upon the impracticability of factoring large numbers.
ElGamal	Asymmetric	An asymmetric algorithm whose security depends upon the difficulty of computing discrete logarithms (MENEZES; OORSCHOT; VANSTONE, 1999).

2.3 Hash-Functions

A Hash-Function H is a one-way (i.e., non-invertible) transformation that maps an arbitrary-length input message M to a fixed-length output $H(M)$, called the *hash-value* or simply *hash* of M. Therefore, the hash can be seen as a "digest" of the original message M.

Hash-functions can be used to verify the messages' integrity: since a modification in M will result in a different hash, any user can verify that the modified message M' does not map to $H(M)$. Note, though, that the validity of $H(M)$ itself must be assured somehow, or a malicious user could replace $H(M)$ by $H(M')$, misleading this verification process. Moreover, since hash-functions are "many-to-one" functions, the existence of collisions (pairs of inputs M and M' that are mapped to the same output $H(M)$) is unavoidable. Indeed, supposing that all input messages have length at most t bits, that the outputs are h-bit long and that all 2^h outputs are equiprobable, then 2^{t-h} inputs will map to each output, and two input picked at random will yield to the same output with a probability of 2^{-h}. In order to prevent attackers from using this property to their advantage, secure hash algorithms must satisfy the following three requirements:

1. *First Pre-image Resistance*: given a hash $H(M)$, it is computationally infeasible to find any message M having that hash-value, i.e., it is computationally infeasible to invert the hash-function.

2. *Second Pre-image Resistance*: given a message M and its corresponding hash $H(M)$, it is computationally infeasible to find any other message M' that maps to the same hash, i.e., such that $H(M) = H(M')$.

3. *Collision Resistance*: it is computationally infeasible to find any two distinct messages M and M' such that $H(M) = H(M')$.

As long as the above properties of the hash-function hold true and one chooses an adequate hash length h (i.e., one that minimizes the probability of collisions, 2^{-h}), the uniqueness of the relationship between M and $H(M)$ can be (and usually is) assumed in practice.

2.3.1 Examples

Some examples of cryptographic hash-functions are shown in Table 3.

Table 3: Examples of hash-functions: MD4 (RIVEST, 1990), MD5 (RIVEST, 1992), SHA-1 (NIST, 2008), SHA-2 (NIST, 2008) and SHA-3 (NIST, 2007a).

Algorithm	Hash length	Comments
MD4	128	A member of the MD (*Message Digest*) family. It is currently known to be highly insecure, due to the easiness of creating collisions with it (SASAKI et al., 2007).
MD5	128	Developed in order to correct some vulnerabilities found in MD4. However, MD5 is also known to be insecure (LEURENT, 2007).
SHA-1	160	SHA stands for *Secure Hash Algorithm*. SHA-1 is one of the most widely used hash-functions. However, it presents security flaws (WANG; YIN; YU, 2005), motivating its replacement by more robust algorithms.
SHA-2	Variable	The SHA-2 family comprises different hash lengths: 224, 256, 384, and 512. Many attacks on SHA-1 also apply to the SHA-2 family, but the latter's longer hashes still provide a reasonable security level (NIST, 2009).
SHA-3	Variable	To be defined by NIST SHA-3 Competition, which started in 2007 (NIST, 2007a). Hash lengths that must be supported are: 224, 256, 384, and 512.

2.4 Digital Signatures

As discussed in section 2.2, a common utilization of asymmetric algorithms is to encrypt some information (e.g., a symmetric key) with the receiver's public key; this procedure assures that the receiver is the only one capable of decrypting the message, since no one else knows the needed private key.

However, asymmetric algorithms have another very interesting utilization: they allow users to digitally sign their messages. This is accomplished in the following manner: whenever Alice wants to send a digitally signed message to Bob, she can send the message M together with its *digital signature* $S = E(M, K_A^r)$, i.e., the result of encrypting M with Alice's own private key K_A^r. When Bob receives both M and S, he can decrypt S using Alice's public key K_A^u, and then verify if $E^{-1}(S, K_A^u) = M$, i.e., if the decryption of S results in M. If that is the case, then Bob can be assured that Alice was the one who sent the message, since she is the only one who could compute S using her private key K_A^r; moreover, and for the same reason, Alice cannot deny that she was the one who signed the message M. If the check fails, though, then Alice was not the one who created the signature, or possibly M or S has been modified (unintentionally or with malicious intents) before the reception by Bob. Therefore, Digital Signatures provide three services at once: integrity, authenticity and non-repudiation.

Due to the considerably high cost of asymmetric operations, in practice the digital signatures do not involve encrypting the entire message but only its hash, which has a fixed and arbitrarily small size. Hence, normally Alice would send a message M and the signature $S' = E(H(M), K_A^r)$, while Bob would compute the hash of the received message, $H(M)$, and only then verify if $E^{-1}(S', K_A^u) = H(M)$. For this reason, it is important to assure the security of the hash-function deployed, or an attacker can generate a fake message M' in such a manner that $H(M') = H(M)$; since $H(M)$ has been signed by Alice, the attacker can claim that message M' has also been signed by her. Moreover, the secrecy of the private keys is essential: the theft of Alice's private key would allow attackers to impersonate her.

2.5 Message Authentication Codes (MACs)

Message Authentication Codes (MACs) are similar to Hash-Functions, since they also are one-way (non-invertible) functions used to create a fixed-length output from

an arbitrary-length input message. The main difference between the two cryptographic solutions is that MACs take a secret key K as argument, together with the message M to be authenticated, generating an *authentication tag* $T = MAC(M, K)$ — also denoted $MAC_K(M)$ along this document. For this reason, MAC algorithms are also referred to as "Keyed Hash-Functions" (MENEZES; OORSCHOT; VANSTONE, 1999). Indeed, the security threats against hash-functions and MAC algorithms are strongly related: an attacker who knows a valid message+tag pair (M, T) and is able to find a message M' such as $MAC_K(M') = T$ (i.e., find a collision) could replace M during the communication without being noticed. Hence, a secure MAC algorithm shall prevent the *forgery* of tags, i.e., the generation of a valid tag for some message shall be computationally infeasible. Additionally, and unlike (unkeyed) hash-functions, the security of MAC algorithms also depend on its capability to prevent attackers from recovering the secret key K. These security requirements will be further explored in Chapter 5.

The utilization of MAC algorithms is as simple as follows. Suppose that Alice and Bob share a secret key K. First Alice computes the authentication tag $T = MAC_K(M)$ and send both M and T to Bob. Upon reception, Bob computes $T' = MAC_K(M)$ and verifies if $T = T'$. If the check is successful, Bob accepts the message; otherwise, either M or T has been modified prior to its reception, or a user who does not know K made a forgery attempt.

Due to the presence of a secret key, MAC algorithms provide not only integrity to the messages, but also authentication, since only authorized users are able to create and correctly verify the authentication tags. In other words, users having no access to K cannot replace M by M' without being noticed, because they will be unable to compute a valid authentication tag for M'; in comparison, such replacement would be possible if an unkeyed hash-function was used instead of a MAC function. It is important to note, however, that MAC algorithms do not provide non-repudiation: in the previous example, Bob cannot prove that an authentication tag T has been generated by Alice

(and vice-versa) because, as both Alice and Bob knows the secret key K, any of them could have computed T.

2.5.1 Examples

MAC functions are usually deployed together with other cryptographic solutions, such as block ciphers and hash-functions. For this reason, many MAC constructions reuse part of these algorithms' machinery, leading to more compact designs. Table 4 shows some MAC algorithms commonly used in modern applications. All MACs in this table are based on an underlying block cipher, except for the Hashed-MAC (NIST, 2002), which is based on a hash-function.

Table 4: Examples of MAC algorithms: CBC-MAC (NIST, 1985), CMAC (IWATA; KUROSAWA, 2003; NIST, 2005), PMAC1 (ROGAWAY, 2004) and Hashed-MAC (NIST, 2002).

Algorithm	Comments
CBC-MAC	The *Cipher Block Chaining MAC*. This algorithm is known to be insecure when used with messages of variable length. (MENEZES; OORSCHOT; VANSTONE, 1999).
CMAC	Originally named One-Key CBC-MAC version 1 (OMAC1), this is a fairly simple variant of CBC-MAC that is secure for messages of variable length.
PMAC1	The *Parallelizable MAC* version 1 is a fully parallelizable MAC algorithm. There are pending patents covering its usage (ROGAWAY, 2001).
HMAC	The *keyed-Hash MAC* is a MAC algorithm based on an underlying hash-function.

2.6 Authenticated-Encryption with Associated Data (AEAD)

Many scenarios require the deployment of confidentiality, integrity and message authentication as security services. Indeed, these are exactly the services offered by most WSN-oriented security architectures, including TinySec (KARLOF; SASTRY; WAGNER, 2004), Minisec (LUK et al., 2007) and Sensec (LI et al., 2005). One important rea-

son for this fact is that confidentiality alone cannot assure a secure communication: without access to the secret key K, attackers are unable to access the content of an encrypted message $C = E_K(M)$; however, they still can flip some bits of C, turning it into C', in such a manner that its receiver will believe that $M' = E_K^{-1}(C')$, and not M, was the message originally sent. Possibly, this malicious modification will create an M' that makes no sense, and can thus be discarded after verification by a human reader; however, such verification process using automatic systems could become a very troublesome task, motivating the deployment of more effective approaches.

Authenticated-Encryption (AE) schemes combine the operations of a block cipher and of a MAC function, under the same key, for providing confidentiality, integrity and message authentication at the same time. Generally speaking, what they do is to take a confidential message M as input, and then create its corresponding ciphertext C and authentication tag T as output. Some schemes allow the tag verification process to take place prior to decryption; this is considered an interesting feature because, if T is not valid, the entire message can be promptly discarded prior to its decryption, thus saving some processing.

Some AE schemes also allow the authentication of some non-encrypted data H, i.e., the authentication tag T generated by them assures the authenticity and integrity of both M and H. This is necessary in scenarios where one desires to authenticate not only a packet's confidential payload, but also some plaintext associated to it (e.g., the TCP/IP information, which shall remain unencrypted for packet routing). An AE scheme that supports the authentication of messages consisting of both plaintext and ciphertext in this manner is called an *Authenticated-Encryption with Associated Data (AEAD)* solution, where the "associated data" (also called a *header*) refers to the portion of the message that is transmitted as plaintext. Most modern AE schemes are also AEAD schemes, including those described in Table 5.

When the AE (or AEAD) scheme makes two passes through the data (one aimed

at providing privacy and the other, authenticating it), it is called a *Two-Pass* scheme; when a single pass is made, we have a *One-Pass* scheme. The latter are usually much more efficient than the former, but have one important disadvantage: to date, all One-Pass schemes have been patented, which encumbers their broader utilization.

2.6.1 Examples

Table 5 briefly describes some one-pass and two-pass AEAD algorithms.

Table 5: Examples of AEAD algorithms: EAX (BELLARE; ROGAWAY; WAGNER, 2004), GCM (MCGREW; VIEGA, 2005) and OCB (KROVETZ; ROGAWAY, 2005).

Algorithm	Comments
EAX	A two-pass AEAD scheme that combines the CTR mode of operation with the CMAC (NIST, 2005) algorithm.
GCM	The Galois/Counter Mode (GCM) is a two-pass AEAD scheme that uses the CTR mode of operation and an internal function, GHASH.
OCB	Also known as AEM (ROGAWAY, 2003), the Offset CodeBook (OCB) is a one-pass AEAD scheme that combines a tweaked ECB mode of operation (BELLOVIN; BLAZE, 2001) with PMAC1.

2.7 Confusion and Diffusion

Before we deepen our discussion on message authentication, it is important to present two basic principles commonly used in cryptography: confusion and diffusion. These concepts have been used for evaluating cryptographic applications for a long time (as we will show with some simple examples), but only in 1949 they were formally defined by the electronic engineer and mathematician Claude Shannon (SHANNON, 1949). They can be described as follows:

- *Confusion*: consists in the insertion of non-linear transformations in the cryptographic function. This is usually achieved by means of a substitution table, which simply substitutes a portion of the message by another. A classic example of cryptographic algorithm based solely on this technique is the “Caesar cipher”,

named after Julius Caesar, who used it to communicate with his generals. This cipher simply substitutes each letter of a message by the K^{th} letter that follows it in a cyclic alphabet (i.e., the letter 'z' is followed by the letter 'a'); for example, using the English alphabet and $K = 3$, the plain message "ATTACK" would become "DWWDFN". Note that the substitution table S adopted in the Caesar cipher is linear: for any input letter A and for an alphabet composed by m letters, we have that $S[A] = (A+K) \bmod m$. However, modern cryptographic algorithms tend to adopt highly non-linear substitution tables, known as an *S-Boxes*, aiming to elevate the complexity of the structure linking the input and the output of the cryptographic process.

- *Diffusion*: measures the dependency between the bits of the input and output of the cryptographic function. Ideally, if we flip one bit of the function's input, each bit of the output should be flipped with a probability of 50%. The goal of diffusion is thus to "dissipate" any statistically measurable information (e.g., redundancy) that could exist in the input data. The most ancient form of applying diffusion is attributed to the Spartans, in the Old Greece, who used a cylinder known as *skytale* (or *scytale*) as a cipher (KELLY, 1998). During the encryption process, a strip of paper was wrapped around the skytale and the message was then written; when the paper was removed, the message letters would appear to be completely scrambled; in order to decrypt, the paper should be wrapped around a skytale of the same diameter, re-aligning the letters and thus revealing the plaintext.

The combination of confusion and diffusion mechanisms is the basis for most modern cryptographic algorithms , and will be frequently explored along this document.

2.8 Summary

This chapter discussed how the basic services provided by Cryptography (namely confidentiality, integrity, authentication and non-repudiation), can be implemented by using cryptographic solutions. More specifically, we covered the utilization of Symmetric and Asymmetric Ciphers (used for confidentiality), Hash-Functions (for integrity), Digital Signatures (which employ asymmetric ciphers for providing non-repudiation), Message Authentication Codes (MACs) (for integrity and authentication) and Authenticated-Encryption with Associated Datas (AEADs) (providing confidentiality, integrity and authentication).

Even if the focus of this thesis is on the design of MAC functions and AEAD schemes, this overall discussion is of vital importance because, as briefly discussed in this chapter, many MACs (and, by extension, AEADs) are often constructed from other cryptographic solutions such as block ciphers or hash-functions.

Additionally, we have described two basic principles commonly explored in the construction and evaluation of cryptographic primitives: confusion and diffusion.

Therefore, the background presented in this section will be constantly explored in the next chapter, which focuses on design techniques for MACs and AEADs, and on how well they can be adapted for use in resource-constrained scenarios.

3 DESIGN TECHNIQUES FOR MACS & AEADS

In the previous chapter, we presented some basic concepts and have described cryptographic functions commonly used for providing security. In this chapter, we focus our attention on the construction of MAC and AEAD algorithms, expanding the basic notions previously presented. More specifically, we analyze the advantages and limitations of some modern construction techniques, illustrating their functioning by means of well-known MACs and AEADs based on them.

The main focus of this analysis is on design strategies based on block ciphers. In fact, as discussed in section 2.5, the adoption of underlying cryptographic functions is an economical way of building MACs and AEADs since, whenever this underlying function is already implemented, most of the required operations become promptly available. In applications where data confidentiality is an important requirement and, thus, a cipher implementation should be present, such an underlying function tends to be the most logical choice. Furthermore, the same cipher can be used for encryption and, together with the resulting MAC algorithm, provide a compact AEAD scheme.

We start the discussion with somewhat classical MAC designs, namely CBC-MAC-like and the Carter-Wegman (WEGMAN; CARTER, 1981) constructions. We then examine a very promising solution for resource-constrained platforms: the ALRED (DAEMEN; RIJMEN, 2005a) construction, which is the one adopted by our MAC proposal, MARVIN. Finally, we discuss the design of AEAD schemes, giving the basis for understanding the reasoning behind the structure of our AEAD proposal, LETTER-

SOUP, which is also based on the ALRED construction. Along this discussion, we try to identify the main features and limitations of each design strategy, thus shedding light on the criteria that motivated our choice for ALRED.

3.1 Conventional Block Cipher Based MACs

Consider the problem of computing the authentication tag for a message M under a k-bit key K. For convenience, assume that M is partitioned into t blocks $M_1 \| \ldots \| M_t$, where every block, except possibly M_t, is n-bits long. If an n-bit block cipher E is available, a commonplace and economical strategy is to construct a MAC function based on E. The most well known of such schemes is probably the Cipher Block Chaining MAC (CBC-MAC) (NIST, 1985), depicted in Figure 3.

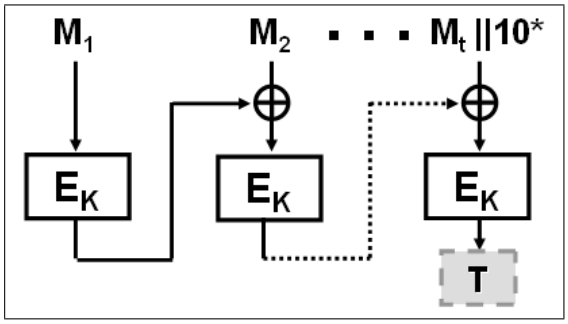

Figure 3: The CBC-MAC algorithm (NIST, 1985).

CBC-MAC displays a simple design, which is very similar to the CBC mode of operation (see section 2.2.1) for an n-bit block cipher and $IV = [0]_n$: the message to be authenticated is broken into t blocks of n bits, each block M_i is XORed with the previously encrypted block C_{i-1} and then the result is encrypted; at the end of this iterative operation, the n-bit authentication tag T is generated. In fact, T can be truncated to $\tau < n$ bits if a smaller tag is desired.

We note that the same truncation strategy works for obtaining τ-bit tags ($\tau < n$) with all algorithms described in this chapter.

A major and well-known problem with CBC-MAC is that it is not secure for au-

thenticating variable-sized messages. An attacker who knows any two valid message-tag pairs (M, T) and (M', T'), such that $M' = M'_1 \| \ldots \| M'_t$, can create a (longer) third message $M'' = M \| M'_1 \oplus T \| M'_2 \| \ldots \| M'_t$ whose authentication tag is also T', i.e., a valid forgery.

This weakness can be overcome with simple modifications in the original CBC-MAC design. For example, the variant adopted in TinySec encrypts the message-length and XORs it with the first plaintext block, as proposed in (BELLARE; KILIAN; ROGAWAY, 2000). However, CMAC (IWATA; KUROSAWA, 2003) is probably the most popular of these CBC-MAC-like schemes, since it has become part of a recommendation by NIST (NIST, 2005). Its strategy consists in XORing the last message block with $L_1 = E_K([0]_n) \cdot x$ (if the last block is complete) or $L_2 = E_K([0]_n) \cdot x^2$ (otherwise) before the final encryption, where $\cdot x^\alpha$ denotes the multiplication by x^α in the finite field GF2^n. This is shown in Figure 4.

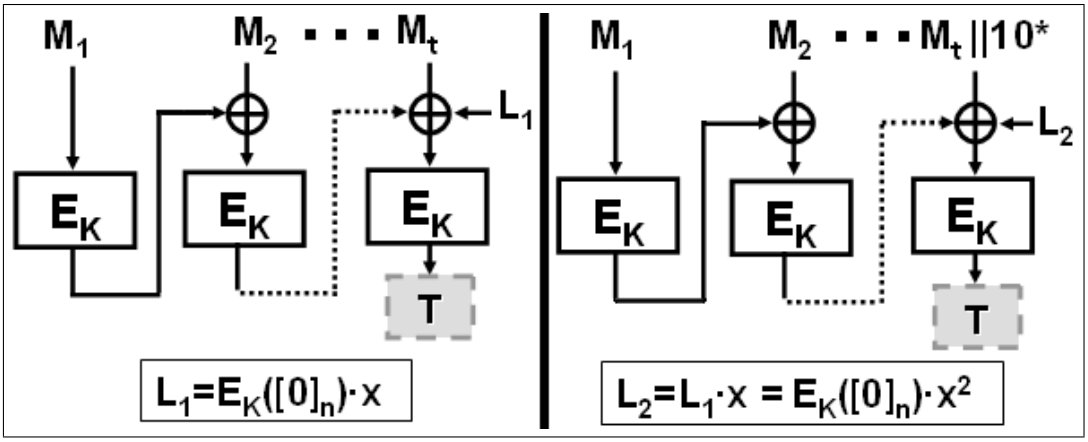

Figure 4: The CMAC algorithm. The left and right sides depict CMAC when the last block is complete and incomplete, respectively.

Also of interest is the PMAC1 algorithm (ROGAWAY, 2004)[1] , depicted in Figure 5. Unlike the strictly sequential CMAC, PMAC1 allows the data blocks $M_1 \| \ldots \| M_t$ to be processed in parallel: after the computation of the first offset $R = 5 \cdot E_K([0]_n)$, different processing units can handle blocks $M_{i<t}$, and the results can be XORed together

[1] The PMAC1 algorithm is a refined version of the original PMAC described in (BLACK; ROGAWAY, 2002). Both are fully parallelizable, but PMAC1 adopts simpler offsets, providing some interesting optimizations as discussed in (ROGAWAY, 2004).

only at the final computation step, when M_t is also processed. This intrinsic parallelism is a very appealing feature when the algorithm is implemented by dedicated hardware or when it runs on modern, multi-core processors. In such scenarios, PMAC1's performance is basically limited by the performance of the underlying block cipher itself. It is important to note, though, that this algorithm has pending patents covering its usage (ROGAWAY, 2001).

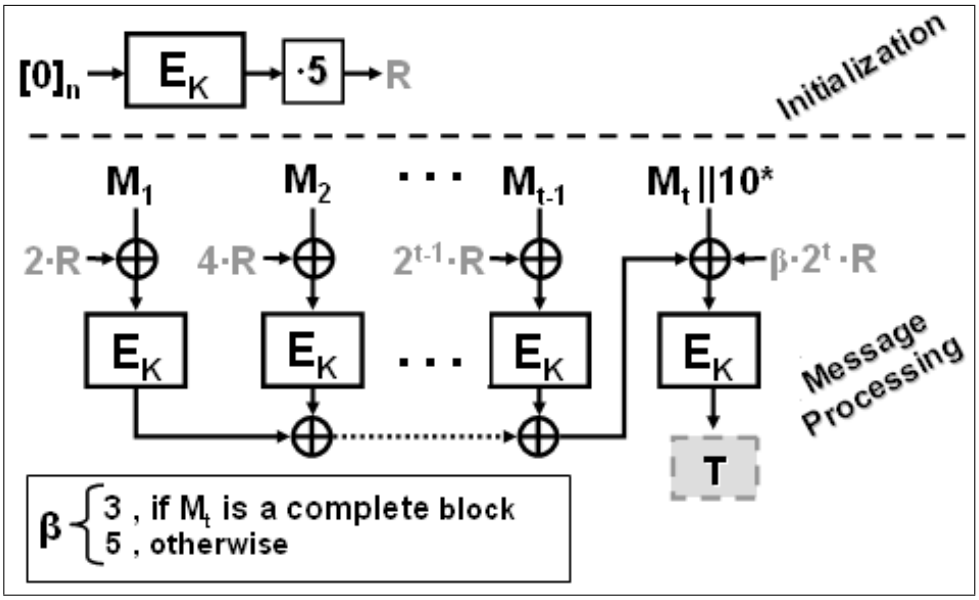

Figure 5: The Parallelizable MAC version 1 (PMAC1) algorithm (ROGAWAY, 2004).

Despite their relative simplicity and good acceptance, a shortcoming with conventional block cipher-based MAC schemes (such as those described in this section) is that they usually invoke the underlying block cipher $t + \varepsilon$ times, where ε stands for a small fixed number. Typically, the ancillary processing in such schemes is much simpler and computationally less expensive than the cost of the block cipher invocations. This is important for space-constrained platforms like smart cards or cheap dedicated hardware, since the extra code storage or circuit area needed for the MAC algorithm is small, say, within 10% of the space needed by E itself. However, the amount of time needed for MACing a block remains lower bounded by E. This can be a minor issue when one adopts a parallelizable MAC function such as PMAC1 and has access to multiple processing units. However, the overhead of such conventional schemes

in highly constrained platforms having a single processing unit becomes an important concern, motivating the development of more efficient designs.

3.2 The Carter-Wegman Design

In 1979, Carter and Wegman introduced the idea of a Universal Hash Family (CARTER; WEGMAN, 1979) and, two years later, the same authors suggested the use of such functions for building MACs (WEGMAN; CARTER, 1981). The basic idea of this construction is to hash the message to be authenticated using a universal hash-function and then encrypt the result with some cipher (in the original paper, the One-Time Pad[2] was used, but later the adoption of block ciphers was suggested (BRASSARD, 1982)), as depicted in Figure 6.

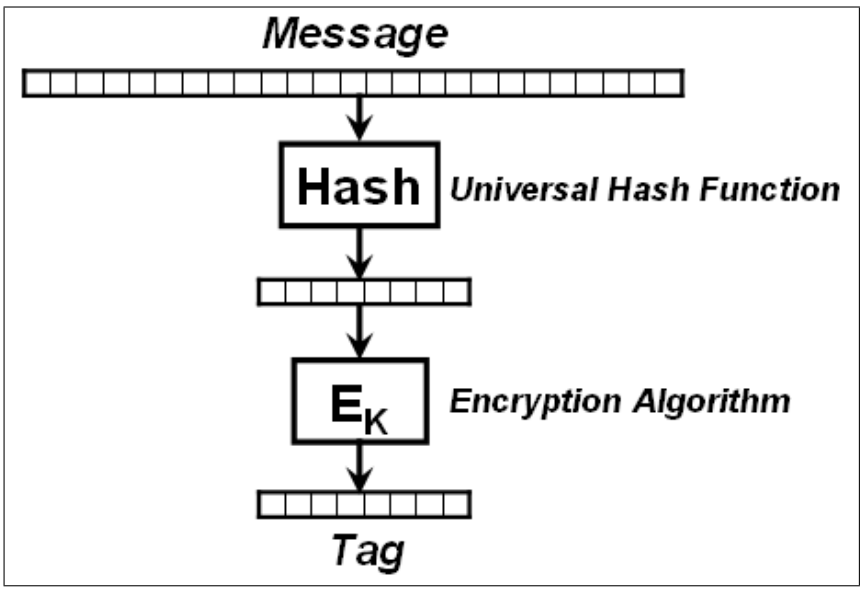

Figure 6: Structure of a Carter-Wegman MAC.

A universal hash-function family is a set of functions endowed with some combinatoric (rather than cryptographic) property. For example, a hash family $\mathcal{H} = \{H : A \rightarrow B\}$ is said to be *ε-almost-universal* if, for any distinct messages $\{M, M'\} \in A$, the probability that $H(M) = H(M')$ is at most ε, when $H \in \mathcal{H}$ is chosen at random.

[2] The One-Time Pad (SHANNON, 1949) approach consists in encrypting a message M simply by XORing it with a $|M|$-bit secret key.

The greatest advantage of universal hash-functions (and the reason why they are adopted in the Carter-Wegman structure) is that they can achieve much higher speeds than cryptographic algorithms. This happens because universal hash-functions need only to satisfy a simple combinatoric property involving collisions in its output, which can be achieved in a fairly simple way. On the other hand, cryptographic functions (e.g., a hash such as SHA-1 (NIST, 2008)) must resemble random functions as much as possible, a property that usually requires several iterations of non-linear and scramble operations for providing enough confusion and diffusion. In fact, due to the difficulty of proving the "randomness" achieved, the design of such algorithms generally involve more iterations than strictly necessary in order to assure a reasonable security margin.

An interesting example of MAC algorithm that adopts the Carter-Wegman design is the *Galois Message Authentication Code (GMAC)* (MCGREW; VIEGA, 2003), which is in fact a specialized, authentication-only variant of the GCM AEAD scheme (MCGREW; VIEGA, 2005) — which will be described in section 3.5. GMAC adopts an internal universal hash-function, GHASH, which is constructed using multiplications by the *hash subkey* $L = E_K([0]_n)$ over the binary finite field GF(2^n). In GMAC, as depicted in Figure 7, this universal hash-function is applied on the input data concatenated with its length in bits (i.e., on $M \,||\, |M|$), and the result is XORed with the encryption of a nonce[3] N, thus yielding the tag T. The nonces may be of any length, since they can be processed using GHASH; however, when using a n-bit underlying block cipher, GMAC offers a significant shortcut for IVs of $(n - 32)$ bits, which are simply padded.

Since $L \cdot (L \cdot M_i \oplus M_{i+1})$ can be obtained by XORing $(L^2 \cdot M_i)$ and $(L \cdot M_{i+1})$, both GHASH and GMAC allow the message to be computed in parallel by means of an exponentiation algorithm.

On the other hand, GMAC has some unusual limitations, not present in the other

[3] A nonce is a non-repeatable IV, i.e., an IV that can be used only once.

MAC algorithms considered in this document. The most remarkable are the use of nonces for its operation (which incurs an additional overhead for each message) and a specification restricted to 64- and 128-bit underlying ciphers.

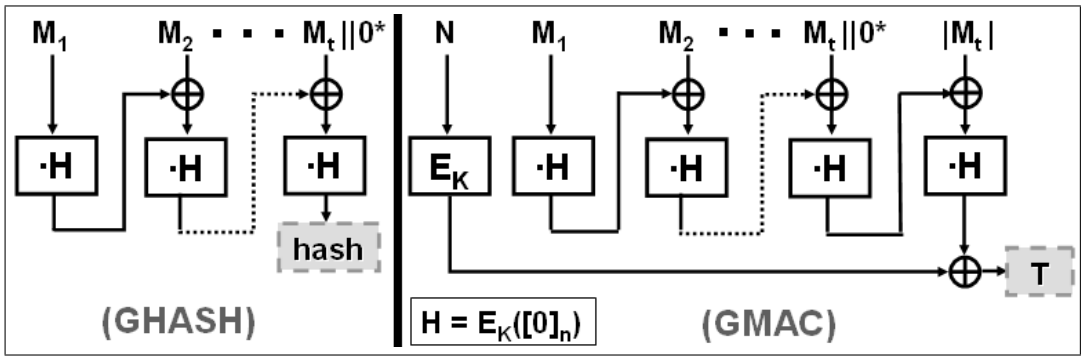

Figure 7: The GHASH universal hash-function (left), used internally by GMAC (right) (MCGREW; VIEGA, 2005).

In contrast with conventional solutions, Carter-Wegman schemes potentially reduce the number of block cipher invocations to ε. This happens when the universal hash can be implemented more efficiently than one E call per message block, which is the case on platforms with large storage spaces for dealing with costly operations such as finite field multiplications and exponentiations. In these cases, the overhead per message block is potentially as low as 10%–25% of a block cipher call, allowing such MAC solutions to provide a high data throughput. For example, GHASH's speed can be considerably increased if the target platform has 64 KiB available for storing key-dependent tables, used for accelerating multiplications over GF(2^n) (MCGREW; VIEGA, 2003). Besides, the Carter-Wegman design provides fully parallelizability, again at the cost of extra memory (for the efficient implementation of exponentiation).

However, space-constrained platforms must resort to space-saving but much slower implementation techniques, which may nevertheless increase code storage or circuit area requirements as compared to conventional schemes and thus constitute an overall efficiency degradation. Hence, in scenarios with such a low margin for processing-memory tradeoffs, the Carter-Wegman design becomes less appealing.

3.3 The ALRED Construction

The ALRED construction (DAEMEN; RIJMEN, 2005a) is also based on an underlying block cipher. However, unlike the traditional schemes described in section 3.1, the ALRED construction provides an interesting trade-off for iterated block ciphers that process data blocks in chunks of fixed bitlength, notably (but not exclusively) the ciphers of the SHARK (RIJMEN et al., 1996) or the SQUARE (DAEMEN; KNUDSEN; RIJMEN, 1997) family, which includes AES (NIST, 2001a).

Using an r-round iterated block cipher operating over n-bit blocks, the idea behind the ALRED design is: (1) split the message into t n-bit blocks $M_1 \| \ldots \| M_t$, padding the last block if necessary; (2) apply the full block cipher to an initial (public) state c, resulting in a state R that is unknown to attackers; (3) combine (via XOR) the first message block with R and process the result using an *iteration function* (which corresponds to $r' < r$ unkeyed rounds of the underlying block cipher), yielding state A_1; (4) iteratively combine the subsequent messages blocks M_i with A_{i-1} and apply the iteration function in order to produce state A_i; (5) finally, encrypt the resulting A_t with the full block cipher in order to generate the authentication tag T, which can be truncated to $\tau < n$ bits if necessary.

One example of MAC function following the ALRED construction is PELICAN (DAEMEN; RIJMEN, 2005b), which uses Rijndael (DAEMEN; RIJMEN, 2002)[4] as underlying block cipher and depicted in Figure 8. PELICAN takes $r' = 4$ unkeyed Rijndael rounds per authenticated block, while a full encryption with this cipher would require, at least, 10 rounds (for 128-bit keys and 128-bit blocks).

In this manner, the cost of processing a message block with the ALRED construction is reasonably lower that the cost of a full encryption (typically 25%–40% of a block cipher call). This can be slower than the Carter-Wegman approach when storage is

[4] The Rijndael block cipher is equivalent to AES (NIST, 2001a), but accepts a wider range of keys and block sizes: any multiple of 32 between 128 and 256 bits.

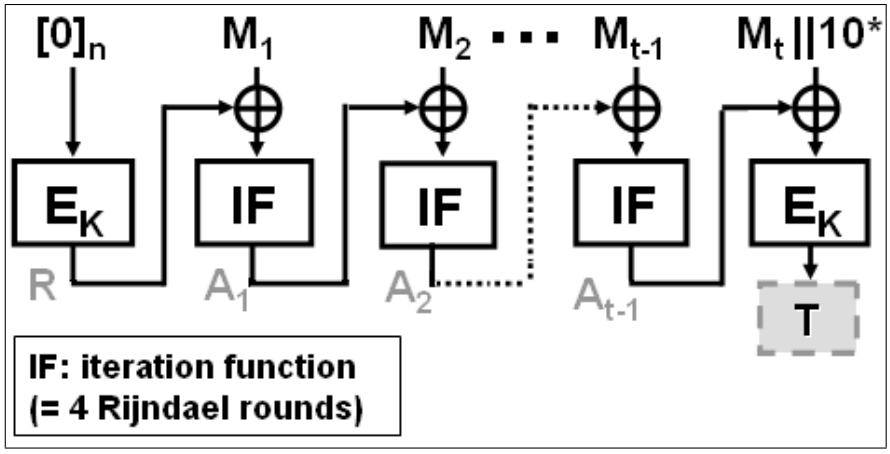

Figure 8: The PELICAN (DAEMEN; RIJMEN, 2005b) MAC function.

abundant, but since the ancillary processing is directly based on the block cipher components, the extra code storage or circuit area is comparable to conventional schemes and thus far smaller than that required by Carter-Wegman solutionss. Therefore, the ALRED construction is a good midway option to either conventional or Carter-Wegman MAC schemes, combining reasonable efficiency with platform flexibility.

On the other hand, the number of messages that can be authenticated with a MAC function based on the ALRED construction is usually lower than the maximum possible with conventional cipher-based MACs. This happens because the iteration function does not display all the properties of the complete cipher and, thus, cannot be considered as secure. Therefore, the algorithm's specification must include the number of messages that can be authenticated under the same key in a secure manner. If this number is not high enough, the algorithm's usefulness may become restricted to a small number of scenarios, such as those in which few messages are exchanged, or those that can afford frequent rekeying operations. We will discuss this issue further in Chapter 5.

Moreover, the original ALRED specification has the disadvantage of being strictly sequential. Nonetheless, it is possible to adapt the ALRED construction in order to create a fully parallelizable MAC algorithm, as is the case of our proposal, MARVIN, described in Chapter 4. Before we describe this algorithm, though, we will analyze some design

techniques for AEAD schemes.

3.4 AEAD and the Generic Composition Design

A straightforward way of designing *Authenticated-Encryption (AE)* schemes is to deploy a cipher (for confidentiality) and a MAC function (for integrity and message authentication) together. Despite its simplicity, some pitfalls exist when adopting this strategy.

One common mistake is *key reuse*, i.e., the usage of a same key for the cipher and the MAC algorithm. This approach is considered a bad practice because, when applied carelessly, it normally leads to flawed designs (BLACK, 2005). In order to circumvent this problem, two different keys (one for each process) may be used, as is the case of the *Generic Composition* design (BELLARE; NAMPREMPRE, 2000).

Concerning the order of the encryption and MAC operations, there are basically three straightforward options for this type of solution (BLACK, 2005):

- *MAC-then-Encrypt (MtE)*: Apply the MAC function over M using key K_1 to create tag T, and then encrypt the resulting pair (M, T) under key K_2. At reception, decrypt to recover (M, T), and then verify T.
- *Encrypt-then-MAC (EtM)*: Encrypt M under key K_2 to create ciphertext C, and then compute $T = MAC_{K_1}(C)$ to yield the pair (C, T). At reception, verify T and then decrypt C to recover M.
- *Encrypt-and-MAC (EaM)*: Encrypt M under key K_2 to create ciphertext C, and then compute $T = MAC_{K_1}(M)$ to yield the pair (C, T). At reception, verify T and then decrypt C to recover M.

In spite of the adopted approach (MtE, EtM or EaM), Generic Composition schemes always make two passes through the data, one aimed at providing privacy

and the other, authenticating it, meaning that they are *Two-Pass* schemes. Nonetheless, if we take a provably-secure encryption scheme and a provably-secure MAC function, each one using independent keys, the best approach for achieving a secure AE scheme is EtM (BELLARE; NAMPREMPRE, 2000). This does not mean that MtE and EaM cannot be made secure, but that this security will often depend on subtle details of the solution, such as happens with the SSH (YLONEN; LONVICK, 2006), which uses a variant of EaM that is indeed secure in most ways (BELLARE; KOHNO; NAMPREMPRE, 2004).

The obvious shortcoming of the Generic Composition is that using two different keys increases the amount of secret information in the system, making their storage, distribution and management more difficult. Another hindrance refers to the performance of the schemes based on the Generic Composition: the cost of the authenticated-encryption process is equivalent to the cost of the encryption plus the cost of the authentication processes, i.e., the resulting scheme is twice as costly as either encryption or authentication alone.

3.5 Beyond Generic Composition

Instead of simply adopting the Generic Composition, a preferred solution would be to develop AE schemes carefully designed for working with a single key, seamlessly and securely merging the cipher's and MAC function's operations in a more efficient way. This motivated the development of *One-Pass* schemes, which make a single pass through the message and perform the needed operations for engendering both privacy and authenticity. Examples include IAPM (JUTLA, 2001), XECB (GLIGOR; DONESCU, 2002) and OCB (KROVETZ; ROGAWAY, 2005). We describe the latter in the following.

In the *Offset CodeBook (OCB)* algorithm[5] , depicted in Figure 9, the message blocks M_i are processed only once: OCB generates the encrypted blocks C_i at the

[5] The OCB algorithm depicted in Figure 9 is the most current form of OCB (KROVETZ; ROGAWAY, 2005), also called OCB 2.0, which adopts optimizations introduced in (ROGAWAY, 2004) and allows for associated data. For the original form of OCB, see (ROGAWAY; BELLARE; BLACK, 2003).

same time that it feeds a "checksum" buffer used for message authentication. The offsets are computed using multiplications over GF(2^n), and then combined with each message block. Only the last block is treated differently: its length is XORed with the corresponding offset, encrypted, and then XORed with this last portion of plaintext, yielding the ciphertext. This stream cipher-like encryption of this last block prevents message expansion. The associated data H, when available, is handled separately by the PMAC1 algorithm (see section 3.1) and simply XORed with the tag generated from the plaintext data. Hence, there is no fixed order for processing message and tag.

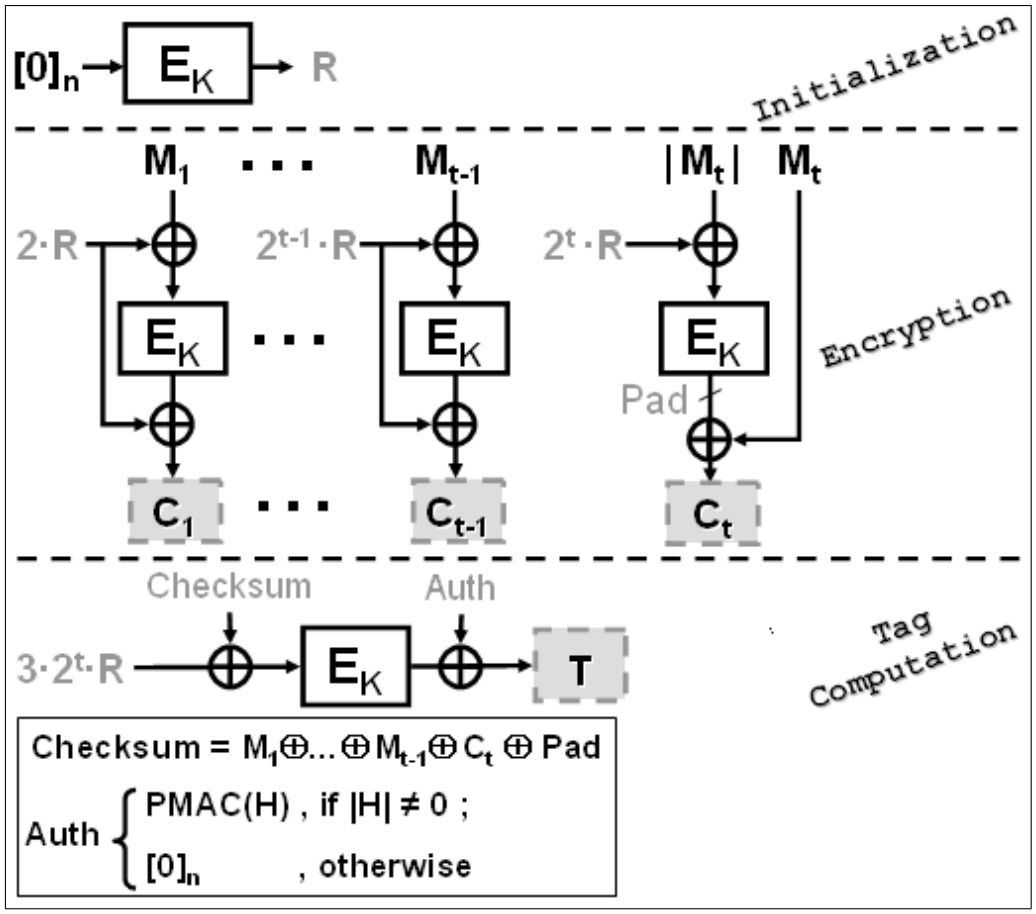

Figure 9: Structure of the One-Pass AEAD scheme OCB (KROVETZ; ROGAWAY, 2005). Note that the PMAC1 (ROGAWAY, 2004) algorithm is used for processing the associated data H when $|H| > 0$.

The decryption in OCB is very similar to encryption, with the main difference that the cipher's decryption function E_K^{-1} is used instead of the encryption function E_K. Finally, note that the tag verification process cannot take place prior to decryption,

since the tag computation depends on the plaintext data.

Unfortunately, due to the importance of these highly-efficient AE solutions, the authors of almost all of them decided to file for patents covering their usage[6] and, perhaps, some of the ideas involved in their construction. It is possible that there are inter-correlations and conflicts between these patents, but the fact is that their simple existence discouraged the wide acceptance of these one-pass schemes. Indeed, such legal complications motivated the research community to develop new, patent-free two-pass AE algorithms. In general, these solutions are not as fast as one-pass schemes, but they offer significant improvements over the Generic Composition construction, such as the usage of a single key. Among these algorithms, we can cite EAX (BELLARE; ROGAWAY; WAGNER, 2004) and GCM (MCGREW; VIEGA, 2005), described in the following.

EAX uses the Counter Mode, described in section 2.2.1, and the $CMAC^u$ algorithm, a simple variant of CMAC (IWATA; KUROSAWA, 2003; NIST, 2005) that concatenates $[u]_n$ (the n-bit representation of u) to the input before its processing. The operation of EAX can be described as follows: first, a nonce N is processed by $CMAC^0$, yielding the result IV for the encryption using Counter Mode (i.e., $IV = CMAC^0_K(N) = CMAC_K([0]_n||N)$); the ciphertext C generated in this manner is authenticated by $CMAC^2$; before or after these operations take place, the Associated Data H is processed by $CMAC^1$; finally, the outputs of the three CMAC building blocks are XORed together yielding the tag T (which can be truncated to $\tau < n$ bits). This process is depicted in figure 10.

EAX's decryption process is very similar: N, H and C are used as the input of the CMAC building blocks and the resulting tag is compared with the one provided by the message sender. If the tag is valid, the message is obtained using the CTR mode; otherwise, it can be discarded without being decrypted. Note that there is no need to

[6] For example, OCB has patents in U.S., allowing its royalty-free use in projects conforming to the GNU General Public License (GPL) (KROVETZ; ROGAWAY, 2005).

implement E_K^{-1} in EAX, since only the encryption function E_K is used.

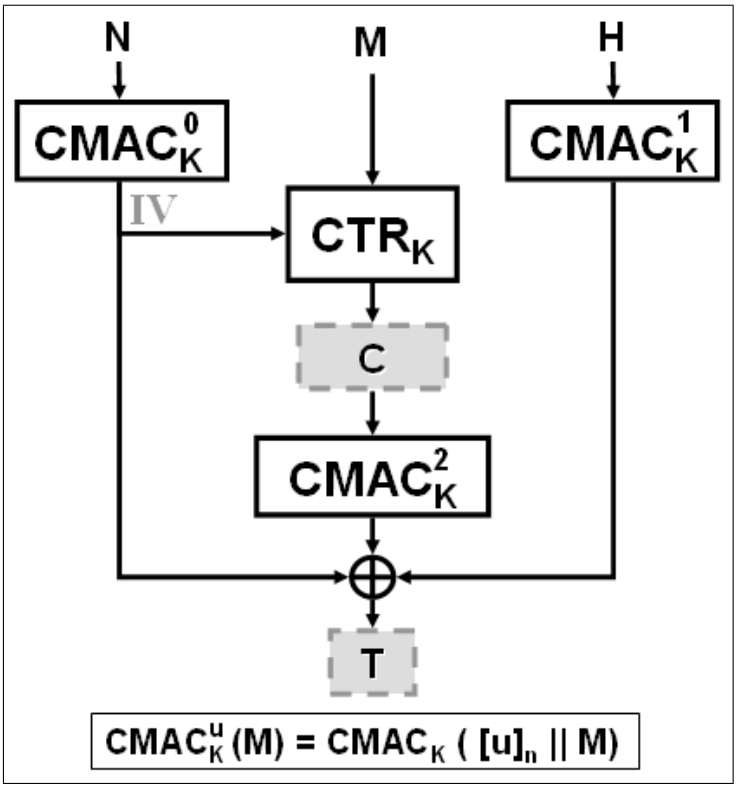

Figure 10: Structure of the Two-Pass AEAD scheme EAX (BELLARE; ROGAWAY; WAGNER, 2004) and its building blocks: CMAC (IWATA; KUROSAWA, 2003) and the CTR mode of operation.

The Galois/Counter Mode (GCM) combines the GHASH algorithm described in section 3.2 with the Counter Mode, and closely follows the Carter-Wegman design. The structure of GCM operating with a 128-bit block cipher is depicted[7] in Figure 11: the message M is encrypted using the CTR mode of operation initialized with $(R + 1)$, where R is computed from a nonce N of any length (as in GMAC IVs of $(n - 32)$ bits are simply padded, avoiding extra processing using GHASH); after the encryption of M, the (possibly padded) ciphertext C is then concatenated with the (possibly padded) associated data H, with $[|H|]_{64}$ and with $[|M|]_{64}$; the whole chunk is then processed by the GHASH algorithm using the hash subkey $L = E_K([0]_{128})$ and the resulting hash value is XORed with $E_K(R)$, yielding the tag T. Like in EAX, the adoption of the CTR

[7] GCM has many minutiae that are omitted in Figure 11 and on the description given in this section. This is done for the sake of simplicity and clarity, since we do not need further details for our purposes. We refer to GCM specification (MCGREW; VIEGA, 2005) for a more detailed discussion on the algorithm.

mode prevents ciphertext expansion and eliminates the need of implementing E_K^{-1}.

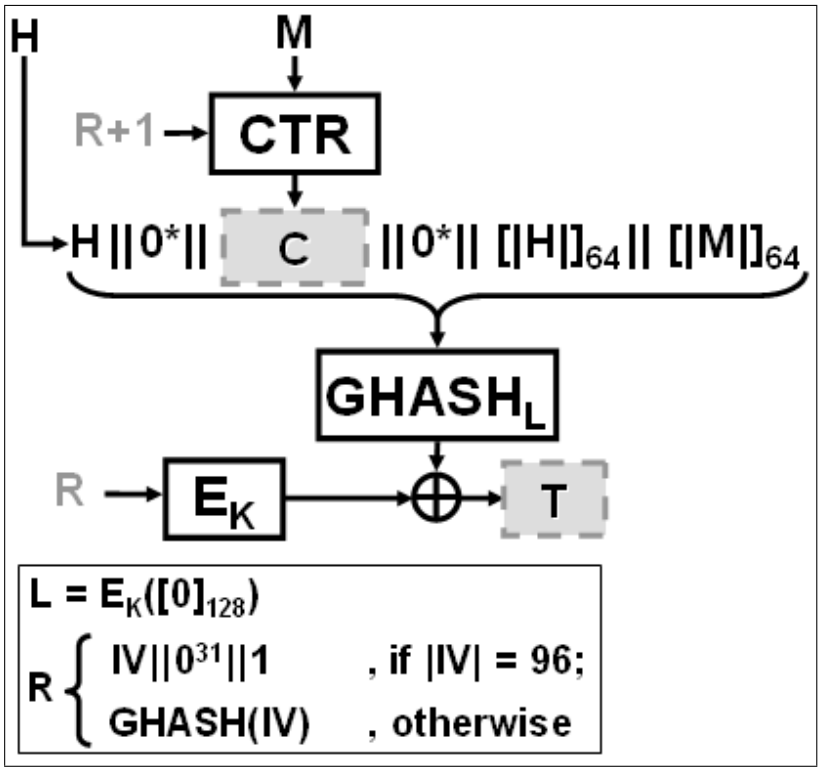

Figure 11: Structure of the Two-Pass AEAD scheme GCM (MCGREW; VIEGA, 2005) and its building blocks: GHASH_L (i.e., GHASH with the hash subkey L) and the CTR mode of operation.

The analysis of the above described schemes reveals some interesting features, but also some limitations.

EAX displays a very straightforward structure, adding a reduced memory overhead when the CMAC algorithm is already available; however, exactly because it adopts CMAC as building block, EAX is not entirely parallelizable and takes one full encryption per authenticated block.

GCM is parallelizable in principle, but the definition of its internal universal hash-function, GHASH, makes it necessary to either implement finite field exponentiation to combine the authentication tags of the message and associated data parts computed in parallel, or else to impose that the full associated data be authenticated before processing the ciphertext, thus preventing complete parallelizability. Moreover, as discussed in section 3.2, the efficiency of Carter-Wegman design usually depends on pre-computed

tables that are usually too large for being implemented in highly constrained devices.

In contrast, OCB is fully parallelizable without the need of expensive arithmetic, and is a one-pass scheme. However, though the encrypted data can be authenticated at low cost, the authentication of associated data still costs one full cipher invocation per block. Besides, unlike EAX and GCM, the OCB algorithm requires the implementation of the decryption function E^{-1}, does not allow the message tag to be verified prior to its decryption, and is covered by patents.

3.5.1 The Cipher-State (CS) Mode

The previously described AE schemes use the underlying block cipher as a black box, not exploring its internal structure. In comparison, the Cipher-State (CS) mode (SANDIA, 2004) uses information from the internal state of an r-round iterated block cipher as shown in Figure 12 and described as follows. First, an offset $R_1 = E_K(IV \oplus K) \oplus K$ is generated; this offset is used as the seed of the subsequent offsets, which are obtained from R_1 by means of multiplications over GF(2^n): $R_i = R_{i-1} \cdot x$. Each offset R_i is combined with message block M_i prior and after encryption, yielding the ciphertext block C_i. Half-way through each block encryption (i.e., after $r/2$ rounds), though, the state A_i corresponding to the block M_i is tapped and added to an accumulator A, which is then multiplied by x; this multiplication is necessary in order to assure the non-commutativity of the various A_i. Finally, the authentication tag is computed as $T = E_K(A \oplus R_{t+1}) \oplus R_{t+1}$.

The main advantage of the CS mode is that it behaves as a One-Pass scheme: whenever the data must be encrypted for confidentiality purposes, the overhead introduced by the authentication process is minimal. On the other hand, when one needs only authentication, the total overhead is 50% of a full encryption per block, which is half the cost of a conventional MAC function, but still higher than a solution following the ALRED structure (e.g., 10% higher when the underlying cipher is AES).

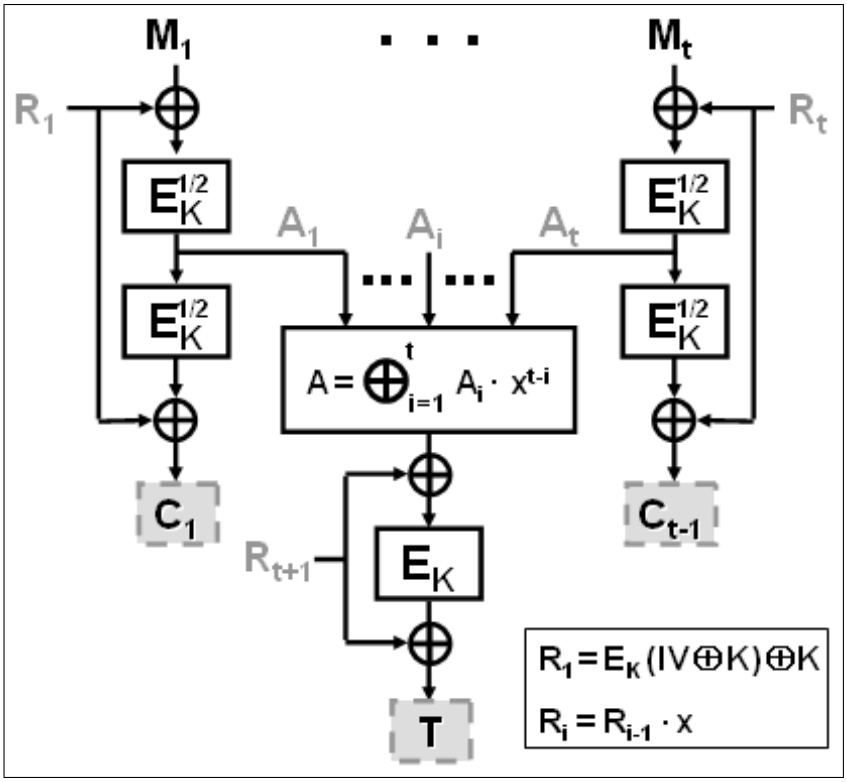

Figure 12: The Cipher-State (CS) mode of operation for iterated block ciphers. Here, $E_K^{1/2}$ stands for the encryption with $r/2$ rounds of an r-round iterated block cipher.

Another issue with the CS mode is that, such as OCB, it requires the deployment of both the encryption and decryption algorithms. This could be a lesser concern if one could adopt an involutional underlying block cipher[8] such as those from the CURUPIRA family; however, according to the authors of the CS mode, ciphers with such a characteristic cannot be used in this scheme, a restriction not present in any other solution described in this document. Finally, the tag verification using the CS mode requires the partial decryption of the data.

3.6 Summary and Discussion

In this chapter, we presented and illustrated some relevant techniques for constructing MAC and AEAD solutions. More specifically, we described the approaches

[8] In an involutional cipher, the encryption and decryption processes are identical except by the key schedule, which leads to a reduced memory footprint when both encryption and decryption processes are needed.

adopted in conventional designs, in the Carter-Wegman structure, and in the ALRED construction, discussing their advantages and limitations. We have also described how a cipher and a MAC function can be combined for creating AEAD schemes, either in the Generic Composition approach or in more efficient constructions.

For the development of our MAC and AEAD algorithms, conventional and Carter-Wegman designs are not very appealing due to the constraints of our target platforms: the former would require a full encryption per authenticated block, while the latter involves fairly complex operations whose performance depends on the availability of large pre-computed tables.

For our AEAD solution, a design based on the CS mode could be a good option but, as discussed in this chapter, the authentication of plain data using this scheme would be less efficient than a scheme based on the ALRED structure. Moreover, the CS mode cannot be used together with involutional ciphers and requires the cipher's decryption algorithm to be implemented; this does not only prevents the deployment of some efficient ciphers, such as the CURUPIRA, but also potentially doubles the total amount of memory needed, since the decryption process in non-involutional ciphers typically takes about as much coding space as the encryption process.

Finally, the reason why a One-Pass strategy was not adopted for our AEAD scheme resides mainly on the existence of patent issues covering such solutions: although the performance achieved by an algorithm combining MARVIN and an OCB-like solution could be interesting from the point of view of performance, we preferred to avoid entering in this "war field" because, historically, the existence of patents over cryptographic mechanisms have leaded many of algorithms to be rejected by standardization organisms in spite of their efficiency[9].

For the reasons stated above and along this section, which are summarized in Ta-

[9] One example: during the choice of the security mechanisms to be adopted by the IEEE 802.11i (IEEE, 2004) standard, the CCM (NIST, 2004) algorithm was chosen as a mandatory component despite lots of criticism (ROGAWAY; WAGNER, 2003), while OCB was relegated to an optional component.

Table 6: Criteria used in the selection of a suitable design technique for developing MARVIN and LETTERSOUP. We use highlighted cells to indicate an issue with the technique.

	Cost for M	Cost for H	Security	Applicability	Patents
Conventional	1E/block	1E/block	Provable	MAC, AEAD	No
Carter-Wegman	< 1E/block iff large LUTs	< 1E/block iff large LUTs	Provable	MAC, AEAD	No
Alred	< 1E/block	< 1E/block	Provable	MAC, AEAD	No
CS-Mode	1E	unsupported	???	AE	No
"OCB"	1E	1E/block	Provable	AEAD	Yes

ble 6 we have decided on the ALRED structure as the basis for the development of our MAC and AEAD algorithms. Moreover, concerning the development of our AEAD solution, we focus our efforts on two-pass strategies in order to avoid patent issues. In the next chapters, we will describe and evaluate the resulting algorithms.

4 MARVIN AND LETTERSOUP

In the previous chapter, we described some interesting design techniques for MACs and AEADs. The analysis of the different strategies presented lead to the adoption of the ALRED structure as the basis for the development of our algorithms. This choice was motivated by the ALRED construction's potential on providing a fairly fast MAC function with reduced memory footprint. The main disadvantage of the original ALRED structure remains thus the strictly sequential nature of the resulting algorithms.

In this chapter, we will describe the MARVIN algorithm in details in this chapter and then show how MARVIN can be used for the construction of a novel AEAD scheme, LETTERSOUP. We show that both proposals are based on a modified version of the ALRED construction, adopting a structure that closely follows the Randomize-then-Combine paradigm. This approach was introduced by Bellare and Micciancio in the context of incremental hash function design (BELLARE; MICCIANCIO, 1997), and its advantage is twofold: algorithms following this strategy are both parallelizable and incremental. While the parallelizability goal shall be clear at this point, the concept of incrementality can be better explained through an example: suppose one computes the MAC of a message M, resulting in the tag T, and later this message is modified, becoming M'. With an incremental MAC function, the new tag M' can be computed from T at a cost proportional to the amount of alteration made in M (e.g., addition or modification of blocks), thus avoiding the need of computing T' completely from the scratch. This feature is particularly interesting when we consider huge messages (e.g., an entire hard disk). We note that the incrementality can be obtained more easily when one

can process each block in an independent manner (otherwise, the modification of one block M_i would affect all other blocks M_j that depend on M_i), which is the case of parallelizable algorithms.

Before we describe our algorithms, though, we provide some additional notation and introduce a new concept that is of central importance for our algorithms: the Square-Complete Transforms (SCTs). We note that some of the notation presented in the following has already been introduced in previous sections, but it is repeated here for completeness and clarity.

4.1 Preliminaries and Notation

In what follows, $E : \{0,1\}^k \times (\{0,1\}^w)^b \rightarrow (\{0,1\}^w)^b$ stands for an iterated block cipher structured as a Substitution-Permutation Network (SPN)[1] with a key size of k bits and a block size of $n = bw$ bits. Here we assume that the n-bit data blocks are organized and processed as b bundles of w bits each (typically $w = 4$ or $w = 8$, meaning that the data is organized in nibbles or bytes, respectively); we say that E is a b-bundle block cipher.

A substitution table or S-box is a nonlinear mapping $S : \{0,1\}^w \rightarrow \{0,1\}^w$. S-boxes are the usual way to introduce non-linearity in iterated block ciphers.

We represent the finite field with 2^n elements as $\mathrm{GF}(2^n) \equiv \mathrm{GF}(2)[x]/p(x)$, for an irreducible polynomial $p(x)$ of degree n such that x^w is a generator[2] modulo $p(x)$. We use the `typewriter font` and the prefix `0x` for indicating hexadecimal representation (e.g., `0xFF` represents the decimal value 255).

The length of a bit-string B is denoted $|B|$. If $|B| \leqslant n$, then $\mathbf{rpad}(B) \equiv B \,||\, 0^*$ (resp.

[1] An SPN is a structure where a substitution layer (a non-linear function that provides confusion) and a permutation layer (which scramble bits, providing diffusion) are interleaved and iteratively applied to the data. See Appendix B for further details.

[2] We say that a polynomial $g(x)$ is a *generator* of $\mathrm{GF}(2^n) \equiv \mathrm{GF}(2)[x]/p(x)$ if the powers of $g(x)$ assume every possible non-zero value in $\mathrm{GF}(2)[x]/p(x)$.

lpad$(B) \equiv 0^* \,||\, B$) is the bit-string consisting of B right-padded (resp. left-padded) with as many binary zeros as needed (possibly none) to produce an n-bit string. We write **bin**(ℓ) for the binary representation of the integer ℓ if $\ell > 0$, or the empty string if $\ell = 0$ (not to be confused with $[\ell]_n$, the n-bit representation of the value ℓ, which is equivalent to **lpad**(**bin**(ℓ))). $B[\tau]$ denotes the string consisting of the leftmost τ bits of B. Finally, if B is a bit-string of fixed length w, the expressions "$B \ll m$", "$B \gg m$", and "B **rotl** m" stand respectively for B left-shifted, right-shifted, and left-rotated by m bits (with its length preserved, and padded with bits '0' where needed).

4.1.1 Square-Complete Transforms (SCTs)

The intuition behind the construction (and also the name) of Square-Complete Transforms (SCTs) comes from the study of techniques to prevent Differential Cryptanalysis (BIHAM; SHAMIR, 1991), which have an important role in the security analysis of the ALRED construction. More specifically, the design of SCTs comes from the observation that 4 rounds of a cipher such as SQUARE (DAEMEN; KNUDSEN; RIJMEN, 1997) create a high level of confusion and diffusion, which in practice helps preventing this modality of attack.

Formally, an SCT can be defined as:

Definition 2. *Let E be a b-bundle iterated block cipher. A* Square-Complete Transform *(*SCT *for short) given by $\blacksquare : (\{0,1\}^w)^b \to (\{0,1\}^w)^b$ is the shortest sequence of unkeyed rounds of E such that any differential characteristic of E spanning that number of rounds contains at least b active S-boxes.*

In the context of Differential Cryptanalysis, an *active* S-Box is an S-Box that is part of a differential trail. In order to be successful, such attacks require the construction of trails having a high enough probability. Since a trail's probability decreases with the number of active S-Boxes in it, this class of cryptanalysis can be effectively prevented

by assuring a high number of active S-Boxes in any possible trail[3].

Although the above formal definition of SCTs is actually given in this document, they are the kernel of message authentication codes following the ALRED structure. Actually, the original ALRED specification (DAEMEN; RIJMEN, 2005a) gives some clues on the number of rounds needed for constructing an iteration function with a certain security level, considering mainly the resistance of the structure against extinguishing differentials, which is analogous to the resistance against differential cryptanalysis. The SCT definition presented here follows exactly the same reasoning and, in practice, should provide a reasonably good security level for most modern block ciphers, especially Substitution-Permutation Networks (SPNs) following the Wide Trail Strategy (DAEMEN; RIJMEN, 2001). We will discuss such security issues further in section 5.3. For now, let us consider some practical examples of SCTs.

Ciphers of the SQUARE (DAEMEN; KNUDSEN; RIJMEN, 1997) family have block size $b = (d-1)^2$ bundles (usually, bytes) where d is the distance of a certain linear error correcting code. One can show (DAEMEN; RIJMEN, 2002) that any 4-round differential characteristic of such a cipher contains at least d^2 active S-boxes, even though there are 3-round differential characteristics containing no more than $2d-1$ active S-boxes. This result is a direct application of the Wide Trail Strategy, and the main reason why Pelican adopts 4-round SCTs derived from these ciphers[4] . The same number of rounds is also necessary for block ciphers like CURUPIRA and BKSQ (DAEMEN; RIJMEN, 1998), which also follow the Wide Trail Strategy and are closely related to SQUARE, but operate on blocks of different size (namely, $b = d^2 - d$ bundles).

Similarly, ciphers of the SHARK (RIJMEN et al., 1996) family have block size $b = d-1$ and any 2-round differential characteristic of such a cipher contains at least d active S-boxes. As a result, SCTs derived from these ciphers would require only 2-

[3] For a comprehensive discussion on differential cryptanalysis, we recommend the reading of (HEYS, 2002).

[4] The Alpha-MAC scheme is somewhat different in that it adopts 1-round transforms, applying them to partial blocks of size $d-1$.

rounds.

Other ciphers are handled analogously, although determining the number of rounds of an SCT may not be straightforward. For instance, PRESENT (BOGDANOV et al., 2007) is a 16-bundle cipher known to contain at least 10 active S-boxes at each 5 rounds; hence, an SCT necessarily spans between 6 and 10 rounds, but computing the exact number (namely, 7 rounds) is somewhat involved. The same holds for NOEKEON (DAEMEN et al., 2000), which is a 32-bundle cipher containing at least 20 active S-boxes at each 4 rounds; in this case an SCT would span between 5 and 8 rounds, but the exact number is not currently known.

4.2 The MARVIN Message Authentication Code

In a construction following the Randomize-then-Combine Paradigm (BELLARE; MICCIANCIO, 1997), (1) the message M to be processed is broken into a sequence of t blocks $M_1 \| \ldots \| M_t$; then, (2) each block M_i is combined with an encoding block R_i and (3) processed by a *randomizer*; finally, (4) all blocks are combined together using a *combining operation*. The combination with the encoding block (step 2) is necessary to prevent the interchangeability of any two blocks M_i and $M_{j\neq i}$. As previously stated, this Randomize-then-Combine approach allows the construction of parallelizable and incremental MAC functions, and that is the main reason why it has been adopted.

MARVIN adopts a variant of Krawczyk's $h_p(M)$ cryptographic Cyclic Redundancy Check (CRC) (KRAWCZYK, 1994) to generate secret offsets R_i, which are XORed with the message blocks (the offsets act, thus, as encoding blocks); the offset seed is generated using a constant c left-padded to n bits. A square-complete transform is the randomizer used to mix the offsets with message blocks. These operations are applied to each message block in parallel and accumulated via XOR. The encrypted (and then possibly truncated) accumulation result defines the resulting authentication tag.

Algorithm 4.1 describes the MARVIN message authentication code, written as MARVIN(M, K, τ) or simply MARVIN$_K(M, \tau)$. Additionally, Figure 13 illustrates the algorithm's structure.

For the sake of comparison, in the incremental hash functions originally described by Bellare and Micciancio (BELLARE; MICCIANCIO, 1997) we have: the encoding blocks are given by $R_i = i$, and are prepended to the message in step (2); the randomizer is a collision-free function derived from a standard hash function; and the combining operation is either group multiplication or addition (the XOR operation does not provide a secure algorithm in the original construction because there is not a final encryption step to protect the accumulated result).

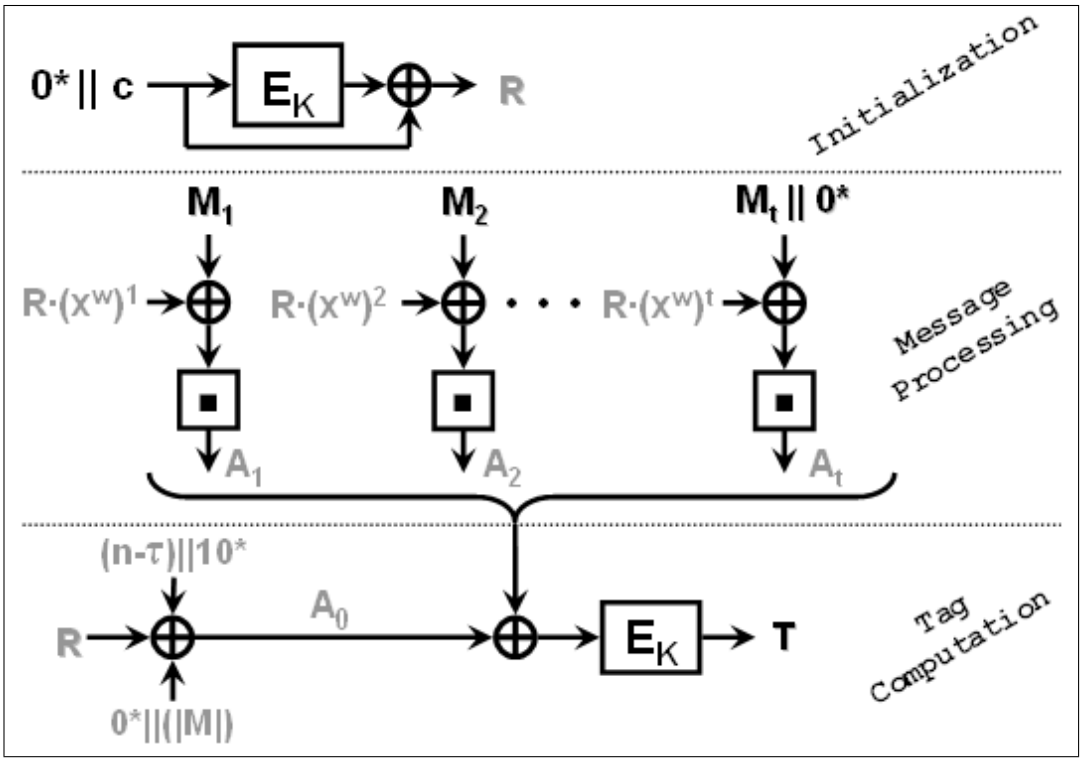

Figure 13: The MARVIN Message Authentication Code.

4.2.1 On the Choice of the Constant *c*

To ensure that the offsets O_i are non-zero it is necessary that R itself be non-zero. Assuming that the block cipher E has no weak key $\tilde{K}$ such that $E_{\tilde{K}}(B) = B$ for any

Algorithm 4.1 The MARVIN message authentication code

INPUT: M ▷ message to authenticate, with $|M| < 2^n$.
INPUT: K ▷ MAC key.
INPUT: τ ▷ desired MAC length ($\tau \leqslant n$).
OUTPUT: T ▷ MAC of message M under key K, truncated to τ bits.

1: Partition the message as $M = M_1 \,||\, \ldots \,||\, M_t$, where $|M_i| = n$ for $i = 1, \ldots, t-1$, and $0 \leqslant |M_t| \leqslant n$ with $|M_t| = 0$ iff $|M| = 0$.
2: $R \leftarrow E_K(\mathbf{lpad}(c)) \oplus \mathbf{lpad}(c)$
3: **for** $i \leftarrow 1$ **to** t **do** ▷ incrementally or in parallel
4: $\quad O_i \leftarrow R \cdot (x^w)^i$
5: $\quad A_i \leftarrow \blacksquare(\mathbf{rpad}(M_i) \oplus O_i)$ ▷ Padding affects only M_t
6: **end for**
7: $A_0 \leftarrow R \oplus \mathbf{rpad}(\mathbf{bin}(n-\tau) \,||\, 1) \oplus \mathbf{lpad}(\mathbf{bin}(|M|))$
8: $A \leftarrow \bigoplus_{i=0}^{t} A_i$ ▷ incrementally or in parallel
9: $T \leftarrow E_K(A)[\tau]$
10: **return** T

block B, or that finding such a key is infeasible for any fixed block B, it is clear that $E_K(B) \oplus B \neq [0]_n$. All things being equal, the simple choice $c = 0$ would be suitable. Indeed, many MAC constructions adopt the encryption of a block filled with '0' bits, $E_K([0]_n)$, as the seed for offset generation.

However, it is essential that R remain secret, yet disclosure of $E_K([0]_n)$ for key confirmation purposes is widespread if not recommended in many scenarios, especially financial applications. It is therefore advisable to choose $c \neq 0$ for the sake of robust deployment in real-world systems. Any non-zero constant would seem to do equally well, a small (say, 1-byte) value being preferred not to impart efficiency on constrained processors. The actual value is $c =$ `0x2A` (42, in decimal representation), which was chosen largely at random, although it might be better justified in certain contexts (ADAMS, 1979).

4.2.2 Message Padding

Perhaps the most popular padding method used in MAC schemes consists of replacing the last block M_t by $\mathbf{rpad}(M_t \,||\, 1)$ when $|M_t| < n$, and adding a new block when $|M_t| = b$. This method ensures that no two distinct messages are padded to the same

string, which would trivially lead to a MAC collision (e.g., this guarantees that the complete block $0^{n-1}1$ and the incomplete padded block 0^{n-1} are not interchangeable). A related method avoids adding an entire new block when $|M_t| = n$ by handling the last block in a different manner, for instance by using a different offset (as in CMAC).

MARVIN adopts a different approach. The last block is simply completed with zero bits; however, the message length and also the truncation length are separately included in the tag computation. This avoids both the block addition and the need for a different block treatment.

4.2.3 Offset Generation

Computing offsets involves multiplication in a finite field GF(2^n), and hence has to be designed carefully to avoid unnecessary burden on a variety of platforms. This is particularly important for scenarios where the tag computation must be carried out incrementally instead of in parallel.

The original Krawczyk's proposal involves multiplication by the polynomial x^n and hence a shift by the full block length n followed by an n-step reduction, which is likely to be overkill. The variant employed in CS mode (SANDIA, 2004) and PMAC1 (ROGAWAY, 2004) consists of multiplication by the polynomial x and hence a shift by a single bit; although the ensuing reduction is very simple, this choice destroys bundle alignment and incurs costly operations on many platforms, particularly 8-bit smart cards and SSE2 processors. Hence, such drawbacks are clearly undesirable.

MARVIN adopts the middle way: since the underlying block cipher organizes the data in w-bit bundles (supposedly fit for efficient implementation on the target platform), the most natural shift distance is w for preserving bundle alignment. Therefore, the offset generation involves multiplication by x^w. The polynomial reduction is potentially more complex than that incurred by 1-bit shift, unless the polynomial itself is chosen carefully, as discussed in the following.

4.2.3.1 Efficient Multiplication by x^w

As previously stated, the offsets used in MARVIN are incrementally defined as $O_0 \leftarrow R$, $O_{i+1} \leftarrow O_i \cdot x^w$. Let $\mathrm{GF}(2^n) = \mathrm{GF}(2)/p(x)$ where $p(x)$ is a primitive pentanomial over GF(2) such that x^w is a primitive root of $p(x)$. We choose a pentanomial basis representation because primitive pentanomials are available for all n of practical interest (IEEE, 2000); though primitive trinomials would result in a better performance, they are seldom available and do not exist at all in the important case where $8 \mid n$ (MENEZES; OORSCHOT; VANSTONE, 1999).

One can implement multiplication by x^w in an efficient manner thanks to the following theorem:

Theorem 1. *Let $p(x) = x^n + x^{k_3} + x^{k_2} + x^{k_1} + 1$ be a primitive pentanomial of degree $n = bw$ over* GF(2) *such that $k_3 > k_2 > k_1$, $k_3 - k_1 \leqslant w$, and either $w \mid k_3$ or $w \mid k_1$. Then multiplication by x^w in $\mathrm{GF}(2^n) = \mathrm{GF}(2)[x]/p(x)$ can be implemented with no more than 5 XORs and 4 shifts on w-bit words. Moreover, if 2×2^w bytes of storage are available, the cost drops to no more than 2 XORs on w-bit words and 2 table lookups.*

Proof: For $u = \bigoplus_{d=0}^{n-1} u_d x^d \in \mathrm{GF}(2^n)$ let $U_i = u_{wi+w-1}x^{w-1} + \cdots + u_{wi}$, $i = 0, \ldots, b-1$, so that $u = U_{b-1}x^{w(b-1)} + U_{b-2}x^{w(b-2)} + \cdots + U_0$, which for brevity we write $u = (U_{b-1}, \ldots, U_0)$. Then one can compute $u \cdot x^w$ as:

$$
\begin{aligned}
&(U_{b-1}x^{w(b-1)} + U_{b-2}x^{w(b-2)} + \cdots + U_0) \cdot x^w = \\
&U_{b-1}x^n + U_{b-2}x^{w(b-1)} + \cdots + U_0x^w = \\
&U_{b-2}x^{w(b-1)} + \cdots + U_0x^w + U_{b-1}(x^{k_3} + x^{k_2} + x^{k_1} + 1) = \\
&(U_{b-2}, \ldots, U_0, U_{b-1}) \oplus U_{b-1}(x^{k_3} + x^{k_2} + x^{k_1}).
\end{aligned}
$$

Assume that $k_1 = wk$ for some k; the case $w \mid k_3$ is handled analogously. Thus:

$$u \cdot x^w = (U_{b-2}, \ldots, U_0, U_{b-1}) \oplus U_{b-1}(x^{k_3-k_1} + x^{k_2-k_1} + 1)x^{wk}.$$

Let $\deg(u)$ denote the degree of polynomial u. Since $\deg(U_{b-1}) \leqslant w-1$ and $\deg(x^{k_3-k_1} +$

$x^{k_2-k_1}+1) \leqslant w$, their product is a polynomial of degree not exceeding $2w-1$, and hence it fits in two w-bit words for any value of U_{b-1}. Besides, multiplication of this value by x^{wk} corresponds to simply displacing it k words to the left. In this manner, we can write $u \cdot x^w = (U_{b-2}, \ldots, U_k \oplus T_1[U_{b-1}], U_{k-1} \oplus T_0[U_{b-1}], \ldots, U_0, U_{b-1})$, where:

$$\begin{aligned} T_0[U] &\equiv (U \ll (k_3 - k_1)) \oplus (U \ll (k_2 - k_1)) \oplus U; \\ T_1[U] &\equiv (U \gg (w - (k_3 - k_1))) \oplus (U \gg (w - (k_2 - k_1))). \end{aligned}$$

The values of T_0 and T_1 can be either computed on demand or else precomputed and stored in two 2^w-entry tables. By direct inspection, one easily sees that the computational cost is that stated by the theorem. □

Depending on the block size of the underlying cipher, a different polynomial may be used by Marvin. Table 7 lists some polynomials that are especially well suited for implementation according to the above theorem, for different block sizes.

Table 7: Suitable polynomials for multiplication by x^w, for different block sizes.

Block Size (bits)	Polynomial
64	$x^{64} + x^8 + x^7 + x^5 + 1$
96	$x^{96} + x^{16} + x^{13} + x^{11} + 1$
128	$x^{128} + x^{29} + x^{27} + x^{24} + 1$
144	$x^{144} + x^{56} + x^{53} + x^{51} + 1$
160	$x^{160} + x^{45} + x^{43} + x^{40} + 1$
192	$x^{192} + x^{48} + x^{45} + x^{43} + 1$
224	$x^{224} + x^{39} + x^{33} + x^{32} + 1$
256	$x^{256} + x^{104} + x^{99} + x^{97} + 1$

As an illustration, we describe the multiplication by x^8 using the above polynomials and commonplace block sizes, assuming that the data organized in 8-bit bundles:

- $x^{64} + x^8 + x^7 + x^5 + 1$:

  $(U_7, \ldots, U_0) \cdot x^8 = (U_6, \ldots, U_1, U_0 \oplus T_1[U_7], T_0[U_7])$, where

  $T_1[U] \equiv U \oplus (U \gg 1) \oplus (U \gg 3)$,

  $T_0[U] \equiv U \oplus (U \ll 7) \oplus (U \ll 5)$.

- $x^{96} + x^{16} + x^{13} + x^{11} + 1$:

  $(U_{11}, \ldots, U_0) \cdot x^8 = (U_{10}, \ldots, U_1 \oplus T_1[U_{11}], U_0 \oplus T_0[U_{11}], U_{11})$, where

  $T_1[U] \equiv U \oplus (U \gg 3) \oplus (U \gg 5)$,

  $T_0[U] \equiv (U \ll 5) \oplus (U \ll 3)$.

- $x^{128} + x^{29} + x^{27} + x^{24} + 1$:

  $(U_{15}, \ldots, U_0) \cdot x^8 = (U_{14}, \ldots, U_3 \oplus T_1[U_{15}], U_2 \oplus T_0[U_{15}], U_1, U_0, U_{15})$, where

  $T_1[U] \equiv (U \gg 3) \oplus (U \gg 5)$,

  $T_0[U] \equiv (U \ll 5) \oplus (U \ll 3) \oplus U$.

It is interesting to notice that, in fact, the circular byte permutation resulting from the multiplication by x^8 does not need to be effectively implemented: the same effect can be achieved if we keep track of the index corresponding to the most significant byte of U, denoted i_{msb}. Consequently, by using the i_{msb}-th byte as the first byte of U in every calculation, it suffices to update i_{msb} after each multiplication by x^w. Using this strategy, word permutations within a block can be performed essentially for free.

For the sake of comparison, algorithm 4.2 describes bytewise multiplication by x in GF(2^n), assuming that $\deg(p(x) - x^n) < 8$ (so that the mask computed at the beginning of the algorithm fits one byte). One sees that the cost is b XORs, $2b - 1$ shifts, and one bit test. In other words, the cost of multiplying by x is proportional to the block size, while that of multiplying by x^w is constant.

Algorithm 4.2 Bytewise multiplication by x in GF(2^n).

INPUT: B ▷ n-bit word to multiply by x.
OUTPUT: $B \cdot x$.

1: **if** most significant bit of $B_{b-1} = 1$ **then** $v \leftarrow p(x) - x^n$ **else** $v \leftarrow 0$
2: **for** $i \leftarrow b - 1$ **downto** 1 **do**
3: $\quad B_i \leftarrow (B_i \ll 1) \oplus (B_{i-1} \gg 7)$
4: **end for**
5: $B_0 \leftarrow (B_0 \ll 1) \oplus v$
6: **return** B

Finally, we point out that these observations concerning 8-bit scenarios do not

impair implementation on platforms that are not byte-oriented. For example, if a block fits one machine word, then clearly multiplication by x^w incurs only 1 rotation, 1 AND, 3 shifts, and 3 XORs, as illustrated by algorithm 4.3. Then again, we can improve the computation performance by means of a single 2^w-word table $T_w[U] = (U \ll k_3) \oplus (U \ll k_2) \oplus (U \ll k_1)$, leading to the a cost of 2 XORs and 2 table lookups (see Theorem 1).

Algorithm 4.3 Wordwise multiplication by x^w in GF(2^n).

INPUT: V ▷ n-bit word to multiply by x^w.
OUTPUT: $V \cdot x^w$.
1: $V \leftarrow V$ **rotl** w ; $R \leftarrow V$ & `0xFF`
2: $V \leftarrow V \oplus (R \ll k_3) \oplus (R \ll k_2) \oplus (R \ll k_1)$
3: **return** V

4.3 The LetterSoup AEAD Scheme

To date, many AE schemes have been proposed and analyzed in terms of security and performance in different platforms. Some examples were presented in Chapter 3, which discussed some drawbacks in EAX, GCM, OCB and the CS mode, including: parallelizability problems (mostly EAX, but also GCM), the presence of complex operations (GCM), the cost of one full encryption per authenticated block (EAX and, for associated data, OCB), the existence of patents (OCB), the need of both encryption and decryption algorithms (OCB and CS mode), and incompatibility with involutional block ciphers (CS mode).

It turns out that one can avoid all these drawbacks by simply substituting CMAC by Marvin in the definition of EAX. However, an even more interesting approach is to use Marvin in a novel AEAD construction. The idea comes from the observation that the Linear Feedback Shift Register Counter (LFSRC) mode for block ciphers (KRAWCZYK, 1994) (see Algorithm 4.4) could be based on, and benefit from, Marvin's choice of offsets. In such a construction, it is possible to maintain a single counter and use it both for the encryption and authentication sub-processes, leading to a memory- and

processing-efficient joint solution. Furthermore, the LFSRC mode is very interesting in constrained scenarios: it is not only length-preserving ($|C| = |M|$) but also involutional (applying it twice recovers the original plaintext, without implementing E_K^{-1}).

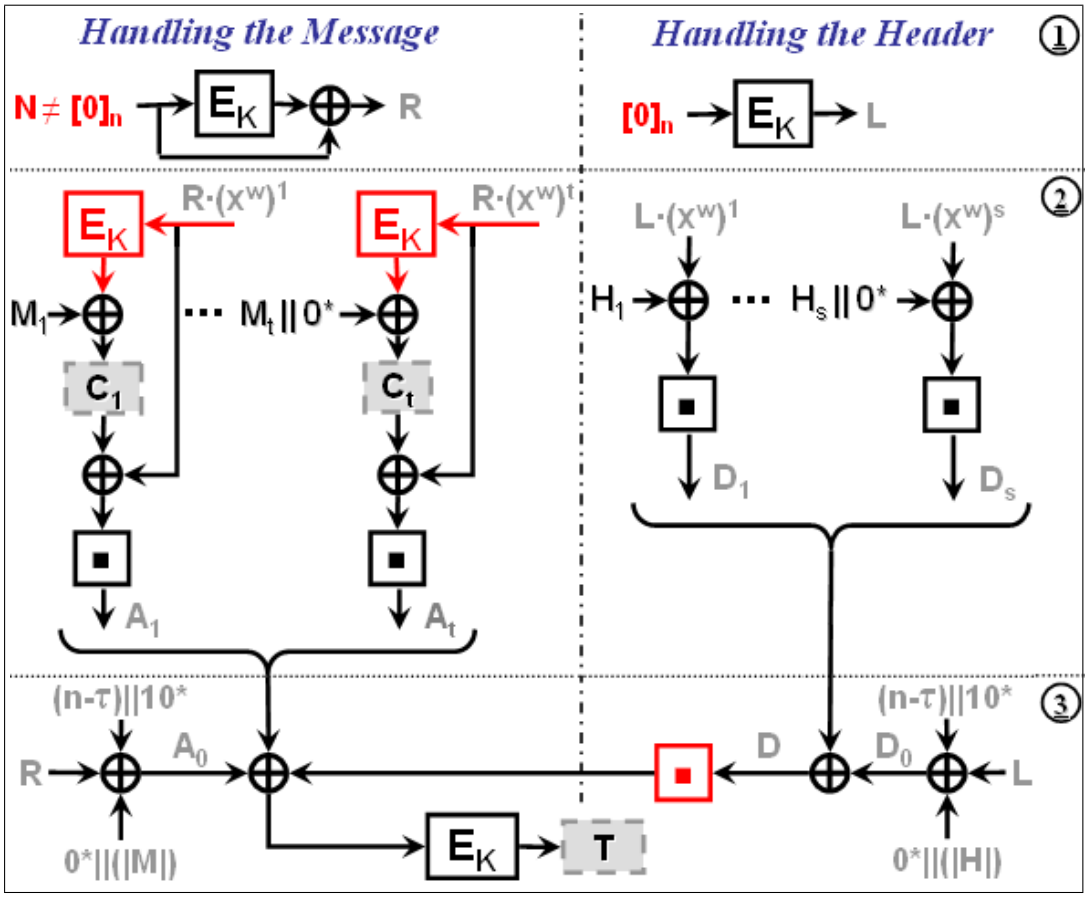

Figure 14: The LetterSoup AEAD scheme. The main differences to Marvin's algorithm are highlighted in red. We can identify the following phases: (1) Initialization, (2) Data Processing, and (3) Tag Computation.

Algorithm 4.5 describes the encryption process using the resulting LFSRC-Marvin AEAD scheme which, for the sake of pronunciation, we call LetterSoup(N, M, H, K, τ) or simply LetterSoup$_K(N, M, H, \tau)$. For convenience, we write $\blacksquare^*(R, C, \tau)$ as the value A accumulated by steps 3 to 8 of Marvin for the (encrypted) message C and tag length τ. Here, LFSRC(N, M, K) denotes the LFSRC mode of operation defined in Algorithm 4.4. The encryption process is also depicted in Figure 14, while Algorithm 4.6 describes the decryption process.

It is important to note that the specific way the nonce is preprocessed in Letter-

Algorithm 4.4 Linear Feedback Shift Register Counter (LFSRC) mode — encryption algorithm.

INPUT: N ▷ nonce, an integer value in range $0 < N < 2^n$.
INPUT: K ▷ cipher key.
INPUT: M ▷ message to encrypt.
OUTPUT: C ▷ ciphertext of M under K.
1: Partition the message as $M = M_1 \| \ldots \| M_t$, where $|M_i| = n$ for $i = 1, \ldots, t-1$, and $0 \leqslant |M_t| \leqslant n$ with $|M_t| = 0$ iff $|M| = 0$.
2: **for** $i \leftarrow 1$ **to** t **do** ▷ incrementally or in parallel
3: $O_i \leftarrow N \cdot (x^w)^i$; $C_i \leftarrow M_i \oplus E_K(O_i)[|M_i|]$
4: **end for**
5: **return** $C \leftarrow C_1 \| \ldots \| C_t$

Soup ensures that the offset seeds R and L passed as parameters to $\blacksquare^*$ and LFSRC are nonzero, as long as the underlying block cipher does not have weak keys (as discussed in section 4.2.1). Moreover, the nonce value $N = [0]_n$ is not allowed by the algorithm, since it is reserved for the computation of L.

Algorithm 4.5 The LetterSoup AEAD mode – Encryption.

INPUT: N ▷ nonce, an integer value in range $0 < N < 2^n$.
INPUT: M ▷ message to encrypt and authenticate, with $|M| < 2^n$.
INPUT: H ▷ associated data, with $|H| < 2^n$.
INPUT: K ▷ cipher key.
INPUT: τ ▷ desired MAC length ($\tau \leqslant n$).
OUTPUT: C ▷ ciphertext of M under K.
OUTPUT: T ▷ MAC of the message, truncated to τ bits.
1: Partition the message as $M = M_1 \| \ldots \| M_t$, where $|M_i| = n$ for $i = 1, \ldots, t-1$, and $0 \leqslant |M_t| \leqslant n$ with $|M_t| = 0$ iff $|M| = 0$
2: $R \leftarrow E_K(\mathbf{lpad}(\mathbf{bin}(N))) \oplus \mathbf{lpad}(\mathbf{bin}(N))$
3: $C \leftarrow \mathrm{LFSRC}(R, M, K)$
4: $A \leftarrow \blacksquare^*(R, C, \tau)$
5: **if** $H \neq \varepsilon$ **then**
6: Partition the associated data as $H = H_1 \| \ldots \| H_s$, where $|H_i| = n$ for $i = 1, \ldots, t-1$, and $0 \leqslant |H_s| \leqslant n$ with $|H_s| = 0$ iff $|H| = 0$
7: $L \leftarrow E_K([0]_n)$; $D \leftarrow \blacksquare^*(L, H, \tau)$; $A \leftarrow (A \oplus \blacksquare(D))$
8: **end if**
9: $T \leftarrow E_K(A)[\tau]$
10: **return** $C \| T$

Algorithm 4.6 The LetterSoup AEAD mode – Decryption.

INPUT: N ▷ nonce, an integer value in range $0 < N < 2^n$.
INPUT: C ▷ message to decrypt and authenticate, with $|C| < 2^n$.
INPUT: H ▷ associated data, with $|H| < 2^n$.
INPUT: K ▷ cipher key.
INPUT: T ▷ tag to be verified, with $|T| = \tau$.
INPUT: τ ▷ desired MAC length ($\tau \leqslant n$).
OUTPUT: M or INVALID ▷ plaintext of C under K if T is a valid tag; INVALID otherwise.

1: Partition the message as $C = C_1 \| \ldots \| C_t$, where $|C_i| = n$ for $i = 1, \ldots, t-1$, and $0 \leqslant |C_t| \leqslant n$ with $|C_t| = 0$ iff $|C| = 0$
2: $R \leftarrow E_K(\mathbf{lpad}(\mathbf{bin}(N))) \oplus \mathbf{lpad}(\mathbf{bin}(N))$
3: $A \leftarrow \blacksquare^*(R, C, \tau)$
4: **if** $H \neq \varepsilon$ **then**
5: Partition the associated data as $H = H_1 \| \ldots \| H_s$, where $|H_i| = n$ for $i = 1, \ldots, t-1$, and $0 \leqslant |H_s| \leqslant n$ with $|H_s| = 0$ iff $|H| = 0$
6: $L \leftarrow E_K([0]_n)$; $D \leftarrow \blacksquare^*(L, H, \tau)$; $A \leftarrow (A \oplus \blacksquare(D))$
7: **end if**
8: $T' \leftarrow E_K(A)[\tau]$
9: **if** $T' \neq T$ **then**
10: **return** INVALID
11: **else**
12: **return** $M \leftarrow \text{LFSRC}(R, C, K)$
13: **end if**

4.3.1 Handling the Associated Data

Note that LetterSoup allows for associated data in a special, non-straightforward manner. Suppose the authentication tag for some confidential message M under key K and nonce N is $A \leftarrow \text{LetterSoup}_K(N, M, \varepsilon, \tau)$, and the authentication tag for some associated (plain) data H is $D \leftarrow \text{LetterSoup}_K([0]_n, H, \varepsilon, \tau)$. The simplest way to combine them would be setting $T = A \oplus D$, as is the case with for the OCB mode (KROVETZ; ROGAWAY, 2005). However, if the same key K is used to authenticate H in a context *distinct* from that where K is used to encrypt and authenticate M (so that actually M and H are semantically unrelated to each other), an attacker can semantically link M and H by intercepting the authenticated encrypted message (N, C, ε, A) corresponding to M, thus forging the authenticated encrypted message with associated data $(N, C, H, A \oplus D)$ without knowing the key.

This undesirable situation could be prevented by using one-time keys, but the whole point of using nonces is to avoid the need to change keys too often. Another possible solution would be similar to the one adopted by EAX, which prepends a different constant t to the CMAC algorithms (CMAC^t) used for authenticating H and M, but this could bring a negative impact to performance. We decided to adopt a more elegant and efficient way to thwart the problem, combining the accumulated tags before the final encryption using the SCT operation on D, the tag computed from the tag H (the same effect could be obtained by applying the SCT on A, but we preferred to conserve a greater similarity with the MARVIN algorithm on the message processing side).

Finally, we point out that LETTERSOUP allows the pre-computation of static associated data. This feature is especially interesting in scenarios where many messages have a same header $\check{H}$: in this case, one can pay the cost of authenticating $\check{H}$ only once, producing a $\check{D} = \blacksquare(\text{LETTERSOUP}_K([0]_n, \check{H}, \varepsilon, \tau))$ that can be re-used in future authentications with no significant cost.

4.4 Additional Features of MARVIN and LETTERSOUP

In addition to the features already described, both MARVIN and LETTERSOUP are online, meaning that they are able to process a stream of data as it arrives, with constant memory, without knowing in advance when the stream will end. The value of this property has been first discussed in (PETRANK; RACKOFF, 2000), and this is an important requirement for applications that do not provide the length of a message in advance: in these scenarios, online algorithms are considerably more efficient than their counterparts, since they do not need to buffer the whole data prior to its processing. For this reason, the lack of this property is usually considered a design flaw for MACs and AEADs (BELLARE; ROGAWAY; WAGNER, 2004). Indeed, this is an important source of criticism of algorithms such as CCM (NIST, 2004), which require the length of the message to be known in advance and thus is not online (ROGAWAY; WAGNER, 2003).

Furthermore, and unlike OCB, LetterSoup allows the tags to be verified prior to their decryption, allowing further processing savings when the message is invalid.

4.5 Summary

In this chapter, we have detailed our MAC and AEAD proposals: Marvin and LetterSoup, respectively. The basis for these algorithms is the Alred structure, due to its reduced cost per authenticated message. Furthermore, they display some interesting features not directly related to this design strategy, which makes them very flexible for implementation in constrained platforms and also on modern computers: both solutions are parallelizable, incremental, and online, while LetterSoup also allows the header pre-computation, length-preserving, and does not require the implementation of E_K^{-1}.

As is *de rigueur* for any cryptographic algorithm proposal, in the next chapter we focus our attention on a detailed security analysis of the Marvin and LetterSoup algorithms, providing the necessary proofs of their security.

5 SECURITY ANALYSIS

In the previous chapter, we specified the MARVIN and LETTERSOUP algorithms and detailed their basic building blocks.

In this chapter, we evaluate the security of the proposed algorithms. We do so by applying the widely accepted Game-Playing technique (BELLARE; ROGAWAY, 2004), first for MARVIN and then for LETTERSOUP. As a result, we show that both solutions are secure against attacks as long as the underlying block cipher adopted is itself a secure cipher.

Before we present our results, though, we give a brief description on the functioning of the Game-Playing technique and present the notation used in the remainder of this chapter.

5.1 Provable Security and the Game-Playing Technique

Provable security is a mathematical technique, based on complexity theory, according to which the goals and requirements of a crypto-system are precisely specified and analyzed. This allows the development of a rigorous security proof for scenarios in which the underlying assumptions hold true.

The Game-Playing technique is a general method to structure and unify cryptographic security proofs (BARTHE; GRÉGOIRE; BÉGUELIN, 2009). In this approach, the

attacker is modeled as the "player" of a sequence of games. In each game, the attacker uses some information (e.g., a number of authentication tags computed with the same key) and interacts with a "challenger" entity (e.g., an oracle that provides authentication tags under request, but never discloses the secret key used to compute them, or other internal information); usually, the attacker is allowed to fix some "coins", i.e., to fix a set of values (e.g., encryption requests and responses) as if they had already appeared in the game prior to its start.

The attacker's goal in each game is to obtain a certain knowledge (e.g., retrieve a MAC function's secret key, tell apart two messages and their respective authentication tags, or simply distinguish a random oracle from a real MAC algorithm) with a high probability and using a limited amount of resources (i.e., the attack must be computationally feasible). Finally, like in a real game, the attacker is obliged to follow some rules that model reality (e.g., a single nonce cannot be reused) during all interactions with the challenger; these rules determine the security assumptions of the proof, and should not be too restrictive or the scope of the security proof itself becomes limited, i.e., it would fail to cover some attacks that can be perpetrated by real adversaries (e.g., if the game does not allow attackers to choose the messages they send to the oracle, the security proof will not address chosen-plaintext attacks).

A detailed discussion on how the Game-Based approach works is beyond the scope of this document, especially because the literature includes some extensive tutorials on this technique — see, for example, (BELLARE; ROGAWAY, 2004; SHOUP, 2004) — and many examples of its application — e.g., (BELLARE; ROGAWAY; WAGNER, 2004; ROGAWAY; BELLARE; BLACK, 2003; ROGAWAY, 2004; BLACK; ROGAWAY, 2002; SARKAR, 2009; ROGAWAY; SHRIMPTON, 2006). Hence, although we try to provide a clear discussion when applying the Game-Based technique, the reading of the aforementioned references is recommended for readers that are unfamiliar with this strategy.

5.2 Definitions and Notation

In the following, we recall some definitions needed for our security proof. We follow the notation presented in (BLACK; ROGAWAY, 2002; ROGAWAY; BELLARE; BLACK, 2003; BELLARE; ROGAWAY; WAGNER, 2004), since these studies were a great source of inspiration for our own analysis.

Block Ciphers and PRFs. As previously stated, a block cipher is a function $E : \mathcal{K} \times \{0,1\}^n \rightarrow \{0,1\}^n$ where $\mathcal{K} : \{0,1\}^k$ is a finite set (the *set of keys*) and each $E_K(\cdot) = E(K,\cdot)$ is a permutation on $\{0,1\}^n$. Let $Perm(n)$ denote the set of all permutations on $\{0,1\}^n$. This set can be regarded as a block cipher by imagining that each permutation is defined by a unique element of $\mathcal{K}$.

Let $\mathcal{A}$ be an adversary (a probabilistic algorithm) with access to an oracle, and suppose that $\mathcal{A}$ always outputs a bit. We write $\mathcal{A}^{e(\cdot)}$ to indicate that $\mathcal{A}$ uses oracle $e(\cdot)$. Define:

$$\mathbf{Adv}_E^{prp}(\mathcal{A}) = Pr[K \xleftarrow{\$} \mathcal{K} : \mathcal{A}^{E_K(\cdot)} = 1] - Pr[\pi \xleftarrow{\$} Perm(n) : \mathcal{A}^{\pi(\cdot)} = 1]$$

This expression gives the probability that adversary $\mathcal{A}$ outputs 1 when given an oracle for $E_K(\cdot)$, minus the probability that A outputs 1 when given an oracle for $\pi(\cdot)$, where K is selected at random from $\mathcal{K}$ and π is selected at random from $Perm(n)$ (this random selection is denoted by the symbol $\xleftarrow{\$}$). Hence, it defines the advantage of $\mathcal{A}$ when trying to distinguish E from a random permutation.

Analogously, a function family from n-bits to n-bits is a map $F : \mathcal{K} \times \{0,1\}^n \rightarrow \{0,1\}^n$, where $\mathcal{K} : \{0,1\}^k$ is a finite set. We write $F_K(\cdot)$ for denoting $F(K,\cdot)$. Let $Rand(n)$ denote the set of all functions from $\{0,1\}^n$ to $\{0,1\}^n$. Again, this set can be regarded as a function family. We can then define the advantage of adversary $\mathcal{A}$ when

trying to distinguish F_K from a random function as:

$$\mathbf{Adv}_F^{prf}(\mathcal{A}) = Pr[K \xleftarrow{\$} \mathcal{K} : \mathcal{A}^{F_K(\cdot)} = 1] - Pr[\rho \xleftarrow{\$} Rand(n) : \mathcal{A}^{\rho(\cdot)} = 1]$$

A function family from $\{0,1\}^*$ to $\{0,1\}^\tau$ is a map $f : \mathcal{K} \times \{0,1\}^* \to \{0,1\}^\tau$, where $\mathcal{K} : \{0,1\}^k$ is a set with an associated distribution. We write $f_K(\cdot)$ for $f(K,\cdot)$. Let $Rand(*,\tau)$ denote the set of all functions from $\{0,1\}^*$ to $\{0,1\}^\tau$. This set has an associated probability measure, since we assert that a random element ρ of $Rand(*,\tau)$ associates to each string $x \in \{0,1\}^*$ a random string $\rho(x) \in \{0,1\}^\tau$. Define:

$$\mathbf{Adv}_f^{prf}(\mathcal{A}) = Pr[K \xleftarrow{\$} \mathcal{K} : \mathcal{A}^{f_K(\cdot)} = 1] - Pr[g \xleftarrow{\$} Rand(*,\tau) : \mathcal{A}^{g(\cdot)} = 1]$$

AEAD Schemes. A (nonce-based) *Authenticated-Encryption with Associated Data* (AEAD) scheme is a pair of algorithms $\Pi = (\mathcal{E}_K, \mathcal{E}_K^{-1})$ and an associated number n (the nonce length). Here, $\mathcal{E}_K$ is a deterministic encryption algorithm that takes a nonce $N \in \{0,1\}^n$, a key $K \in \mathcal{K}$, a header $H \in \{0,1\}^*$, and a message $M \in \{0,1\}^*$, and returns the string $CT \leftarrow \mathcal{E}_K(N,H,M)$, which corresponds to the ciphertext and the authentication tag; analogously, $\mathcal{E}_K^{-1}$ denotes a deterministic decryption algorithm taking as input $K \in \mathcal{K}$, $N \in \{0,1\}^n$, $H \in \{0,1\}^*$ and $CT \in \{0,1\}^*$, and returns $\mathcal{E}_K^{-1}(N,H,M)$, which is either a string $M \in \{0,1\}^n$ or the distinguished symbol Invalid. We require that $\mathcal{E}_K^{-1}(N,H,CT) = M$ whenever $CT \leftarrow \mathcal{E}_K(N,H,M)$, and $\mathcal{E}_K^{-1}(N,H,CT) = \text{Invalid}$ otherwise. For notational simplicity, we assume that $|\mathcal{E}_K(N,H,M)| = \ell(|M|)$, i.e., that the size of the encryption algorithm's output depends only on the size of M.

Nonce-Respecting. Suppose $\mathcal{A}$ is an adversary with access to an *encryption oracle* $\mathcal{E}_K(\cdot,\cdot,\cdot)$. This oracle, on input (N,H,M), returns $\mathcal{E}_K(N,H,M)$. Let the tuples $(N_1,H_1,M_1),\ldots,(N_q,H_q,M_q)$ denote the oracle queries. $\mathcal{A}$ is said to be *nonce-respecting* if N is never repeated, i.e., if $N_1,\ldots,N_q$ are always distinct, regardless of oracle responses and of $\mathcal{A}$'s internal coins (if any). As usually done in the security analysis of AEAD schemes, we assume all adversaries to be nonce-respecting.

Privacy of AEAD Schemes. Consider a nonce-respecting adversary $\mathcal{A}$ with access to one of two possible oracles: a "real" and a "fake" one. A "real" encryption oracle $\mathcal{E}_K(\cdot,\cdot,\cdot)$ takes as input (N, H, M) and returns $CT \leftarrow \mathcal{E}_K(N, H, M)$. A "fake" encryption oracle $\$(\cdot,\cdot,\cdot)$ takes as input (N, H, M) and returns a random string $CT \xleftarrow{\$} \{0,1\}^{\ell(|M|)}$. The advantage of $\mathcal{A}$ in violating the privacy of an AEAD scheme $\Pi = (\mathcal{E}_K, \mathcal{E}_K^{-1})$ is defined as:

$$\mathbf{Adv}_{\Pi}^{priv}(\mathcal{A}) = Pr[K \xleftarrow{\$} \mathcal{K} : \mathcal{A}^{\mathcal{E}_K(\cdot,\cdot,\cdot)} = 1] - Pr[K \xleftarrow{\$} \mathcal{K} : \mathcal{A}^{\$(\cdot,\cdot,\cdot)} = 1]$$

Authenticity of AEAD Schemes. Suppose that we fix an AEAD scheme $\Pi = (\mathcal{E}_K, \mathcal{E}_K^{-1})$ and run a nonce-respecting adversary $\mathcal{A}$ with an encryption oracle $\mathcal{E}_K(\cdot,\cdot,\cdot)$ for some key K. We say that $\mathcal{A}$ *forges* (in this run) if $\mathcal{A}$ outputs (N, H, CT) such as $\mathcal{E}_K^{-1}(N, H, CT) \neq$ INVALID, and $\mathcal{A}$ made no earlier query (N, H, M) that resulted in CT as response. We stress that, in this forgery attempt, $\mathcal{A}$ is allowed to choose a nonce already used in one of the previous queries to oracle $\mathcal{E}_K(\cdot,\cdot,\cdot)$. We define the advantage of $\mathcal{A}$ in violating the authenticity of Π as:

$$\mathbf{Adv}_{\Pi}^{auth}(\mathcal{A}) = Pr[K \xleftarrow{\$} \mathcal{K} : \mathcal{A}^{\mathcal{E}_K(\cdot,\cdot,\cdot)} forges]$$

5.3 Accumulation Collisions

The notion of *accumulation collisions* generalizes internal collisions — as defined in (DAEMEN; RIJMEN, 2005a, section 2.2) — for parallelizable MAC (and AEAD) solutions, and is essential for the security analysis of the ALRED construction. In this section, we focus our discussion on MAC algorithms rather than on AEAD schemes, but the concepts described here can be explored in the security analysis of both algorithms.

Two messages $M = M_1 \| \ldots \| M_t$ and $M' = M'_1 \| \ldots \| M'_{t'}$ are said to cause an accumulation collision if there exists a subset of indexes $Z \subseteq \{1, \ldots, \min(t, t')\}$ such that

$M_i = M'_i$ for all $i \in Z$ and $\bigoplus_{i \notin Z} A_i = \bigoplus_{j \notin Z} A'_j$, where $A_i = \blacksquare(\mathbf{rpad}(M_i) \oplus R \cdot (x^w)^i)$ and $A'_j = \blacksquare(\mathbf{rpad}(M'_j) \oplus R \cdot (x^w)^j)$. Clearly such messages have the same MAC value and, hence, given the tag of either of them, one can forge the tag of the other. This phenomenon is unavoidable regardless of the details of the (parallelizable) MAC function: if the same key is used to authenticate a large enough number of messages (namely, $O(2^{n/2})$ messages for an n-bit underlying block cipher), the birthday paradox (STINSON, 2002) makes the condition $\bigoplus_{i \notin Z} A_i = \bigoplus_{j \notin Z} A'_j$ likely to occur, opening the way to forgery. Therefore, the number of messages authenticated under any particular key must be much smaller than $2^{\lfloor n/2 \rfloor}$. As long as the effective upper bound is large enough for the target application (e.g., if it allows more messages than the network's lifespan), this can be considered a minor limitation.

Finding the collision on the A_i and A'_j internal states is equivalent to solving the balance problem (BELLARE; MICCIANCIO, 1997) in the additive group of a binary field *without* knowing the group elements, since these states depend on a secret key (and this secrecy is assumed along all this chapter).

Given some message M, we distinguish two approaches for creating a second message M' that can lead to an accumulation collision. The first is to create M' such that $|M'| \neq |M|$, which we call a *Fixed Point Collision*[1] . The second is to modify some blocks of M, keeping its size unchanged, which we call a *Extinguishing Differential Collision*[1]. We discuss the efficacy of these methods in the following, thus building the basis for the our algorithms' security analysis, presented in sections 5.4 and 5.5.

5.3.1 Fixed Point Collision

In this case, the $\mathcal{A}$ could concatenate blocks to M, and possibly modify some of its blocks at the same time. Note that this approach is equivalent to removing some blocks from M (and possibly modifying the remainder data), since we can think of the

[1] Here we loosely follow ALRED's naming convention introduced in (DAEMEN; RIJMEN, 2005a).

shorter message as being the one to which the attacker added blocks.

The choice of the additional blocks must be made in such a manner that the final value of the accumulator is not modified. For example, if $\mathcal{A}$ simply concatenates a blocks to M, yielding message $M' = M_1 \| \ldots \| M_t \| M_{t+1} \| \ldots \| M_{t+a}$, a collision will occur only if $\bigoplus_{t<i\leqslant t+a} A_i = [0]_n$.

As we will show in our security analysis, constructing messages that satisfy these properties is a difficult task due to the secrecy of the offset seeds used in our constructions.

5.3.2 Extinguishing Differential Collision

The second strategy is to keep the size of $M = M_1 \| \ldots \| M_t$ unchanged, but modify some of its internal blocks. This means that the attacker creates a message $M' = M'_1 \| \ldots \| M'_t$ that is identical to M at indexes $Z \subseteq \{1, \ldots, t\}$, and is different everywhere else. In this case, the attacker's goal is to find an M' such that $\bigoplus_{i \notin Z} A_i = \bigoplus_{j \notin Z} A'_j$.

Note that the modification of a single block would not work, since the SCT is an invertible function and, thus, $\blacksquare(\mathbf{rpad}(M_i) \oplus R \cdot (x^w)^i) \neq \blacksquare(\mathbf{rpad}(M'_i) \oplus R \cdot (x^w)^i)$ for $M_i \neq M'_i$. Therefore, in its simplest form, the attack would consist in modifying at least two indexes, say, i and j. This strategy can be seen as a type of Differential Cryptanalysis (HEYS, 2002): the attacker injects a difference ΔM at the input of some structure (in this case, an SCT) expecting to create some difference ΔA at the structure's output. As long as both (M_i, M_j) and (M'_i, M'_j) (or, more specifically, the differences $\Delta M = (\mathbf{rpad}(M_i) \oplus R \cdot (x^w)^i \oplus \mathbf{rpad}(M_j) \oplus R \cdot (x^w)^j)$ and $\Delta M' = (\mathbf{rpad}(M'_i) \oplus R \cdot (x^w)^i \oplus \mathbf{rpad}(M'_j) \oplus R \cdot (x^w)^j)$) lead to the same difference ΔA with a high enough probability, an accumulation collision can be mounted in practice. The pair $(\Delta M, \Delta A)$ is called a *differential*.

In our MAC and AEAD constructions, such probabilities are low due to the combination of two factors. The first is the secrecy of the offset seeds: since the exact value of the difference at the entry point of SCTs depend on this secret value, attackers cannot freely manipulate ΔM and $\Delta M'$. This property is likely to thwart the construction of well-known high-probability differentials (e.g., those starting with a few non-zero differences in the input block) and, in practice, raises the number of S-Boxes in all differentials that an attacker can expect to construct.

The second factor refers to the structure of the SCTs. As discussed in section 4.1.1, the use of SCTs in the ALRED construction aims specifically to prevent differential trails with high probability, which is achieved by assuring a high number of active S-Boxes in every possible trail. If the underlying b-bundle block cipher adopts an S-Box S whose δ-parameter[2] is δ_S, the probability $1/p_t$ of any differential trail t constructed over an SCT is such that $1/p_t \leqslant (\delta_S)^b$ since, by definition, each SCT has at least b active S-Boxes. Nonetheless, some trails that lead to the same differential are likely to exist and, hence, the probability of most differentials is likely higher than the probability of a single trail (KELIHER, 2004).

Let $1/p$ denote the maximum probability of any differential $(\Delta M, \Delta A)$ over an SCT. Ideally, we would have $p = 2^n$, ensuring that the probability of a collision when authenticating $m < 2^{n/2}$ messages would be upper bounded by $1 - exp(-m^2/2^{n+1})$ due to the birthday paradox. However, in practice this probability is higher. For example, as discussed in Appendix B, we have $1/p \leqslant 1.881 \times 2^{-114}$ for the AES block cipher, while the CURUPIRA displays $1/p \leqslant 1.4926 \times 2^{-71}$.

This impacts on the maximum number of messages that can be authenticated using our algorithms, as further discussed in our security analysis.

[2] The δ-parameter of an S-Box gives the highest probability of differentials in that S-Box. See section A.3 for a formal definition.

5.4 On the Security of Marvin

As in many security proofs, we start with an information-theoretic approach. For this, we use an abstraction of MARVIN denoted by MARV[$Perm(n), \tau$], meaning that the underlying block cipher is replaced by a pseudo-random permutation.

First, we analyze the pseudo-randomness of this abstraction. More specifically, we show that adversaries who are unable to create accumulation collisions are also unable to distinguish MARVIN from a random function $\rho : \{0,1\}^* \rightarrow \{0,1\}^\tau$. The usefulness of this analysis relies on the fact that being secure in the sense of a Pseudo-Random Function (PRF) implies an inability to forge with good probability (i.e., higher than 50%) (GOLDREICH; GOLDWASSER; MICALI, 1986; BELLARE; KILIAN; ROGAWAY, 2000).

We then analyze the effect of accumulation collisions, considering attacks based either on fixed points or extinguishing differentials, as described in section 5.3.

Finally, we pass to a complexity-theoretic analysis: we show that the adversary's advantage is increased by a small amount when we replace $Perm(n)$ by a block cipher E in our scheme, unless some adversary is able to distinguish E from a random permutation with a high enough probability.

5.4.1 Basic Games

Let $\mathcal{A}$ be an adversary that attacks MARV[$Perm(n), \tau$], trying to distinguish it from a random function. We consider that $\mathcal{A}$ is computationally unbounded (i.e., we do not limit the amount of resources available to $\mathcal{A}$) and, thus, we can assume without loss of generality that $\mathcal{A}$ is deterministic. We can model the interaction between $\mathcal{A}$ and a MARV[$Perm(n), \tau$] oracle as the Game M1, described in Algorithm 5.1, in which the attacker performs q queries.

In Game M1, we can observe the occurrence of "failure events", i.e., situations

Algorithm 5.1 – Game M1, which accurately simulates MARV[$Perm(n), \tau$].

Initialization:

1: $R \xleftarrow{\$} \{0,1\}^n$; $\pi(\mathbf{lpad}(c)) \leftarrow \{R\}$; $bad \leftarrow \texttt{false}$

When $\mathcal{A}$ makes its r-th query ($M^r = M^r_1 \| \ldots \| M^r_{t^r}$)**, where** $r \in [1 \ldots q]$**, do:**

2: **for** $i \leftarrow 1$ **to** t^r **do**
3: $X^r_i \leftarrow \mathbf{rpad}(M^r_i) \oplus R \cdot (x^w)^i$; $A^r_i \leftarrow \blacksquare(X^r_i)$
4: **end for**
5: $A^r_0 \leftarrow R \oplus \mathbf{rpad}(\mathbf{bin}(n-\tau) \| 1) \oplus \mathbf{lpad}(\mathbf{bin}(|M^r|))$; $\Sigma^r \leftarrow \bigoplus_{i=0}^{t^r} A^r_i$
6: **if** $\Sigma^r \in Domain(\pi)$ **then**
7: $bad \leftarrow \texttt{true}$; $TAG^r \leftarrow \pi(\Sigma^r)$ ▷ Accumulation Collision
8: **else**
9: $TAG^r \xleftarrow{\$} \{0,1\}^n$
10: **if** $TAG^r \in Range(\pi)$ **then**
11: $bad \leftarrow \texttt{true}$; $TAG^r \xleftarrow{\$} \overline{Range(\pi)}$
12: **end if**
13: **end if**
14: $\pi(\Sigma^r) \leftarrow TAG^r$
15: **return** $T^r \leftarrow TAG^r[\tau]$

where a result is not exactly what it should be. This is the case of line 11, in which we notice that the value of TAG^r chosen at random corresponds to some value already defined in $Range(\pi)$, conflicting with the definition of π as a permutation. A similar event occurs in line 7, where the occurrence of an accumulation collision limits the number of choices for the value of TAG^r. In these situations, the game sets a special flag *bad* to `true`, indicating that some problem occurred, but does not tell anything to the adversary. We observe that, if this flag is not set to `true` during an execution of Game M1, then the value of T^r returned by the Game at line 15 is completely random, since it corresponds to the first τ bits of the string selected randomly at line 9. It follows that $\mathbf{Adv}^{prf}_{\text{MARV}[Perm(n),\tau]}(\mathcal{A})$ is at most the probability that *bad* is set to `true` in Game M1. Then, in order to evaluate this advantage, we need to bound this probability.

We start analyzing the probability that *bad* gets set to `true` in line 11. As discussed above, this happens when a n-bit string chosen at random is already part of the set $Range(\pi)$. The size of this set starts at 1, after the execution of line 1, and then grows

by one element at each query. Therefore, we have:

$$\begin{aligned} Pr_{M1}[bad \text{ gets set at line 11}] &\leqslant (1 + 2 + \cdots + q)/2^n \\ &\leqslant (q+1)^2/2^{n+1} \end{aligned} \tag{5.1}$$

We use the subscript M1 in the probability of Equation 5.1 as a reminder that we are considering the behavior of Game M1.

Now that we have bound the probability for line 11, we modify Game M1 by changing its behavior when (and only when) *bad* is set to `true`. We do so simply by omitting line 11, as well as the second statement of line 7 (highlighted) and the subsequent **else**. The resulting Game, M2, is described in Algorithm 5.2.

Algorithm 5.2 – Game M2: A simplification of Game **M1**, used to bound $\mathbf{Adv}^{prf}_{\text{MARV}[Perm(n),\tau]}(\mathcal{A})$ as given in Equation 5.2.

Initialization:

1: $R \xleftarrow{\$} \{0,1\}^n \quad ; \quad \pi([c]_n) \leftarrow \{R\} \quad ; \quad bad \leftarrow \texttt{false}$

When $\mathcal{A}$ makes its r-th query $(M^r = M^r_1 \| \ldots \| M^r_{t^r})$**, where** $r \in [1 \ldots q]$**, do:**

2: **for** $i \leftarrow 1$ **to** t^r **do**

3: $\quad X^r_i \leftarrow \mathbf{rpad}(M^r_i) \oplus R \cdot (x^w)^i \quad ; \quad A^r_i \leftarrow \blacksquare(X^r_i)$

4: **end for**

5: $A^r_0 \leftarrow R \oplus \mathbf{rpad}(\mathbf{bin}(n-\tau) \| 1) \oplus \mathbf{lpad}(\mathbf{bin}(|M^r|)) \quad ; \quad \Sigma^r \leftarrow \bigoplus_{i=0}^{t^r} A^r_i$

6: **if** $\Sigma^r \in Domain(\pi)$ **then** $bad \leftarrow \texttt{true}$ ▷ Accumulation Collision

7: $TAG^r \xleftarrow{\$} \{0,1\}^n \quad ; \quad \pi(\Sigma^r) \leftarrow TAG^r$

8: **return** $T^r \leftarrow TAG^r[\tau]$

With this modified Game, we can bound the adversary's advantage simply by using the correction factor given by Equation 5.1, as follows:

$$\mathbf{Adv}^{prf}_{\text{MARV}[Perm(n),\tau]}(\mathcal{A}) \leqslant Pr_{M2}[bad \text{ gets set to } \texttt{true}] + (q+1)^2/2^{n+1} \tag{5.2}$$

Note that we use the subscript M2 in equation 5.2 as a reminder that we are considering the probability in Game M2. Note also that, in this game, the value of T^r returned in response to a query is always a random τ-bit string. Nonetheless, Game M2 does more than just return these random strings: it also randomly chooses the value of R

(line 1), fills in the values of π (line 7) and sets *bad* to `true` under certain conditions (namely, if an accumulation collision occurs at line 6). We can, however, defer doing all those things and simply return random strings $T^1, \ldots, T^r, \ldots, T^q$ to the adversary $\mathcal{A}$ at each query. This "trick", adopted in Game M3 (see algorithm 5.3), is not perceptible for an adversary $\mathcal{A}$ interacting with this modified game, since $\mathcal{A}$ would still receive the same results expected from Game M2. Moreover, this modification does not change the probability that *bad* is set to `true`. Hence, we can use Game M3 in order to bound $\mathbf{Adv}^{prf}_{\textsc{Marv}[Perm(n),\tau]}(\mathcal{A})$ using Equation 5.2.

Algorithm 5.3 – Game M3: Equivalent to Game M2 from the adversary's point of view. It simply returns random strings to the attacker, not changing the probability that *bad* is set to `true`.

When $\mathcal{A}$ makes its r-th query ($M^r = M^r_1 \| \ldots \| M^r_{t^r}$)**, where** $r \in [1 \ldots q]$**, do:**

1: $TAG^r \xleftarrow{\$} \{0,1\}^n$
2: **return** $T^r \leftarrow TAG^r[\tau]$

After $\mathcal{A}$ finishes making its q queries, do:

3: $R \xleftarrow{\$} \{0,1\}^n$; $\pi([c]_n) \leftarrow \{R\}$; $bad \leftarrow$ `false`
4: **for** $r \leftarrow 1$ **to** q **do**
5: **for** $i \leftarrow 1$ **to** t^r **do**
6: $X^r_i \leftarrow \mathbf{rpad}(M^r_i) \oplus R \cdot (x^w)^i$; $A^r_i \leftarrow \blacksquare(X^r_i)$
7: **end for**
8: $A^r_0 \leftarrow R \oplus \mathbf{rpad}(\mathbf{bin}(n-\tau) \| 1) \oplus \mathbf{lpad}(\mathbf{bin}(|M^r|))$; $\Sigma^r \leftarrow \bigoplus_{i=0}^{t^r} A^r_i$
9: **if** $\Sigma^r \in Domain(\pi)$ **then** $bad \leftarrow$ `true` ▷ Accumulation Collision
10: $\pi(\Sigma^r) \leftarrow TAG^r$
11: **end for**

The probability that *bad* is set to `true` in Game M3 is over the random values of TAG^r selected at line 1 and the random value of R selected at line 3, since those are the only points where the set $Domain(\pi)$ is modified. We want to show that, over these random values, *bad* will rarely be set to `true`. In fact, we can show something stronger: even if we let $\mathcal{A}$ fix the values of $TAG^r \in \{0,1\}^n$ for every r ($1 \leqslant r \leqslant q$) and the probability is taken just over the remaining values, the probability that *bad* gets set to `true` is still small. We do so by writing Game M4[φ], described in Algorithm 5.4, in which we eliminate the interaction with adversary $\mathcal{A}$: the oracle responses

are fixed and, since adversary $\mathcal{A}$ itself is deterministic, the queries $M^1, \ldots, M^q$ made by $\mathcal{A}$ are fixed as well. In this scenario, the adversary can be imagined as an entity who knows all of the queries (M^r) that would be asked, and all the corresponding answers (TAG^r) that would be received. Therefore, Game M4[φ] depends only on constants $\varphi = (q, TAG^1, \ldots, TAG^q, M^1, \ldots, M^q)$, which are limited by the amount of resources available to $\mathcal{A}$.

Algorithm 5.4 – Game M4[φ]: This Game depends on constants φ, which specify the values: q, $TAG^1, \ldots, TAG^q \in \{0,1\}^n$ and $M^1, \ldots, M^q$, eliminating the interaction with attackers.

1: $R \xleftarrow{\$} \{0,1\}^n$; $\pi([c]_n) \leftarrow \{R\}$; $bad \leftarrow$ `false`
2: **for** $r \leftarrow 1$ **to** q **do**
3: **for** $i \leftarrow 1$ **to** t^r **do**
4: $X_i^r \leftarrow \mathbf{rpad}(M_i^r) \oplus R \cdot (x^w)^i$; $A_i^r \leftarrow \blacksquare(X_i^r)$
5: **end for**
6: $A_0^r \leftarrow R \oplus \mathbf{rpad}(\mathbf{bin}(n-\tau) \,\|\, 1) \oplus \mathbf{lpad}(\mathbf{bin}(|M^r|))$; $\Sigma^r \leftarrow \bigoplus_{i=0}^{t^r} A_i^r$
7: **if** $\Sigma^r \in Domain(\pi)$ **then** $bad \leftarrow$ `true` ▷ Accumulation Collision
8: $\pi(\Sigma^r) \leftarrow TAG^r$
9: **end for**

Using this new game, we can write:

$$\mathbf{Adv}^{prf}_{\text{MARV}[Perm(n),\tau]}(\mathcal{A}) \leqslant \max_{\varphi}\{Pr_{M4[\varphi]}[bad \text{ gets set to } \texttt{true}]\} + (q+1)^2/2^{n+1} \qquad (5.3)$$

The analysis of Game M4 show that *bad* is set to `true` in line 7 only if: (1) the value of Σ^r computed at line 6 is equal to $[c]_n = [\texttt{0x2A}]_n$; or (2) an accumulation collision occurs with some previous (adaptively chosen) message.

In the following, we will bound the probabilities of these events. First, we show that the probability of having $\Sigma^r = [\texttt{0x2A}]_n$ is low; then, we turn our attention to the two techniques for creating an accumulation collision given some message (or a set of messages): adding or modifying blocks.

5.4.2 Collision with `0x2A`

In this case, we have to calculate the probability $Pr[\Sigma^r = [\texttt{0x2A}]_n]$ or, more generally, $Pr[\Sigma^r = [c]_n]$ for some constant c. This probability can be rewritten as $Pr[R = [c]_n \oplus \mathbf{rpad}(\mathbf{bin}(n-\tau) \,\|\, 1) \oplus \mathbf{lpad}(\mathbf{bin}(|M^r|)) \oplus A_1^r \oplus \ldots A_{t^r}^r] = Pr[R = \alpha \oplus A_1^r \oplus \ldots A_{t^r}^r]$ for some α that does not depend on R and for $A_i^r = \blacksquare(\mathbf{rpad}(M_i^r) \oplus R \cdot (x^w)^i)$.

Due to the randomness of R in Game M4[φ], we would have $Pr[\Sigma^r = [c]_n] = 1/2^n$ if all A_i^r were also independent of R. Although this is not the case, we claim that this probability is not higher than $1/2^n$. The reasoning behind this affirmation is that even if we had a much simpler algorithm, in which the SCTs where replaced by the identity function (i.e., in which $A_i^r = \mathbf{rpad}(M_i^r) \oplus R \cdot (x^w)^i$), this probability would still be $1/2^n$ because, in this scenario, we would have: $Pr[R = \alpha \oplus A_1^r \oplus \ldots \oplus A_{t^r}^r] = Pr[R = \alpha \oplus M_1^r \oplus R \cdot (x^w) \ldots \oplus \mathbf{rpad}(M_{t^r}^r) \oplus R \cdot (x^w)^{t^r}] = Pr[R = (\alpha \oplus M_1^r \oplus \ldots \oplus \mathbf{rpad}(M_{t^r}^r))/(1 \oplus (x^w) \oplus \ldots \oplus (x^w)^{t^r})] = 1/2^n$. Since SCTs are invertible, they follow a uniform probability distribution just like the identity function and, thus, they can be interchanged as above without impacting on the value of $Pr[\Sigma^r = [c]_n]$.

We can thus write, considering all q queries:

$$Pr_{M4[\varphi]}[\text{collision with } \texttt{0x2A}] \;\leqslant\; q/2^n \tag{5.4}$$

5.4.3 Accumulation Collisions: Fixed Points

First, let us take a simple scenario: consider that $\mathcal{A}$ adds a single block to message M^1 when the second query is performed. In this case, the added block $M_{t^1+1}^2$ shall be chosen in such a manner that $A_{t^1+1}^2 = \blacksquare(\mathbf{rpad}(M_{t^1+1}^2) \oplus R \cdot (x^w)^{t^1+1}) = [0]_n$. We need to show that finding such a block is difficult.

In fact, we can show something stronger: that the probability of finding a block M_i such that $A_i = [\alpha]_n$ is at most $1/2^n$ for any fixed value α. This is shown in equation 5.5,

which uses the fact that $(x^w)^i$ is non-zero (and, thus, invertible) and that R is randomly chosen in Game M4.

$$\begin{aligned} Pr[A_i = [\alpha]_n] &= Pr[\blacksquare(\mathbf{rpad}(M_i) \oplus R \cdot (x^w)^i) = [\alpha]_n] \\ &= Pr[\mathbf{rpad}(M_i) \oplus R \cdot (x^w)^i = \blacksquare^{-1}([\alpha]_n)] \\ &= Pr[R \cdot (x^w)^i = \blacksquare^{-1}([\alpha]_n) \oplus \mathbf{rpad}(M_i)] \\ &= Pr[R = (\blacksquare^{-1}([\alpha]_n) \oplus \mathbf{rpad}(M_i)) \cdot (x^w)^{-i}] \\ &= 1/2^n \quad \triangleright \text{ since } R \text{ is random} \end{aligned} \tag{5.5}$$

Equation 5.5 not only upper bounds the probability of our simple scenario, but actually goes beyond it. The addition of more blocks in the adversary's second query also leads to the same probability: by adding a blocks $\{M^2_{t^1+1}, \ldots, M^2_{t^1+a}\}$ to M^1, $\mathcal{A}$ would still need to find a block $M^2_{t^1+i}$ such that $A^2_{t^1+i} = [\alpha]_n$, with the only difference that in this case we would have $[\alpha]_n = \bigoplus_{1 \leqslant j \leqslant a, j \neq i} A^2_{t^1+j}$. And more: if $\mathcal{A}$ decides to create M_2 by adding $a > 0$ blocks to M_1 and by modifying m of its blocks, there would still be the need of finding a block $M^2_{t^1+i}$ such that $A^2_{t^1+i} = [\alpha]_n = (\bigoplus_{1 \leqslant j \leqslant a, j \neq i} A^2_{t^1+j}) \oplus \Delta_m$, where Δ_m is the difference introduced into the accumulator by the m modified blocks. Hence, without knowing R, $\mathcal{A}$ would not be able to create a collision with a higher probability even if the value of $[\alpha]_n$ could be freely chosen.

However, in the r^{th} query, the probability of succeeding in such an attack would increase, since the added blocks could lead to a collision with any of the previously $r - 1$ queried messages. Hence, for a total o q queries, the probability of creating an accumulation collision between any pair of messages that differ in size is given by:

$$\begin{aligned} Pr_{M4[\varphi]}[\text{fixed point}] &\leqslant (0 + 1 + 2 + \cdots + q - 1)/2^n \\ &\leqslant q(q-1)/2^{n+1} \end{aligned} \tag{5.6}$$

5.4.4 Accumulation Collisions: Extinguishing Differentials

Let us start considering the attack where only two blocks M_i^1 and M_j^1 from message M^1 are modified, yielding the message M^2 that is used in the adversary's second query. In this case, $\mathcal{A}$ aims to find a pair of blocks (M_i^2, M_j^2) such that $A_i^2 \oplus A_j^2 = A_i^1 \oplus A_j^1$.

Let $1/p$ denote the maximum probability that a difference ΔM at the input of the SCT will lead to a difference ΔA at its output, whichever the values of ΔM and ΔA. By making $\Delta M = M_i^2 \oplus M_j^2$ and $\Delta A = A_i^1 \oplus A_j^1$, it is easy to see that $1/p$ upper bounds $\mathcal{A}$'s probability of success in any query where only two blocks are modified.

Now suppose that a set $\overline{Z}$ of $m > 2$ indexes are modified in the second query performed by $\mathcal{A}$. In this case, $\mathcal{A}$ has to find an input difference $\bigoplus_{i \in \overline{Z}} M_i^2$ such that $\bigoplus_{i \in \overline{Z}} A_i^1 = \bigoplus_{j \in \overline{Z}} A_j^2$. This is equivalent to find two blocks at indexes α and β such that the input difference $\Delta M = M_\alpha^2 \oplus M_\beta^2$ leads to the output difference $\Delta A = (\bigoplus_{i \in \overline{Z}} A_i^1) \oplus \bigoplus_{j \in \overline{Z} \setminus \{\alpha, \beta\}} A_j^2)$. Therefore, $1/p$ upper bounds the probability of success when $m \geqslant 2$ blocks are modified in the attacker's second query.

In consequence, considering the effect of q queries, the probability of having an accumulation collision by modifying blocks is given by:

$$\begin{aligned} Pr_{M4[\varphi]}[\text{extinguishing differential}] &\leqslant (0 + 1 + 2 + \cdots + q - 1)/p \\ &\leqslant q(q-1)/2p \end{aligned} \tag{5.7}$$

5.4.5 Conclusions

By combining the probabilities given in Equations 5.3, 5.4, 5.6 and 5.7, and from the previous discussion, we can finally state the following theorem:

Theorem 2. ***[Security of* MARVIN*]*** *Fix n, $\tau \geqslant 1$. Let $\mathcal{A}$ be an adversary with access to an oracle, and let $1/p$ be the maximum probability of any differential constructed over*

an SCT. Consider that $\mathcal{A}$ performs q queries to its oracle. Then:

$$\begin{aligned}\mathbf{Adv}^{prf}_{\textsc{Marv}[Perm(n),\tau]}(\mathcal{A}) &\leqslant \max\{q(q-1)/2^{n+1}, q(q-1)/2p\} + q/2^n + (q+1)^2/2^{n+1} \\ &\leqslant q(q-1)/2p + (q^2+4q+1)/2^{n+1} \\ &\leqslant q(q-1)/2p + (q+2)^2/2^{n+1}\end{aligned}$$

Note that, in this theorem, we do not need consider the contributions from both Equations 5.6 and 5.7 but only the maximum between them, which corresponds to the value given in the latter. This happens because, in any query i, a collision with the message from a previous query $j < i$ can occur due to a fixed point (if $|M_i| \neq |M_j|$) or to an extinguishing differential (if $|M_i| = |M_j|$), but not due to both; hence, we consider that the adversary uses the approach with the highest probability in each query.

Using Theorem 2, it is standard to pass to a complexity-theoretic analog. First, replace $Perm(n)$ by a block cipher $E : \mathcal{K} \times \{0,1\}^n \to \{0,1\}^n$, and fix parameters n and $\tau \geqslant 1$. Let $\mathcal{A}$ be an adversary with an oracle and consider that $\mathcal{A}$ performs q queries having aggregate length of σ blocks. Then there is an adversary $\mathcal{B}$ for attacking E with advantage $\mathbf{Adv}^{prp}_{E}(\mathcal{B}) \geqslant \mathbf{Adv}^{prf}_{\textsc{Marv}[E,\tau]}(\mathcal{A}) - \mathbf{Adv}^{prf}_{\textsc{Marv}[Perm(n),\tau]}(\mathcal{A})$. This happens because one strategy for distinguishing an oracle $f = E_K(\cdot)$ (for a random key K) from an oracle $f = \pi(\cdot)$ (where π is a random permutation on n bits) is to run $\mathcal{A}$ and (adaptively) use the responses obtained for the attack. Indeed, we can construct adversary $\mathcal{B}$ as follows: first, $\mathcal{B}$ asks query ($M^0 = [\texttt{0x2a}]_n$) to its own oracle, receiving R as response, and then runs $\mathcal{A}$; when $\mathcal{A}$ makes query M^r, $\mathcal{B}$ computes the corresponding Σ^r, and then asks query (Σ^r) to its own oracle, returning the result, TAG^r, to $\mathcal{A}$. Adversary $\mathcal{B}$ asks at most $q + 1$ queries and has a running time equal to $\mathcal{A}$'s running time plus the time required for computing E on $q + 1$ points plus a time $O(n\sigma)$ that depends on the details of the model of computation.

As a final and important remark, note that $2p$ is considerably lower than 2^{n+1} in practice. Therefore, for a secure block cipher E, the security of Marvin is mostly

determined by the parameter $1/p$, i.e., by the security of the SCTs themselves. This means that the key used for authentication with MARVIN should be replaced long before $q \approx 2^{115/2}$ and $q \approx 2^{72/2}$ queries are made, respectively, for AES and CURUPIRA as underlying ciphers. It is important to stress that, though these numbers are lower than usual for conventional networks, this is most probably high enough for WSNs. This happens because, as noted in (KARLOF; SASTRY; WAGNER, 2004), the reduced throughput of the sensor nodes considerably limits the number of messages they can receive per second. Since the only way to know if a forgery attempt is valid is by sending it to an authorized receiver (the "verification oracle"), this limitation of the sensors hardware prevents adversaries from performing a large number of forgery attempts in a short period of time.

For example, if CURUPIRA-2 is the underlying block cipher, the bandwidth is 250 Kbps (available on motes such as MicaZ and TelosB) and the packets are 21-byte long (a TinySec-Auth packet having the minimal number of bytes for completing 2 blocks of data — 8 bytes of header and 13 bytes of data), an attacker flooding the network with forgery attempts would be able to send less than $2^{10.5}$ packets per second. In this scenario, the attacker would need approximately $2^{25.5}$ seconds, which is equivalent to 549 days, to produce a forgery with probability higher than 50%. Long before that, though, the sensor's battery would be completely depleted, since they are expected to last for about 72 hours if the mote constantly operates in active mode (ZHANG et al., 2004); besides, it would be easy to detect such attacks due to the large number of invalid messages involved and to its "denial-of-service" nature (KARLOF; SASTRY; WAGNER, 2004).

5.5 On the Security of LETTERSOUP

Now we turn our attention to LETTERSOUP's security. Again, we start with an information-theoretic approach: we use the LETTERSOUP$[Perm(n), \tau]$ abstraction, in

which the underlying block cipher is replaced by a pseudo-random permutation.

First, we bound the advantage of an adversary trying to attack the authenticity of this abstraction. Then, we pass to a complexity-theoretic analysis, replacing $Perm(n)$ by a block cipher E and showing that the resulting construction is secure against forgery as long as E is itself secure (i.e., unless there is an adversary that distinguishes E from a random permutation with a high probability). Finally, we build on this discussion to develop our privacy bound, which is similar to the one obtained for authenticity, thus completing our analysis.

5.5.1 Basic Games

Let $\mathcal{A}$ be an adversary that attacks the authenticity of $\Pi = \textsc{LetterSoup}[Perm(n), \tau]$. We assume that $\mathcal{A}$ is computationally unbounded and, without loss of generality, deterministic. Consider that $\mathcal{A}$ has access to an oracle for LetterSoup's encryption function, denoted $\textsc{LetterSoup}.\text{Enc}_\pi(\cdot,\cdot,\cdot)$. We need to bound $\mathbf{Adv}^{auth}_{\Pi}(\mathcal{A})$, the probability that $\mathcal{A}$ forges after performing q adaptive queries, those having aggregate length of σ.

We can model the interaction between $\mathcal{A}$ and this oracle as Game L1, composed by Algorithms 5.5 (when $\mathcal{A}$ gathers information) and 5.6 ($\mathcal{A}$'s forgery attempt). Without loss of generality, in this game and in the games that follow we adopt the following conventions for abbreviated notation: we write "Partition $M = M_1 \| \ldots \| M_t$" as shorthand for "Partition $M = M_1 \| \ldots \| M_t$, where $|M_i| = n$ for $i = 1, \ldots, t-1$, and $0 \leqslant |M_t| \leqslant n$ with $|M_t| = 0$ iff $|M| = 0$"; we assume that $|N| = n$, i.e., the nonce size is exactly one block; we also assume that $|H| > 0$, since the case $|H| = 0$ can be handled with minor modifications in these games, just by making $\Sigma^r \leftarrow \Sigma^r_A$ instead of $\Sigma^r \leftarrow (\Sigma^r_A \oplus \blacksquare(\Sigma^r_D))$ — nonetheless, we consider both cases in our security analysis.

In Game L1, the flag *bad* is set to `true` in two situations: (1) when we carelessly choose the result of $\pi(\cdot)$ from $\{0,1\}^n$ but discover that this random value does not fit, which obliges us to change it accordingly (in Algorithm 5.5, this happens for the nonce

Algorithm 5.5 – Game L1 (Part 1), which accurately simulates LETTER-SOUP$[Perm(n), \tau]$.

Initialization:

$L \xleftarrow{\$} \{0,1\}^n$; $\pi([0]_n) \leftarrow \{L\}$; $bad \leftarrow \texttt{false}$

When $\mathcal{A}$ makes its r-th query (N^r, H^r, M^r) , where $r \in [1 \dots q]$, do:

$X_0^r \leftarrow N^r$; $Y_0^r \xleftarrow{\$} \{0,1\}^n$
if $X_0^r \in Domain(\pi)$ **then** { $bad \leftarrow \texttt{true}$; $Y_0^r \leftarrow \pi(X_0^r)$ } **else**
if $Y_0^r \in Range(\pi)$ **then** {$bad \leftarrow \texttt{true}$; $Y_0^r \xleftarrow{\$} \overline{Range}(\pi)$}
$\pi(X_0^r) \leftarrow Y_0^r$; $R^r \leftarrow Y_0^r \oplus X_0^r$

Partition $M^r = M_1^r \| \dots \| M_{t^r}^r$
for $i \leftarrow 1$ **to** t^r **do**
  $X_i^r \leftarrow R^r \cdot (x^w)^i$; $Y_i^r \xleftarrow{\$} \{0,1\}^n$
  if $X_i^r \in Domain(\pi)$ **then** {$bad \leftarrow \texttt{true}$; $Y_i^r \leftarrow \pi(X_i^r)$} **else**
  if $Y_i^r \in Range(\pi)$ **then** { $bad \leftarrow \texttt{true}$; $Y_i^r \xleftarrow{\$} \overline{Range}(\pi)$ }
  $\pi(X_i^r) \leftarrow Y_i^r$; $C_i^r \leftarrow (M_i^r \oplus Y_i^r)[|M_i^r|]$ ▷ Truncation: applies only to $M_{t^r}^r$
  $A_i^r \leftarrow \blacksquare(\mathbf{rpad}(C_i^r) \oplus X_i^r)$ ▷ Padding: applies only to $C_{t^r}^r$
end for

Partition $H^r = H_1^r \| \dots \| H_{s^r}^r$
for $i \leftarrow 1$ **to** s^r **do** { $Z_i^r \leftarrow \mathbf{rpad}(H_i^r) \oplus L \cdot (x^w)^i$; $D_i^r \leftarrow \blacksquare(Z_i^r)$ }

$A_0^r \leftarrow R^r \oplus \mathbf{rpad}(\mathbf{bin}(n-\tau) \| 1) \oplus \mathbf{lpad}(\mathbf{bin}(|M^r|))$
$D_0^r \leftarrow L \oplus \mathbf{rpad}(\mathbf{bin}(n-\tau) \| 1) \oplus \mathbf{lpad}(\mathbf{bin}(|H^r|))$
$\Sigma_A^r \leftarrow \bigoplus_{i=0}^{t^r} A_i^r$; $\Sigma_D^r \leftarrow \bigoplus_{i=0}^{s^r} D_i^r$; $\Sigma^r \leftarrow (\Sigma_A^r \oplus \blacksquare(\Sigma_D^r))$

if $\Sigma^r \in Domain(\pi)$ **then**
  $bad \leftarrow \texttt{true}$; $TAG^r \leftarrow \pi(\Sigma^r)$
else
  $TAG^r \xleftarrow{\$} \{0,1\}^n$
  if $TAG^r \in Range(\pi)$ **then**
    $bad \leftarrow \texttt{true}$; $TAG^r \xleftarrow{\$} \overline{Range}(\pi)$
  end if
end if
$\pi(\Sigma^r) \leftarrow TAG^r$; $T^r \leftarrow TAG^r[\tau]$
return $CT^r \leftarrow C_1^r \| \dots \| C_t^r \| T^r$

Algorithm 5.6 – Games L1, L1b, L2, L2b and L3 (Part 2): Forgery Attempt

When $\mathcal{A}$ makes forgery attempt (N, H, CT) **, do:**

1: Partition $CT = C_1 \| \ldots \| C_t \| T$
2: $X_0 \leftarrow N$
3: **if** $X_0 \in \mathit{Domain}(\pi)$ **then** $Y_0 \leftarrow \pi(X_0)$ **else** $Y_0 \xleftarrow{\$} \overline{\mathit{Range}}(\pi)$
4: $\pi(X_0) \leftarrow Y_0 \quad ; \quad R \leftarrow Y_0 \oplus X_0$
5:
6: **for** $i \leftarrow 1$ **to** t **do**
7: $\quad X_i \leftarrow R \cdot (x^w)^i$
8: $\quad A_i \leftarrow \blacksquare(\mathbf{rpad}(C_i) \oplus X_i)$ ▷ Padding: applies only to C_t
9: **end for**
10:
11: Partition $H = H_1 \| \ldots \| H_s$
12: **for** $i \leftarrow 1$ **to** s **do** { $Z_i \leftarrow \mathbf{rpad}(H_i) \oplus L \cdot (x^w)^i \quad ; \quad D_i \leftarrow \blacksquare(Z_i)$ }
13:
14: $A_0 \leftarrow R \oplus \mathbf{rpad}(\mathbf{bin}(n - \tau) \| 1) \oplus \mathbf{lpad}(\mathbf{bin}(|C|))$
15: $D_0 \leftarrow L \oplus \mathbf{rpad}(\mathbf{bin}(n - \tau) \| 1) \oplus \mathbf{lpad}(\mathbf{bin}(|H|))$
16: $\Sigma_A \leftarrow \bigoplus_{i=0}^{t} A_i \quad ; \quad \Sigma_D \leftarrow \bigoplus_{i=0}^{s} D_i \quad ; \quad \Sigma \leftarrow (\Sigma_A \oplus \blacksquare(\Sigma_D))$
17: **if** $\Sigma \in \mathit{Domain}(\pi)$ **then** $TAG \leftarrow \pi(\Sigma)$ **else** $TAG \xleftarrow{\$} \overline{\mathit{Range}}(\pi)$
18: **if** $TAG[\tau] = T$ **then** *bad* $\leftarrow$ `true`

X_0 in lines 3 and 4, for the offsets X_i in lines 10 and 11 and for the accumulator Σ in lines 24 and 28); and (2) when the adversary successfully forges (line 18 of Algorithm 5.6). Thus, upper bounding the probability that *bad* gets set to `true` in Game L1 allows us to upper bound $\mathcal{A}$'s forging probability.

Let us make some minor modifications in the first part of Game L1, yielding an "easier" Game L1b. First, we eliminate the truncation in line 12 of Algorithm 5.5, giving the attacker some extra bits of ciphertext: $C_i \leftarrow ((M_i \| 0^*) \oplus Y_i)$. Clearly, this extra piece of information cannot decrease the adversary's probability of forging (in fact, quite the opposite). Second, we remove the tag truncation in line 31 of Algorithm 5.5, i.e., we provide the adversary with a full tag having n bits instead of only τ bits. Once again, this modification can only increase $\mathcal{A}$'s chance of success, not decrease it. We can thus bound $\mathcal{A}$'s probability of forging as the probability of setting *bad* to `true` in this slightly modified Game L1b.

Now let us make some major modifications to Game L1b, yielding Game L2,

shown in Algorithm 5.7. In this new game, we eliminate every statement that immediately follows *bad* being set to `true` in lines 3, 4, 10, 11, 24, and 28 of Algorithm 5.5 (highlighted statements). We also remove the corresponding **else** statements. With these modifications, the responses obtained by adversary $\mathcal{A}$ interacting with Game L2 are clearly different from those given by Game L1b. Nonetheless, the behavior of Game L2 is equivalent to the behavior of Game L1b until *bad* is set to `true`. Therefore, the probability that *bad* gets set to `true` when $\mathcal{A}$ plays Game L2 is identical to the probability that *bad* gets set to `true` when playing Game L1b, and upper bounds the probability that $\mathcal{A}$ forges.

Algorithm 5.7 – Game L2 (Part 1)

Initialization:

1: $L \xleftarrow{\$} \{0,1\}^n \quad ; \quad \pi([0]_n) \leftarrow \{L\} \quad ; \quad bad \leftarrow \texttt{false}$

When $\mathcal{A}$ makes its r-th query (N^r, H^r, M^r) , where $r \in [1 \dots q]$, do:

2: $X_0^r \leftarrow N^r \quad ; \quad Y_0^r \xleftarrow{\$} \{0,1\}^n$
3: **if** $X_0^r \in Domain(\pi)$ **then** $bad \leftarrow \texttt{true}$
4: **if** $Y_0^r \in Range(\pi)$ **then** $bad \leftarrow \texttt{true}$
5: $\pi(X_0^r) \leftarrow Y_0^r \quad ; \quad R^r \leftarrow Y_0^r \oplus X_0^r$
6:
7: Partition $M^r = M_1^r \| \dots \| M_{t^r}^r$
8: **for** $i \leftarrow 1$ **to** t^r **do**
9: $\quad X_i^r \leftarrow R^r \cdot (x^w)^i \quad ; \quad Y_i^r \xleftarrow{\$} \{0,1\}^n$
10: $\quad$ **if** $X_i^r \in Domain(\pi)$ **then** $bad \leftarrow \texttt{true}$
11: $\quad$ **if** $Y_i^r \in Range(\pi)$ **then** $bad \leftarrow \texttt{true}$
12: $\quad \pi(X_i^r) \leftarrow Y_i^r \quad ; \quad C_i^r \leftarrow ((M_i^r \| 0^*) \oplus Y_i^r) \quad ; \quad A_i^r \leftarrow \blacksquare(C_i^r \oplus X_i^r)$
13: **end for**
14:
15: Partition $H^r = H_1^r \| \dots \| H_{s^r}^r$
16: **for** $i \leftarrow 1$ **to** s^r **do** $\{ Z_i^r \leftarrow \mathbf{rpad}(H_i^r) \oplus L \cdot (x^w)^i \quad ; \quad D_i^r \leftarrow \blacksquare(Z_i^r) \}$
17:
18: $A_0^r \leftarrow R^r \oplus \mathbf{rpad}(\mathbf{bin}(n-\tau) \| 1) \oplus \mathbf{lpad}(\mathbf{bin}(|M^r|))$
19: $D_0^r \leftarrow L \oplus \mathbf{rpad}(\mathbf{bin}(n-\tau) \| 1) \oplus \mathbf{lpad}(\mathbf{bin}(|H^r|))$
20: $\Sigma_A^r \leftarrow \bigoplus_{i=0}^{t^r} A_i^r \quad ; \quad \Sigma_D^r \leftarrow \bigoplus_{i=0}^{s^r} D_i^r \quad ; \quad \Sigma^r \leftarrow (\Sigma_A^r \oplus \blacksquare(\Sigma_D^r)) \quad ; \quad T^r \xleftarrow{\$} \{0,1\}^n$
21: **if** $\Sigma^r \in Domain(\pi)$ **then** $bad \leftarrow \texttt{true}$
22: **if** $T^r \in Range(\pi)$ **then** $bad \leftarrow \texttt{true}$
23: $\pi(\Sigma^r) \leftarrow T^r$
24: **return** $CT^r \leftarrow C_1^r \| \dots \| C_t^r \| T^r$

Bounding the probability that *bad* is set to `true` in lines 4, 11, and 22 of Game L2 is simple: in all of them, an n-bit string is chosen at random and then we test for its membership in the set $Range(\pi)$. This set starts with a single element (the one added at line 1 of Algorithm 5.7), and then grows one element at a time until the final size of $2q + \sigma$ elements when line 22 is executed for the last time. Hence, the probability that a repetition occurs as we add a random value to the growing set $Range(\pi)$ is at most $(1 + 2 + \cdots + 2q + \sigma)/2^n \leqslant (1 + 2q + \sigma)^2/2^{n+1}$. If we imagine a new game, denoted L2b, in which we remove these three lines, we can simply write:

$$\mathbf{Adv}^{auth}_{\textsc{LetterSoup}[Perm(n),\tau]}(\mathcal{A}) \leqslant Pr_{L2b}[bad \text{ gets set to } \texttt{true}] + (1 + 2q + \sigma)^2/2^{n+1} \quad (5.8)$$

Now let us turn our attention to the string $CT = C_1 \| \ldots \| C_t \| T$ returned by Game L2b in response to a query (N, H, M). Note that each block C_i of this string is a uniformly distributed random value (Y^r_i, selected in line 9 of Algorithm 5.7) XORed with an independent value (the message block M_i), while T is an uniformly distributed random value (chosen in line 20 of Algorithm 5.7). Therefore, CT itself is an uniformly distributed random string whose only dependency on M refers to its length. This means that, when a query (N, H, M) is performed, we can simply return a random answer CT (where $|CT| = |M| + n$) to the adversary, delaying further tasks to when $\mathcal{A}$ finishes making all q queries; only at that given time, we set the remaining random values, make the corresponding assignments to π (i.e., fill the sets $Domain(\pi)$ and $Range(\pi)$), and check whether *bad* should be set to `true` or not. This is the behavior of Game L3, described in Algorithms 5.8 and 5.6.

From the adversary's point of view, Game L3 is identical to Game L2b. Moreover, *bad* gets set to `true` in both games with the same probability. The analysis of Algorithm 5.8 shows that this probability is taken over a set, $Domain(\pi)$, composed by the values of C^r_i and T^r chosen at the interactive part of the game (line 10), and also by some other values chosen only after $\mathcal{A}$ is done making queries. We need to show that,

over these random values, *bad* is rarely set to `true`.

Analogous to the analysis of MARVIN, we will show something stronger: even if we let $\mathcal{A}$ fix the values of $C_i^r \in \{0,1\}^n$ and $T^r \in \{0,1\}^n$ for every r ($1 \leqslant r \leqslant q$) and i ($0 \leqslant i \leqslant t^r$), and this probability is taken over just the remaining values, the probability that *bad* gets set to `true` remains low. In this case, $\mathcal{A}$ can be imagined as an entity who knows all of the queries (N^r, H^r, M^r) that would be asked and all the corresponding answers (CT^r) that would be received. With this knowledge, $\mathcal{A}$ can simply make a forgery attempt (N, H, M) from the start, skipping the phase where q oracle queries would be made. Note, though, that the value used in this forgery attempt is also fixed in this scenario, because $\mathcal{A}$ is deterministic.

Algorithm 5.8 – Game L3 (Part 1)

When $\mathcal{A}$ makes its r-th query (N^r, H^r, M^r) **, where** $r \in [1 \ldots q]$**, do:**

Partition $M^r = M_1^r \| \ldots \| M_{t^r}^r$
$C_1^r, \ldots, C_t^r, T^r \xleftarrow{\$} \{0,1\}^n$
return $CT^r \leftarrow C_1^r \| \ldots \| C_t^r \| T^r$

When $\mathcal{A}$ is done making queries, do:

$L \xleftarrow{\$} \{0,1\}^n \quad ; \quad \pi([0]_n) \leftarrow \{L\} \quad ; \quad bad \leftarrow \texttt{false}$
for $r \leftarrow 1$ **to** q **do**
    $X_0^r \leftarrow N^r \quad ; \quad Y_0^r \xleftarrow{\$} \{0,1\}^n \quad ; \quad R^r \leftarrow Y_0^r \oplus X_0^r$
    if $X_0^r \in Domain(\pi)$ **then** $bad \leftarrow \texttt{true}$
    $\pi(X_0^r) \leftarrow Y_0^r$
    for $i \leftarrow 1$ **to** t^r **do**
        $X_i^r \leftarrow R^r \cdot (x^w)^i \quad ; \quad Y_i^r \leftarrow ((M_i^r \| 0^*) \oplus C_i^r) \quad ; \quad A_i^r \leftarrow \blacksquare(C_i^r \oplus X_i^r)$
        if $X_i^r \in Domain(\pi)$ **then** $bad \leftarrow \texttt{true}$
        $\pi(X_i^r) \leftarrow Y_i^r$
    end for

    Partition $H^r = H_1^r \| \ldots \| H_{s^r}^r$
    for $i \leftarrow 1$ **to** s^r **do** $\{ Z_i^r \leftarrow \mathbf{rpad}(H_i^r) \oplus L \cdot (x^w)^i \quad ; \quad D_i^r \leftarrow \blacksquare(Z_i^r) \}$

    $A_0^r \leftarrow R^r \oplus \mathbf{rpad}(\mathbf{bin}(n-\tau) \| 1) \oplus \mathbf{lpad}(\mathbf{bin}(|M^r|))$
    $D_0^r \leftarrow L \oplus \mathbf{rpad}(\mathbf{bin}(n-\tau) \| 1) \oplus \mathbf{lpad}(\mathbf{bin}(|H^r|))$
    $\Sigma_A^r \leftarrow \bigoplus_{i=0}^{t^r} A_i^r \quad ; \quad \Sigma_D^r \leftarrow \bigoplus_{i=0}^{s^r} D_i^r \quad ; \quad \Sigma^r \leftarrow (\Sigma_A^r \oplus \blacksquare(\Sigma_D^r))$
    if $\Sigma^r \in Domain(\pi)$ **then** $bad \leftarrow \texttt{true}$
    $\pi(\Sigma^r) \leftarrow T^r$
end for

Now that there is no interactivity, the figure of the adversary is effectively gone. This allows us to build Game L4[φ], in which we eliminate the interactive part of Game L3 and fix all oracle responses . This new game, described in Algorithm 5.9, depends only on constants $\varphi = (q, N^1, H^1, M^1, C^1, T^1, \ldots, N^q, H^q, M^q, C^q, T^q, N, C, T)$. These constants are limited by the amount of resources available to $\mathcal{A}$ and are not completely arbitrary, since the N^r values must be distinct for all r.

From Equation 5.8 and from the previous discussion, we can write:

$$\mathbf{Adv}^{auth}_{\text{LetterSoup}[Perm(n),\tau]}(\mathcal{A}) \leqslant Pr_{L4[\varphi]}[bad \text{ gets set to } \texttt{true}] + (1 + 2q + \sigma)^2/2^{n+1} \quad (5.9)$$

5.5.2 Collisions in Game L4: First Half

Prior to line 21 of Game L4[φ], the flag *bad* is set to `true` whenever:

- *Line 4*: $X_0^r = N^r$ is already in the set $Domain(\pi)$ due to some previous query. Note that we do not need to consider the whole set in this line, since all N^r must be distinct from each other and from $[0]_n$. Indeed, even if we replace $Domain(\pi)$ by its subset $S = Domain(\pi) \setminus \{X_0^1, \ldots, X_0^{r-1}, [0]_n\}$ in this line, *bad* would still be set to `true` with the same probability. In other words, *bad* is set to `true` in this line only when, for $r > u$, some value of $X_0^r = N^r$ matches any of the offsets $X_i^u = R^u \cdot (x^w)^i$ computed at line 7 or any of the accumulator values Σ^u computed at line 16.

- *Line 8*: some offset $X_i^r \leftarrow R^r \cdot (x^w)^i$ is already in $Domain(\pi)$. Again, we do not need to consider the entire set in this line because x^w is a generator of the field we are working with, which means that the offsets do not repeat for a same message (i.e., $X_i^r \neq X_j^r$ for $0 < i < j < 2^n - 1$). Moreover, if $X_i^r = X_j^u$ for some $r > u$ and $i, j \geqslant 2$, then necessarily $X_{i-1}^r = X_{j-1}^u$ — this happens because $R^r \cdot (x^w)^i = R^u \cdot (x^w)^j$ iff $R^r \cdot (x^w)^{i+1} = R^u \cdot (x^w)^{j+1}$. Hence, when we generate the offset X_i^r for some

Algorithm 5.9 – Game L4[φ]: This Game depends on constants $\varphi = (q, N^1, H^1, M^1, C^1, T^1, \ldots, N^q, H^q, M^q, C^q, T^q, N, H, C, T)$, eliminating the interaction with adversaries.

$L \xleftarrow{\$} \{0,1\}^n \quad ; \quad \pi([0]_n) \leftarrow \{L\} \quad ; \quad bad \leftarrow \texttt{false}$
for $r \leftarrow 1$ **to** q **do**
  $X_0^r \leftarrow N^r \quad ; \quad Y_0^r \xleftarrow{\$} \{0,1\}^n \quad ; \quad R^r \leftarrow Y_0^r \oplus X_0^r$
  if $X_0^r \in Domain(\pi)$ **then** $bad \leftarrow \texttt{true}$
  $\pi(X_0^r) \leftarrow Y_0^r$
  for $i \leftarrow 1$ **to** t^r **do**
    $X_i^r \leftarrow R^r \cdot (x^w)^i \quad ; \quad Y_i^r \leftarrow ((M_i^r \,\|\, 0^*) \oplus C_i^r) \quad ; \quad A_i^r \leftarrow \blacksquare(C_i^r \oplus X_i^r)$
    if $X_i^r \in Domain(\pi)$ **then** $bad \leftarrow \texttt{true}$
    $\pi(X_i^r) \leftarrow Y_i^r$
  end for

  for $i \leftarrow 1$ **to** s^r **do** { $Z_i^r \leftarrow \mathbf{rpad}(H_i^r) \oplus L \cdot (x^w)^i \quad ; \quad D_i^r \leftarrow \blacksquare(Z_i^r)$ }

  $A_0^r \leftarrow R^r \oplus \mathbf{rpad}(\mathbf{bin}(n-\tau) \,\|\, 1) \oplus \mathbf{lpad}(\mathbf{bin}(|M^r|))$
  $D_0^r \leftarrow L \oplus \mathbf{rpad}(\mathbf{bin}(n-\tau) \,\|\, 1) \oplus \mathbf{lpad}(\mathbf{bin}(|H^r|))$
  $\Sigma_A^r \leftarrow \bigoplus_{i=0}^{t^r} A_i^r \quad ; \quad \Sigma_D^r \leftarrow \bigoplus_{i=0}^{s^r} D_i^r \quad ; \quad \Sigma^r \leftarrow (\Sigma_A^r \oplus \blacksquare(\Sigma_D^r))$
  if $\Sigma^r \in Domain(\pi)$ **then** $bad \leftarrow \texttt{true}$
  $\pi(\Sigma^r) \leftarrow T^r$
end for

$X_0 \leftarrow N$
if $X_0 \in Domain(\pi)$ **then** $Y_0 \leftarrow \pi(X_0)$ **else** $Y_0 \xleftarrow{\$} \overline{Range}(\pi)$
$\pi(X_0) \leftarrow Y_0 \quad ; \quad R \leftarrow Y_0 \oplus X_0$

for $i \leftarrow 1$ **to** t **do**
  $X_i \leftarrow R \cdot (x^w)^i$
  $A_i \leftarrow \blacksquare(\mathbf{rpad}(C_i) \oplus X_i)$ ▷ Padding: applies only to C_t
end for

for $i \leftarrow 1$ **to** s **do** { $Z_i \leftarrow \mathbf{rpad}(H_i) \oplus L \cdot (x^w)^i \quad ; \quad D_i \leftarrow \blacksquare(Z_i)$ }

$A_0 \leftarrow R \oplus \mathbf{rpad}(\mathbf{bin}(n-\tau) \,\|\, 1) \oplus \mathbf{lpad}(\mathbf{bin}(|C|))$
$D_0 \leftarrow L \oplus \mathbf{rpad}(\mathbf{bin}(n-\tau) \,\|\, 1) \oplus \mathbf{lpad}(\mathbf{bin}(|H|))$
$\Sigma_A \leftarrow \bigoplus_{i=0}^{t} A_i \quad ; \quad \Sigma_D \leftarrow \bigoplus_{i=0}^{s} D_i \quad ; \quad \Sigma \leftarrow (\Sigma_A \oplus \blacksquare(\Sigma_D))$
if $\Sigma \in Domain(\pi)$ **then** $TAG \leftarrow \pi(\Sigma)$ **else** $TAG \xleftarrow{\$} \overline{Range}(\pi)$
if $TAG[\tau] = T$ **then** $bad \leftarrow \texttt{true}$

$r > u$ and $i \geqslant 2$, we do not need to check if $X_i^r = X_j^u$ for any $j \geqslant 2$: if that is the case, *bad* was already set to `true` due to the previous offset, X_{i-1}^r.

- *Line 17*: some accumulator value Σ^r is already in $Domain(\pi)$. Since there is no restriction on the value of Σ^r, nor on the relationship between Σ^r and the other elements of $Domain(\pi)$, we need to consider the entire set in this line.

Using the above observations, we build Game Col1[φ], described in Algorithm 5.10.

Algorithm 5.10 – Game Col1[φ]: This game depends on constants $\varphi = (q, N^1, H^1, M^1, C^1, T^1, \ldots, N^q, H^q, M^q, C^q, T^q)$, and is used to upper bound the probability of *bad* being set to `true` in the first half of Game L4[φ].

1: $L \xleftarrow{\$} \{0,1\}^n$; $Full \leftarrow \{[0]_n\}$; $Sub_0 \leftarrow \{\}$; $Sub_1 \leftarrow \{\}$; $bad \leftarrow$ `false`
2: **for** $r \leftarrow 1$ **to** q **do**
3: $X_0^r \leftarrow N^r$; $Y_0^r \xleftarrow{\$} \{0,1\}^n$; $R^r \leftarrow Y_0^r \oplus X_0^r$
4: **if** $X_0^r \in Sub_0$ **then** $bad \leftarrow$ `true`
5: $Full \leftarrow Full \cup \{X_0^r\}$; $Sub_1 \leftarrow Sub_1 \cup \{X_0^r\}$
6:
7: $X_1^r \leftarrow R^r \cdot (x^w)$; $A_1^r \leftarrow \blacksquare(C_1^r \oplus X_1^r)$
8: **if** $X_1^r \in Full \setminus \{[0]_n\}$ **then** $bad \leftarrow$ `true`
9:
10: **for** $i \leftarrow 2$ **to** t^r **do**
11: $X_i^r \leftarrow R^r \cdot (x^w)^i$; $A_i^r \leftarrow \blacksquare(C_i^r \oplus X_i^r)$
12: **if** $X_i^r \in Sub_1$ **then** $bad \leftarrow$ `true`
13: $Full \leftarrow Full \cup \{X_i^r\}$; $Sub_0 \leftarrow Sub_0 \cup \{X_i^r\}$
14: **end for**
15: $Full \leftarrow Full \cup \{X_1^r\}$; $Sub_0 \leftarrow Sub_0 \cup \{X_1^r\}$; $Sub_1 \leftarrow Sub_1 \cup \{X_1^r\}$
16:
17: **for** $i \leftarrow 1$ **to** s^r **do** { $Z_i^r \leftarrow \mathbf{rpad}(H_i^r) \oplus L \cdot (x^w)^i$; $D_i^r \leftarrow \blacksquare(Z_i^r)$ }
18:
19: $A_0^r \leftarrow R^r \oplus \mathbf{rpad}(\mathbf{bin}(n-\tau) \,\|\, 1) \oplus \mathbf{lpad}(\mathbf{bin}(|M^r|))$
20: $D_0^r \leftarrow L \oplus \mathbf{rpad}(\mathbf{bin}(n-\tau) \,\|\, 1) \oplus \mathbf{lpad}(\mathbf{bin}(|H^r|))$
21: $\Sigma_A^r \leftarrow \bigoplus_{i=0}^{t^r} A_i^r$; $\Sigma_D^r \leftarrow \bigoplus_{i=0}^{s^r} D_i^r$; $\Sigma^r \leftarrow (\Sigma_A^r \oplus \blacksquare(\Sigma_D^r))$
22: **if** $\Sigma^r \in Full$ **then** $bad \leftarrow$ `true`
23: $Full \leftarrow Full \cup \{\Sigma^r\}$; $Sub_0 \leftarrow Sub_0 \cup \{\Sigma^r\}$; $Sub_1 \leftarrow Sub_1 \cup \{\Sigma^r\}$
24: **end for**

Game Col1[φ] uses three different sets, $Full$, Sub_0 and Sub_1: while $Full$ grows exactly as $Domain(\pi)$ in Game L4[φ], both Sub_0 and Sub_1 grow slower because they

do not receive as many elements; namely, for $r \geqslant 1$, only X_i^r ($i \geqslant 1$) and Σ^r are added to Sub_0, while Sub_1 gets only the values of X_0^r, X_1^r and Σ^r. The goal of adopting this growth behavior is to have sets whose size take into account the above-mentioned relationships between values. In this manner, all elements in these sets must be considered when checking for repetitions that can lead to *bad* getting set to `true`. Indeed, the analysis of Game Col1[φ] shows that the probability that *bad* gets set to `true` in lines 4, 8, 12 and 22 of this game is identical to the probability associated to lines 4, 8 and 17 of Game L4[φ]. We will now bound this probability. Let $\alpha^r = \mathbf{rpad}(\mathbf{bin}(n-\tau) \,\|\, 1) \oplus \mathbf{lpad}(\mathbf{bin}(|M^r|))$. First we consider the probability of each possible collision (between X_i^r, Σ^r, etc.):

- *Collisions with* X_0^r: For $i > 0$ and $r > u$, the chance that $X_0^r = X_i^u$ is exactly $1/2^n$. This happens because $Pr[X_0^r = X_i^u] = Pr[X_0^r = (Y_0^u \oplus X_0^u) \cdot (x^w)^i] = Pr[Y_0^u = (X_0^r \cdot (x^w)^{-i}) \oplus X_0^u]$ and, since Y_0^u is selected at random, this probability is $1/2^n$. Moreover, we claim that $Pr[X_0^r = \Sigma^u] \leqslant 1/2^n$. This happens because $Pr[X_0^r = \Sigma^u] = Pr[X_0^r = Y_0^u \oplus X_0^u \oplus \alpha^u \oplus (\bigoplus_{i=1}^{t^u} \blacksquare(A_i^u)) \oplus \blacksquare(\Sigma_D^u)]$, which can be rewritten as $Pr[Y_0^u = \beta \oplus (\bigoplus_{i=1}^{t^u} \blacksquare(A_i^u))]$ for some β independent of Y_0^u. As discussed in section 5.4.2, though the values of A_i^u depend on the random value of Y_0^u, the invertibility of the SCTs leads to a probability that is also bounded by $1/2^n$.

- *Collisions with* X_i^r: For $r > u$ and $i, j \geqslant 1$, we have that $Pr[X_i^r = X_j^u] = Pr[R^r \cdot (x^w)^i = R^u \cdot (x^w)^j] = Pr[Y_0^r \oplus X_0^r = (Y_0^u \oplus X_0^u) \cdot (x^w)^{j-i}] = Pr[Y_0^r = X_0^r \oplus (Y_0^u \oplus X_0^u) \cdot (x^w)^{j-i}]$. This probability is just $1/2^n$ because Y_0^r is random and neither Y_0^u, X_0^r or X_0^u depends on its value. For the same reasons, we also have $Pr[X_i^r = X_i^u] = 1/2^n$. Additionally, we have that $Pr[X_i^r = \Sigma^u] = 1/2^n$ because $Pr[X_i^r = \Sigma^u] = Pr[(Y_0^r \oplus X_0^r) \cdot (x^w)^i = Y_0^u \oplus X_0^u \oplus \alpha^u \oplus (\bigoplus_{i=1}^{t^u} \blacksquare(A_i^u)) \oplus \blacksquare(\Sigma_D^u)] = Pr[Y_0^r = X_0^r \oplus \beta \cdot (x^w)^{-i}]$ for some β independent of Y_0^r, which is again $1/2^n$ because Y_0^r is random and X_0^r does not depend on its value.

- *Collisions with* Σ^r: For the reasons discussed in section 5.4.2, we have that $Pr[\Sigma^r = [0]_n] \leqslant 1/2^n$. Using an analogous argument, we can also argue that $Pr[\Sigma^r = \Sigma^u] \leqslant 1/2^n$. This result comes from the fact that $Pr[\Sigma^r = \Sigma^u] = Pr[Y_0^r \oplus X_0^r \oplus \alpha^r \oplus (\bigoplus_{i=1}^{t^r} \blacksquare(A_i^r)) \oplus \blacksquare(\Sigma_D^r) = Y_0^u \oplus X_0^u \oplus \alpha^u \oplus (\bigoplus_{i=1}^{t^u} \blacksquare(A_i^u)) \oplus \blacksquare(\Sigma_D^u)]$. Even if we remove the SCTs from the algorithm and let the adversary freely choose the values of Σ_D^r and Σ_D^r (without knowing Y_0^r and Y_0^u), we still have that:

$$
\begin{aligned}
&Pr[\Sigma^r = \Sigma^u] = \\
&Pr[\ Y_0^r \oplus X_0^r \oplus \beta^r \oplus (\textstyle\bigoplus_{i=1}^{t^r} C_i^r \oplus (Y_0^r \oplus X_0^r) \cdot (x^w)^i) = \\
&\qquad Y_0^u \oplus X_0^u \oplus \beta^u \oplus (\textstyle\bigoplus_{i=1}^{t^u} C_i^u \oplus (Y_0^u \oplus X_0^u) \cdot (x^w)^i)] = \\
&Pr[(Y_0^r \oplus X_0^r)(1 + x^w + \cdots + (x^w)^{t^r}) \oplus \beta^r \oplus (C_1^r \ldots C_{t^r}^r) = \\
&\qquad (Y_0^u \oplus X_0^u)(1 + x^w + \cdots + (x^w)^{t^u}) \oplus \beta^u \oplus (C_1^u \ldots C_{t^u}^u)]
\end{aligned}
$$

  and this probability would be just $1/2^n$ because, though the adversary has complete control over X_0^r, X_0^u, β^r, β^u and C_i^r for all i, Y_0^r is random and independent of the (random) value of Y_0^u previously chosen. As discussed in section 5.4.2, adding the SCTs back to the algorithm would not raise this probability, which justifies our claim. Finally, as previously discussed, $Pr[\Sigma^r = X_i^u] = 1/2^n$ for all $r > u$ and $i \geqslant 0$.

Now we are ready to consider how the above probabilities affect each of the lines in Game Col1 where *bad* can be set to `true`:

- *Line 4*: For $r > u$ and $i \geqslant 1$, there is a collision in this line if $X_0^r = X_i^u$, or if $X_0^r = \Sigma^u$; this leads to a probability of $1/2^n$ for each element in Sub_0. When line 4 is reached for the first time, Sub_0 has 0 elements (no element is added at line 1). In each query r, Sub_0 grows by $(1 + t^r)$ elements (at lines 13, 15 and 23), and $(q - 1 + \sigma - t^q)$ elements exist when line 4 is reached for the last time. One can upper bound this probability by making $t^1 = \sigma$ and letting the remaining

$t^r = 0$. Thus, we can write:

$$\begin{aligned} Pr_{Col1}[bad \leftarrow \texttt{true} \text{ in line 4}] &\leqslant (0 + (1 + t^1) + \cdots + (q - 1 + \sigma - t^q))/2^n \\ &\leqslant q(q-1)/2^{n+1} + (q-1)\sigma/2^n \\ &\leqslant (q-1)(q + 2\sigma)/2^{n+1} \end{aligned} \tag{5.10}$$

- *Line 8*: For $r > u$ and $i \geqslant 0$, there is a collision in this line if $X_1^r = X_0^r$, if $X_1^r = X_i^u$ or if $X_1^r = \Sigma^u$; hence, this happens with probability $1/2^n$ for each element in $Full \setminus \{[0]_n\}$. When line 8 is reached for the first time, $Full \setminus \{[0]_n\}$ has 1 element (the X_0^1 added at line 5). In each query r, this set grows by $(2 + t^r)$ elements (at lines 5, 13, 15 and 23) before line 8 is reached again. When this line is reached for the last time, $Full \setminus \{[0]_n\}$ contains $(2q - 1 + \sigma - t^q)$ elements. Again, one can bound this probability by making $t^1 = \sigma$ and the remaining $t^r = 0$, allowing us to write:

$$\begin{aligned} Pr_{Col1}[bad \leftarrow \texttt{true} \text{ in line 8}] &\leqslant (1 + (3 + t^1) + \cdots + (2q - 1 + \sigma - t^q))/2^n \\ &\leqslant (1 + 2q - 1)q/2^{n+1} + (q-1)\sigma/2^n \\ &\leqslant (q^2 + (q-1)\sigma)/2^n \end{aligned} \tag{5.11}$$

- *Line 12*: For $r \geqslant u$, there is a collision in this line if $X_i^r = X_0^u$ or if $X_i^r = X_1^u$; there is also a collision if $X_i^r = \Sigma^u$ for $r > u$; hence, this happens with probability $1/2^n$ for each element in Sub_1. When line 12 is reached for the first time, Sub_1 has 1 element (the X_0^1 added at line 5). In each query r, Sub_1 grows by 3 elements (at lines 5, 15 and 23) before line 12 is reached again; finally, when this line is reached for the last time, Sub_1 contains $(3q - 2)$. Unlike the other lines, though, note that line 8 is reached $(t^r - 1)$ times (one for each X_i^r) during query r. Thus,

by making $t^q = \sigma$ and the remaining $t^r = 0$, we can bound this probability as:

$$\begin{aligned} Pr_{Col1}[bad &\leftarrow \texttt{true} \text{ in line 12}] \\ &\leqslant (1(t^1 - 1) + 4(t^2 - 1) + \cdots + (3q - 2)(t^q - 1))/2^n \\ &\leqslant (3q - 2)(\sigma - 1)/2^n \end{aligned} \tag{5.12}$$

- *Line 22*: A collision happens in three cases: if $\Sigma^r = [0]_n$, if $\Sigma^r = X_i^u$ ($r \geqslant u$, $i \geqslant 0$), or if $\Sigma^r = \Sigma^u$ ($r > u$); thus, the probability a collision is simply $1/2^n$ for each element in $Full$. When line 22 is reached for the first time, $Full$ has $2 + t^1$ elements (more specifically, $Full = \{[0]_n, X_0^1, \ldots, X_{t^r}^1\}$), but each query raises the number of elements in this set by $2 + t^r$ until the size of $2q + \sigma$. We can again bound the probability for all queries by making $t^1 = \sigma$ and the remaining $t^r = 0$:

$$\begin{aligned} Pr_{Col1}[bad \leftarrow \texttt{true} \text{ in line 22}] &\leqslant ((2 + t^1) + \cdots + (2q + \sigma))/2^n \\ &\leqslant (2 + 2q)q/2^{n+1} + q\sigma/2^n \\ &\leqslant q(q + \sigma + 1)/2^n \end{aligned} \tag{5.13}$$

Finally, by combining the Equations 5.10, 5.11, 5.12 and 5.13, we arrive at:

$$Pr_{Col1}[bad \leftarrow \texttt{true}] \leqslant (5q^2 + 12q\sigma - 8\sigma - 5q + 4)/2^{n+1} \tag{5.14}$$

5.5.3 Collisions in Game L4: Second Half

In order to quantify the probability of setting *bad* to `true` in line 36 of Game L4[φ], we build Game Col2[φ], shown in Algorithm 5.11, by introducing the following modifications in the second half of Game L4[φ] (starting at line 21):

- We simplify lines 22 and 35, assigning a random value from $\{0,1\}^n$ to Y_0 and to TAG instead of some value from the $\overline{Range}(\pi)$. As discussed for Games L1 and L2, these modifications reduce the probability that *bad* is set to `true`, but by at most $(\sigma + 2q + 1)/2^n + (\sigma + 2q + 2)/2^n = (2\sigma + 4q + 3)/2^n$.

- We modify the last two lines of the game in such a manner that we "give up" earlier: we set *bad* to `true` whenever the condition of the **if** clause in line 35 is satisfied, and we suppress the subsequent statements. Hence, we remove both the **else** clause that was previously modified (i.e., the clause $TAG \xleftarrow{\$} \{0,1\}^n$) and the entire line 36. Again, this may reduce the probability that *bad* gets set to `true`. However, this reduction is at most $1/2^\tau$ because, when the suppressed **else** clause is executed, the probability that $TAG[\tau] = T$ is exactly $1/2^\tau$ due to the randomness of TAG.

- We modify line 22, setting *bad* to `true` whenever the nonce N used in the forgery attempt differs from the previously used nonces but N is already in $Domain(\pi)$. Note that this modification does not reduce the probability of setting *bad* to `true`.

Algorithm 5.11 – Game Col2[φ] (Part 2): This game depends on constants $\varphi = (q, N^1, H^1, M^1, C^1, T^1, \ldots, N^q, H^q, M^q, C^q, T^q, N, H, C, T)$, and its Part 1 corresponds to lines 1 to 19 of Game L4[φ]. It is used to upper bound the probability of *bad* being set to `true` in the second half of Game L4[φ].

1:
2: $X_0 \leftarrow N$
3: **if** $X_0 \neq X_0^r$ for all r and $X_0 \in Domain(\pi)$ **then** *bad* $\leftarrow$ `true`
4: **if** $X_0 = X_0^r$ for some r **then** $Y_0 \leftarrow Y_0^r$ **else** $Y_0 \leftarrow \{0,1\}^n$
5: $\pi(X_0) \leftarrow Y_0 \quad ; \quad R \leftarrow Y_0 \oplus X_0$
6:
7: **for** $i \leftarrow 1$ **to** t **do**
8: $\quad X_i \leftarrow R \cdot (x^w)^i$
9: $\quad A_i \leftarrow \blacksquare(\mathbf{rpad}(C_i) \oplus X_i)$ ▷ Padding: applies only to C_t
10: **end for**
11:
12: **for** $i \leftarrow 1$ **to** s **do** { $Z_i \leftarrow \mathbf{rpad}(H_i) \oplus L \cdot (x^w)^i \quad ; \quad D_i \leftarrow \blacksquare(Z_i)$ }
13: $A_0 \leftarrow R \oplus \mathbf{rpad}(\mathbf{bin}(n-\tau) \,\|\, 1) \oplus \mathbf{lpad}(\mathbf{bin}(|C|))$
14: $D_0 \leftarrow L \oplus \mathbf{rpad}(\mathbf{bin}(n-\tau) \,\|\, 1) \oplus \mathbf{lpad}(\mathbf{bin}(|H|))$
15: $\Sigma_A \leftarrow \bigoplus_{i=0}^{t} A_i \quad ; \quad \Sigma_D \leftarrow \bigoplus_{i=0}^{s} D_i \quad ; \quad \Sigma \leftarrow (\Sigma_A \oplus \blacksquare(\Sigma_D))$
16: **if** $\Sigma \in Domain(\pi)$ **then** *bad* $\leftarrow$ `true`

Game Col2[φ] sets the flag *bad* to `true` in line 3 only if the value of N, different from any other nonce N^r ($1 \leqslant r \leqslant q$), is already in $Domain(\pi)$. The probability of this

event is no different from the probability of having collisions with X_0^r in Game Col1[φ]. Since this is just $1/2^n$ for each element in $Domain(\pi) \setminus \{[0]_n\}$ when this line is executed (remember that $[0]_n$ is a reserved value), we can write the following equation:

$$Pr_{Col2}[bad \leftarrow \texttt{true} \text{ in line 3}] \leqslant (\sigma + 2q)/2^n \tag{5.15}$$

Finally, *bad* is set to `true` in line 16 when the accumulator value Σ resulting from the processing of (N, H, C) is already in $Domain(\pi)$. If $N \neq N^r$ for all r, then the probability associated to this event is equivalent to the probability of having Collisions with Σ^r in Game Col1[φ], which is just $1/2^n$ for each element in $Domain(\pi)$. If $N = N^r$ for some r, though, the same probability does not apply. We still can show that each of the probabilities $Pr[\Sigma = [0]_n]$, $Pr[\Sigma = X_0]$, $Pr[\Sigma = X_0^u]$ and $Pr[\Sigma = X_i^u]$ ($1 \geqslant u \geqslant q$ and $i \geqslant 1$) is at most $1/2^n$, just by using the same reasoning as before. However, $Pr[\Sigma = \Sigma^r]$ is at most $\max\{1/2^n, 1/p\} = 1/p$, where $1/p$ denotes the maximum probability of any differential constructed over an SCT. This happens because the nonce-repetition allows the adversary to exploit accumulation collisions in LetterSoup, leading to the following situations:

- Exploiting the message blocks only: like in Marvin, the adversary can construct M by adding blocks to M^r or by modifying some of its blocks. However, this will lead to $\Sigma = \Sigma^r$ only if the adversary is able to build M in such a manner that $\Sigma_A = \Sigma_A^r$. Using a similar reasoning as in sections 5.4.3 and 5.4.4, this happens with a probability if at most $1/p$, where p denotes the probability of any differential constructed over an SCT. Therefore, in this case we have that $Pr[\Sigma = \Sigma^r] \leqslant 1/p$.

- Exploiting the header blocks only: if $H^r \neq \varepsilon$, the adversary can construct H by adding blocks to H^r or by modifying some of its blocks. Analogous to the above scenario, this will lead to $\Sigma = \Sigma^r$ only if the adversary is able to build H in such a manner that $\Sigma_D = \Sigma_D^r$, which happens with a probability of at most $1/p$. On the

other hand, if $H^r = \varepsilon$, the adversary would have to build H in such a manner that $\Sigma_D = \blacksquare^{-1}([0]_n)$; using a similar reasoning as in section section 5.4.2, one can show that this happens with a probability of at most $1/2^n$. Hence, this approach is not more effective than exploring the blocks of M^r itself, leading once again to $Pr[\Sigma = \Sigma^r] \leqslant \max\{1/2^n, 1/p\} \leqslant 1/p$.

- Exploiting both message and header blocks: the adversary can build M by manipulating M^r, and try to compensate the resulting $\Delta_A \neq 0$ difference in Σ^r_A by manipulating H. If $H^r = \varepsilon$, we have that $Pr[\Sigma = \Sigma^r] = Pr[\Sigma_A \oplus \blacksquare(\Sigma_D) = \Sigma^r_A] = Pr[\Sigma^r_A \oplus \Delta_A \oplus \blacksquare(\Sigma_D) = \Sigma^r_A] = Pr[\Sigma_D = \blacksquare^{-1}(\Delta_A)]$; since Δ_A does not depend on Σ_D, we can use a similar reasoning as in section 5.4.2 to show that this probability is at most $1/2^n$. If $H^r \neq \varepsilon$, then $Pr[\Sigma = \Sigma^r] = Pr[\Sigma^r_A \oplus \Delta_A \oplus \blacksquare(\Sigma_D) = \Sigma^r_A \oplus \blacksquare(\Sigma^r_D)] = Pr[\blacksquare(\Sigma_D) \oplus \blacksquare(\Sigma^r_D) = \Delta_A]$; the exact values of Σ_D and Σ^r_D cannot be chosen by the adversary because of the random value L combined with them, and even if $\mathcal{A}$ is able to choose the difference $\Delta_D = \Sigma_D \oplus \Sigma^r_D$ (note that this is difficult because SCTs are non-linear), this probability is at most $1/p$ by the definition of p. Therefore, in this case we have again that $Pr[\Sigma = \Sigma^r] \leqslant \max\{1/2^n, 1/p\} \leqslant 1/p$.

As $Domain(\pi)$ has at most $\sigma + 2q + 2$ elements when line 16 is achieved, and only one of them may correspond to a Σ^r obtained from a repeated nonce value, we can write:

$$Pr_{Col2}[bad \leftarrow \texttt{true} \text{ in line } 16] \leqslant (\sigma + 2q + 1)/2^n + 1/p \tag{5.16}$$

From equations 5.15 and 5.16, we have that:

$$\begin{aligned} Pr_{Col2}[bad \leftarrow \texttt{true}] &\leqslant (2\sigma + 4q + 3 + \sigma + 2q + \sigma + 2q + 1)/2^n + 1/2^\tau + 1/p \\ &\leqslant (4\sigma + 8q + 4)/2^n + 1/2^\tau + 1/p \end{aligned} \tag{5.17}$$

5.5.4 Conclusions: Authenticity

By combining Equation 5.9 with the probabilities given in Equations 5.14 and 5.17, we can finally state the following theorem:

Theorem 3. ***[Security of* LetterSoup*: Authenticity]*** *Fix n, $\tau \geqslant 1$. Let $\mathcal{A}$ be an adversary with access to an oracle, and let $1/p$ be the maximum probability of any differential constructed on an SCT. Consider that $\mathcal{A}$ performs q queries to its oracle, and let σ denote the aggregate length of these queries. Then:*

$$\begin{aligned}
&\mathbf{Adv}^{auth}_{\text{LetterSoup}[Perm(n),\tau]}(\mathcal{A}) \\
&\qquad \leqslant (\sigma^2 + 16q\sigma + 9q^2 + 2\sigma + 15q + 14)/2^{n+1} + 1/2^{\tau} + 1/p \\
&\qquad \leqslant (3\sigma^2 + 12q\sigma + 11q^2 + 2\sigma + 16q + 16)/2^{n+1} + 1/2^{\tau} + 1/p \\
&\qquad \leqslant (1.5\sigma^2 + 6q\sigma + 5.5q^2 + \sigma + 8q + 8)/2^{n} + 1/2^{\tau} + 1/p \\
&\qquad \leqslant 1.5(\sigma + 2q + 3)^2/2^{n} + 1/2^{\tau} + 1/p
\end{aligned}$$

For the second inequality, we used the fact that $(\sigma - q)^2 \geqslant 0$ (and, thus, $\sigma^2 + q^2 \geqslant 2q\sigma$), and added some extra resources to the right hand of the equation ($q + 2$, to be specific).

Note that the presence of the $1/p$ factor in the resulting equation leads to a lower security against forgery than that observed in many conventional authentication schemes (e.g., OCB). Nonetheless, as discussed in section 5.4.5, exploiting this factor would require a large number of messages, which is difficult in the context of WSNs due to the reduced sensors' throughput. Furthermore, tags in WSNs are typically as small as $\tau = 4$ bytes (KARLOF; SASTRY; WAGNER, 2004; LUK et al., 2007), meaning that $1/2^{\tau} = 2^{-32}$, which is already much higher than the factor $1/p = 2^{-114}$ or $1/p = 2^{-71}$ for AES and Curupira, respectively; in this context, it would be much easier to perform exhaustive forgery attempts instead of trying to exploit accumulation collisions.

Using Theorem 3, we can again pass to a complexity-theoretic analog. First, replace $Perm(n)$ by a block cipher $E : \mathcal{K} \times \{0,1\}^n \to \{0,1\}^n$, and fix parameters n and $\tau \geqslant$

1. Let $\mathcal{A}$ be an adversary with access to an oracle and consider that $\mathcal{A}$ asks q queries with a total aggregate length of σ blocks. Then there is an adversary $\mathcal{B}$ for attacking E that achieves advantage $\mathbf{Adv}^{prp}_{E}(\mathcal{B}) \geqslant \mathbf{Adv}^{auth}_{\textsc{LetterSoup}[E,\tau]}(\mathcal{A}) - \mathbf{Adv}^{auth}_{\textsc{LetterSoup}[Perm(n),\tau]}(\mathcal{A})$. Adversary $\mathcal{B}$ asks at most $q' = \sigma + 2q + 3$ queries to its own oracle and has a running time equal to $\mathcal{A}$'s running time plus the time required for computing E on q' points plus a time $O(n\sigma)$ that depends on the details of the model of computation.

5.5.5 Conclusions: Privacy

Using the previous discussion, it is straightforward to bound the adversary's advantage when attacking the privacy of $\textsc{LetterSoup}[Perm(n), \tau]$. For this, we use games L1 to L4 as previously specified, except that:

- We omit the second half of each game, corresponding to the forgery attempt, since the adversary does not try to forge in this context;
- For each query, we return the truncated ciphertext blocks instead of the full ciphertext blocks.

We identify the games obtained with these modifications by using the '^{p}' upperscript.

Consider Game L3^p, in which we return to the adversary a random string of length $|M^r| + \tau$ as the response to a query M^r. This game corresponds exactly to the behavior of Game L1^p, unless the flag *bad* is set to `true` at some point of Game L3^p. This means that we can bound $\mathbf{Adv}^{priv}_{\textsc{LetterSoup}[Perm(n),\tau]}(\mathcal{A})$ by bounding the probability that *bad* is set to `true` is Game L3^p, which is at most the probability that this happens in Game L4$^p[\varphi]$ (or, equivalently, in game Col1$[\varphi]$).

Using the same reasoning as before, we arrive at the following theorem:

Theorem 4. ***[Security of* LetterSoup*: Privacy]*** *Fix n, $\tau \geqslant 1$. Let $\mathcal{A}$ be an adversary with access to an oracle. Consider that $\mathcal{A}$ performs q queries, and let σ denote the aggregate length of these queries. Then:*

$$\begin{aligned}
&\mathbf{Adv}^{priv}_{\text{LetterSoup}[Perm(n),\tau]}(\mathcal{A}) \\
&\qquad \leqslant (\sigma^2 + 16q\sigma + 9q^2 - 6\sigma - q + 5)/2^{n+1} \\
&\qquad \leqslant (3\sigma^2 + 12q\sigma + 11q^2)/2^{n+1} \\
&\qquad \leqslant (1.5\sigma^2 + 6q\sigma + 5.5q^2)/2^{n} \\
&\qquad \leqslant 1.5(\sigma + 2q)^2/2^{n}
\end{aligned}$$

Once again, we can pass to a complexity-theoretic analog of Theorem 4. First, replace $Perm(n)$ by a block cipher $E : \mathcal{K} \times \{0,1\}^n \to \{0,1\}^n$, and fix parameters n and $\tau \geqslant 1$. Let $\mathcal{A}$ be an adversary that asks q queries, these having an aggregate length of σ blocks. Then there is an adversary $\mathcal{B}$ for attacking E that achieves advantage $\mathbf{Adv}^{prp}_{E}(\mathcal{B}) \geqslant \mathbf{Adv}^{priv}_{\text{LetterSoup}[E,\tau]}(\mathcal{A}) - \mathbf{Adv}^{priv}_{\text{LetterSoup}[Perm(n),\tau]}(\mathcal{A})$. Adversary $\mathcal{B}$ performs at most $q' = \sigma + 2q + 1$ oracle queries and $\mathcal{B}$'s running time is equal to $\mathcal{A}$'s running time plus the time required for computing E on q' points plus a time $O(n\sigma)$.

5.5.6 On the Effect of IV Repetition over AEAD Schemes

It is important to stress that the security of most existing AEAD solutions (including LetterSoup) depend on the non-repeatability of the IVs for a same key. The effects of IV repetition is especially disastrous when the encryption is performed using a stream mode of operation (e.g., CTR or LFSRC): in schemes such as EAX, GCM and LetterSoup, two messages M_1 and M_2 yield ciphertexts C_1 and C_2 in such a manner that $C_1 \oplus C_2 = M_1 \oplus M_2$; the same behavior is observed for the last blocks of OCB only if they have the same length, while the effect of IV repetition for all other blocks (encrypted under an ECB-like mode) is that identical plaintexts result in identical ciphertexts. In contrast, IV repetition in a CBC-like encryption mode — such as the one adopted in TinySec (KARLOF; SASTRY; WAGNER, 2004) — allows attackers to gain a

minimal amount of information about the plaintexts, namely the length of their longest shared prefix, in blocks.

IV repetition can be avoided by using sufficiently large IVs, e.g., by taking their values from monotonically increasing counters having the size of a block. However, this strategy must be adopted with care, since adding large IVs to each packet transmitted will inevitably increase energy consumption and thus reduce the sensors lifespan. There are, though, some techniques for addressing this issue. For example, instead of sending the whole IVs in every packet, both sender and receiver could keep a synchronized counter, incremented at the reception of a packet, and from which the IV value is taken. This strategy is adopted by SNEP (PERRIG et al., 2001), in which the counter value corresponds to the whole IV and, thus, no IV is sent; it is also used by MiniSec (LUK et al., 2007), in which the packets include a few bits of the IV, thus facilitating resynchronization when some packets are lost. It is also possible to reuse some of the packet header fields as part of the IVs, as done in TinySec and SenSec (LI et al., 2005): both solutions reuse 4 bytes of the header for composing the IV. Finally, before the IV repetition occurs, rekeying mechanisms such as those proposed in (PARK; SHIN, 2004; ELTOWEISSY; MOHARRUM; MUKKAMALA, 2006; ZHANG; ZHU; CAO, 2009) should be employed in order to update the nodes's keys.

5.6 Summary

In this chapter, we developed a detailed security analysis of MARVIN and LETTERSOUP. Using the Game-Playing technique, we have bounded the adversary's advantage when trying to forge a valid tag using MARVIN, to forge a valid attack using LETTERSOUP's (i.e., attack the scheme's authenticity) and to distinguish the ciphertext produced by LETTERSOUP from a random sequence of bits (i.e., attack its privacy). According to our results, both schemes are secure as long as (1) the underlying block cipher E adopted is a good pseudo-random permutation, and (2) the SCT created from

E has a low maximum differential probability.

Indeed, the security analysis of practical SCTs shows that the level of security provided by ciphers based on the Wide Trail Strategy (DAEMEN; RIJMEN, 2001) remains reasonably high: with SCTs constructed from AES (a 128-bit block cipher), the security level provided by our solutions is comparable to that of conventional schemes adopting a 114-bit block cipher; if AES is replaced by the WSN-oriented cipher CURUPIRA-2 (which adopts 96-bit blocks), then the same comparison could be made with a conventional scheme using a 71-bit block cipher. Even in the latter case, an attacker using the whole bandwidth of a TelosB mote to check the validity of the forgery attempts would still need about 550 days in order to produce a valid tag with probability higher than 50%.

In the next chapter, we turn our attention to the efficiency of the proposed algorithms, comparing them with existing schemes in a qualitative and quantitative manner.

6 PERFORMANCE EVALUATION

In the previous chapter, we provided a detailed security analysis of the proposed MAC and AEAD solutions, MARVIN and LETTERSOUP. In this chapter, we evaluate their performance both in a constrained device (a mote) and in a powerful platform (a modern PC), and provide a comparison with existing schemes.

We start our analysis with a review of the general features presented by MARVIN, LETTERSOUP and other modern algorithms. This way, we provide a theoretical and qualitative comparison between these solutions. Afterward, we present our benchmark results, comparing the performance of real implementations in a quantitative manner.

6.1 Preliminary Evaluation

Ignoring setup time, the computational cost to process one message block with a Galois-Carter-Wegman MAC functions like GMAC can be as small as that needed to compute a few rounds of a block cipher. This is achieved by using w-bit Look-Up Tables (LUTs), which may be very large: $O(2^w b)$ n-bit blocks of extra storage, or about 64 KiB per key for a 128-bit underlying block cipher and 8-bit lookup indexes. In storage-constrained environments the cost of a plain GMAC implementation is likely to be comparable to more than a full encryption. In contrast, MARVIN typically consists of 2 rounds of a SHARK-like cipher or 4 rounds of a SQUARE-like cipher. Although this is potentially slower than GMAC, MARVIN does not require any extra storage space, since it simply reuses part of the block cipher implementation that is al-

ready available. Therefore, on storage-constrained platforms MARVIN can easily match Galois-Carter-Wegman schemes and potentially overcome them. An analogous discussion applies to LETTERSOUP when compared to GCM, but the latter displays some extra limitations such as the need of processing the whole plaintext data before the ciphertext in a parallelizable implementation.

For very short messages, MARVIN is expected to be slower than some conventional constructions that need a full cipher invocation per message block, such as PMAC1 and CMAC. This happens because MARVIN requires 1 SCT per authenticated block plus one final encryption, while PMAC1 and CMAC require only one encryption per block. Hence, for messages smaller than one block, MARVIN would require about 1.25–1.4 encryption, while PMAC1 and CMAC would require only 1 encryption. For messages larger than one block, though, MARVIN should be advantageous and, in the long run, about 2.5–4 times faster than conventional MACs. Furthermore, contrary to strictly sequential MAC constructions like CMAC or Pelican, MARVIN is fully parallelizable.

Table 8: Comparison of MAC algorithms, considering an n-bit (b-byte) underlying block cipher.

	CMAC	PMAC1	GMAC (LUTs)	GMAC (plain)	MARVIN
Tag length (bits)	0 to n	0 to n	0 to n	0 to n	0 to n
Block size (bits)	any	any	64 or 128	64 or 128	any
Encryptions/block	1	1	($\approx$ 0.1–0.25)	> 1	$\approx$ 0.25–0.4
Storage (blocks)	$O(1)$	$O(1)$	$O(2^8 b)$	$O(1)$	$O(1)$
Parallelizable?	No	Yes	Yes	Yes	Yes
Requires a nonce?	No	No	Yes	Yes	No
Patents?	No	Pending	No	No	No

Analogously, the authentication process using LETTERSOUP requires two encryptions per message (one for computing the offset seed R and the other for the final tag computation) plus 1 SCT per authenticated block. When compared to EAX with precomputed constants, whose authentication process requires 1 encryption per block, LETTERSOUP should be advantageous when more than 2 blocks (confidential message, header or both) are authenticated.

Conversely, OCB requires a full encryption per block of associated data (since this part of the message is handled by the PMAC1 algorithm), but introduces little overhead when processing the ciphertext blocks (basically, it requires 2 XORs and one multiplication by x in GF(2^n)). Hence, when compared to OCB, LetterSoup is expected to deliver superior performance only in situations where the size of the header is on the order of the confidential message's length (i.e., when the $|M|/|H|$ is not too big). Nonetheless, LetterSoup displays some important advantages over OCB, like a reduced memory footprint (no need to implement E_K^{-1}), the absence of patents, and the possibility of verifying the tag prior to the message decryption.

Table 9: Comparison of AEAD schemes, considering an n-bit (b-byte) underlying block cipher and τ-bit tags. The MAC cost is computed as the number of extra encryptions needed for the authentication process; thus, it depends on the $|H|/|M|$ ratio for OCB and on the type of LUTs used for GCM.

	EAX	OCB	GCM	CS	LetterSoup
#Passes	Two	One	Two	One	Two
Tag length (bits)	0 to n	0 to n	0 to n	0 to n	0 to n
Block size (bits)	Any	Any	64 or 128	Any	Any
IV size (bits)	Any	n	Any (favored: $n - 32$)	n	n
Requires non-repeating IV?	Yes	Yes	Yes	Yes	Yes
Encryption Mode	CTR	Tweaked ECB	CTR	Tweaked ECB	LFSRC
Header Handling	CMAC	PMAC1	GHASH	none	Marvin
Input order	Free	Free	Partially	-	Free
Tag Verifiable Before Decryption?	Yes	No	Yes	Partially	Yes
MAC cost	1	≈ 0.1 to 1	≈ 0.1–0.25 to > 1	≈ 0.1	≈ 0.25–0.4
Storage (blocks)	$O(1)$	$O(1)$	$O(2^8 b)$ to $O(1)$	$O(1)$	$O(1)$
Parallelizable?	No	Yes	Yes	Yes	Yes
Only E_K required?	Yes	No	Yes	No	Yes
Minimal Data Expansion?	Yes	Yes	Yes	No	Yes
Patents?	No	in U.S.	No	No	No

Table 8 summarizes the features of the authentication schemes discussed in this document, while Table 9 does the same for the AEAD schemes.

6.2 Benchmark Methodology

In this section, we detail the methodology adopted in our benchmark analysis, which focus on the comparison between our algorithms and some relevant general-purpose solutions.

6.2.1 Platforms and Operating Systems

We chose the following platforms as testbeds for our benchmarks:

- TelosB: the Crossbow TelosB (CROSSBOW, 2008c) sensor node is equipped with a 16-bit 8MHz micro-controller (Texas Instruments MSP430) having 10 KiB of RAM (data memory) and 48 KiB of flash memory (code memory). The operating system used in our analysis is TinyOS (LEVIS et al., 2004).

- PC: a desktop equipped with a 32-bit 2.4 GHz Intel Pentium Quad Core and 2GiB of RAM running Windows XP.

TinyOS (LEVIS et al., 2004) is a lightweight, event-driven operating system for sensor nodes that was originally developed as a research project at the University of California Berkeley. Nowadays it is maintained by an open-source community[1], and has become the *de facto* standard for WSNs. Its memory requirements depend on the libraries used by the specific application developed, but the base system occupies about 400 bytes of code memory (LEVIS et al., 2004).

TinyOS is based on the nesC programming language (GAY et al., 2003b; GAY et al., 2003a), a component-oriented extension of C. The components in WSNs usually are abstractions of the mote's hardware modules (radio interface, sensors, LEDs, etc.).

[1] http://www.tinyos.net/

6.2.2 Algorithms and Implementation Details

We have chosen the following algorithms to be part of our performance evaluation:

MACs Algorithms:

1. MARVIN: our MAC proposal.
2. CMAC: chosen because it is a NIST recommendation; moreover, CMAC is probably the most straightforward substitute for the CBC-MAC variant adopted by TinySec (KARLOF; SASTRY; WAGNER, 2004), since both have very similar structures, and thus should provide some insight on the impact of adopting MARVIN in such architecture.
3. GMAC: included in our analysis because it is part of a NIST recommendation (NIST, 2007b); in this manner, it seems useful to evaluate its suitability on constrained platforms, where space for large LUTs is hardly available.
4. PMAC1: chosen due to the interest on evaluating another fully parallelizable algorithm; moreover, this is the algorithm used for processing plaintext data in OCB (KROVETZ; ROGAWAY, 2005), which is also evaluated in this document and is part of the Minisec (LUK et al., 2007) architecture.

AEAD Schemes:

1. LETTERSOUP: our AEAD proposal.
2. EAX: as it should be simple to implement EAX from the CMAC algorithm, this could be an interesting AEAD scheme for use in TinySec, in the same manner that LETTERSOUP would be interesting if one replaces CMAC by MARVIN in the same architecture.
3. GCM: its evaluation is interesting for the same reasons that led to the choice of GMAC.

4. OCB: this is the algorithm adopted in MiniSec (LUK et al., 2007) and, thus, its efficiency evaluation should provide some insight on the impact of replacing this algorithm by LETTERSOUP in such architecture.

All algorithms used in the TelosB platform were developed from scratch, using the C language. The compiler used is the GNU msp-gcc[2] using the "-Os" optimization option, which resulted in a more compact and also efficient code.

We tested some variations of coding strategy, trying to identify optimized constructions for constrained platforms. As a result, we obtained two versions for each algorithm, one speed- and another memory-optimized. Basically, we vary the number of inline functions, the amount of look-up tables in the underlying block cipher, and the amount of pre-computed constants in the MAC and AEAD algorithms: each algorithm uses one key-dependent constant that can be computed at its initialization, except EAX, which has three (one for each tweaked CMAC); these constants are pre-computed in both implementation strategies, but less expensive constants (namely, values that can be computed by shifting other constants, which is common in PMAC1, CMAC and EAX) are pre-computed only in the speed-optimized versions. This methodology was adopted aiming to provide a fair comparison through similar optimizations, interfaces and coding style.

We tested the same solutions in our PC platform, but in this case they were written in Java and compiled using the Netbeans[3] IDE 6.5 with the Java Development Kit (JDK) 1.6. We included optimizations related to the abundant memory availability (enough to support the deployment of as many pre-computed tables as needed) and higher word length (allowing the algorithms to operate over groups of bytes instead of byte-wisely). Despite the availability of multiple cores in the PC where the benchmarks where run, we do not explore parallelizability in our implementations because some

[2]http://mspgcc.sourceforge.net/
[3]http://www.netbeans.org/

of the tested algorithms cannot be parallelized; with a parallelized implementation, though, we would expect even higher throughputs for MARVIN and LETTERSOUP than those reported in this document.

The implementations adopted for our PC platform are those provided by the JAES library (BARRETO, 2003), which include only CMAC, EAX, and GCM, though the GMAC algorithm can be easily derived from the GCM code provided. To this set of algorithms, we added MARVIN and LETTERSOUP, developed according to their specification; again, we tried to keep similar optimizations, interfaces and coding style for all the algorithms aiming for a fair comparison.

The underlying block cipher for all algorithms is the CURUPIRA-2 (SIMPLICIO et al., 2008) with 96-bit keys. This cipher has been chosen mainly due to its high performance and fairly reduced memory footprint in constrained platforms (SIMPLICIO et al., 2008, Section 5.2), to the public availability of highly optimized code for both constrained and powerful platforms (SIMPLICIO, 2009a; SIMPLICIO, 2009b), and to easiness of implementing SCTs with algorithms from the SQUARE family (the function for computing SCTs was derived directly from the code already available). In order to save space, the cipher's decryption function was removed from all implementations where it was not needed, which corresponds to all MAC and AEAD algorithms considered here, except OCB. Additionally, we pre-compute the cipher subkeys during initialization only in our PC-oriented implementation, while they are computed on-the-fly in our sensor-oriented code.

As a last remark, we note that some modifications in the codes from the JAES library were necessary for providing compatibility with the CURUPIRA-2 block cipher (originally, the algorithms from this library did not support 96-bit ciphers).

6.2.2.1 On the Implementations of GMAC and GCM

The specifications of GCM and GMAC do not cover 96-bit block ciphers, which obliged us to adapt the 128-bit version for allowing this block size. This adaptation has a reduced impact of the algorithm's efficiency, since it concerns basically the field polynomial used for multiplications and the encoding of the message and header lengths at the end of the authentication process.

Another issue refer to use of key-dependent LUTs in these algorithms: even the slightest level of optimization suggested in (MCGREW; VIEGA, 2005) with LUTs would require 256 bytes of RAM[4]. Since the same amount of memory could easily be used to accelerate the other algorithms considered here (e.g., by pre-computing the underlying cipher's round keys), we decided not to use LUTs in our GCM and GMAC implementations as a way to keep the fairness of the comparison.

6.2.3 Metrics

For our constrained platforms, the metrics considered in this benchmark are code size, RAM utilization, execution time and power consumption of the given algorithms. Conversely, due to the abundant availability of memory and energy in modern devices, in our PC we consider only the execution time of the different solutions.

For the TelosB mote, the amount of memory used by each algorithm was provided by the compiler itself, facilitating its measurement.

In all platforms, the execution time is obtained using the system's clock function, called just before a given operation starts and right after it finishes. In the following sections, the total time and energy consumption shown in our graphs correspond to the mean of several executions of each algorithm in the given platform.

[4] Key-dependent LUTs can be implemented using code memory instead of RAM only if the key never changes (which is usually not the case in robust security architectures).

In order to obtain an accurate measurement of the energy consumption, we performed direct measurements on the motes. We use an Agilent E3631A power supply (AGILENT, 2004) configured to provide 3.0 V for the TelosB mote. An Agilent 34401A digital multimeter (AGILENT, 2007) is then used to measure the system current flow as the different algorithms are executed. The data measured in this manner is sent to a computer through a GPIB cable, and displayed in the LabView[5] interface. Our measurement setup is shown in Figure 15.

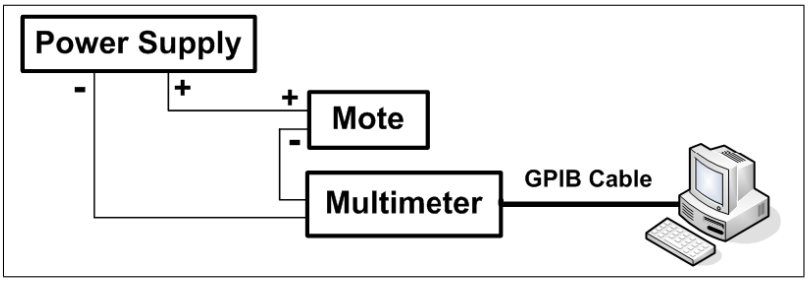

Figure 15: Energy measurements setup.

We configured the multimeter to provide a reading rate of 60 Hz, which allows us to measure both steady-state currents and transients. After we eliminated the influence of the current drained when the system is idle, I_{idle}, the energy consumption E was obtained as the time integration of the total current drained I multiplied by the (constant) voltage V used to power up the mote — i.e., $E = V \times \int_t (I - I_{idle})dt$.

6.3 Results for Constrained Platforms

We start our evaluation with the code size and the amount of RAM used by each of the algorithms, which is shown in Table 10.

The results obtained show that MARVIN is indeed a very compact MAC algorithm: both in its speed- and memory-optimized versions, our proposal requires reasonably less code memory for its implementation than the other solutions considered here, while maintaining a low level of RAM usage (though slightly higher than the other

[5]http://www.ni.com/labview/

algorithms). A similar result is also observed for LetterSoup, which is considerably more compact than EAX, GCM and, especially, OCB (remember that the latter's code includes the implementation Curupira-2's decryption algorithm). Note also that EAX in its memory-optimized version requires considerably less RAM memory than in its speed-optimized version, which can be explained by the larger number of pre-computed constants used in the latter case.

Table 10: Memory occupation (bytes) of MAC and AEAD algorithms on TelosB.

	Speed Optimized		Memory Optimized	
Algorithm	Code	RAM	Code	RAM
Marvin	2784	148	2416	148
CMAC	3340	144	2600	132
GMAC	3240	140	2508	140
PMAC1	3222	132	2620	132
LetterSoup	3682	218	3260	218
EAX	4528	252	3778	228
GCM	3862	220	3260	220
OCB	5120	216	4224	216

The amount of time and energy required for initializing the algorithms is displayed in Table 11. Note that the speed-optimized algorithms initialize faster than their memory-optimized counterparts, which means that the faster cipher used in the former versions compensates for the larger number of constants computed in the latter.

Table 11: Time and energy required for the algorithms' initialization on TelosB

	Speed Optimized		Memory Optimized	
Algorithm	Time (ms)	Energy (μJ)	Time (ms)	Energy (μJ)
Marvin	1.54	8.77	1.62	9.35
CMAC	1.21	6.78	1.50	8.60
GMAC	1.14	6.28	1.46	8.48
PMAC1	1.36	7.60	1.69	9.68
LetterSoup	1.54	9.01	1.63	9.16
EAX	2.29	12.96	1.52	8.93
GCM	1.12	6.60	1.47	8.64
OCB	2.08	12.01	2.15	12.37

This table shows that Marvin's initialization is slightly more resource-consuming than its counterparts in most cases; LetterSoup's initialization, on the other hand, is

faster than OCB's and EAX's, but not than GCM's. Also according to this table, the mean power consumption for these tasks fluctuates between 5.6 and 5.9 μW. Since the algorithms' initialization need to be performed only once for each key, this small difference between them should not be too significant in scenarios where the authentication key is not changed often (e.g., in WSNs where each sensor communicate with a same neighbor most of times).

Figure 16 shows the time required for authenticating messages of different sizes using the MAC algorithms in their speed-optimized versions, while Figure 17 does the same for the memory-optimized versions.

As expected, GMAC displays a poor performance, much worse than all other algorithms considered in our analysis due to the complexity of the operations performed by this algorithm. The performance of MARVIN surpasses that of CMAC and PMAC1 for messages larger than 12 bytes (i.e., larger than 1 block) in their memory-optimized versions, and for messages larger than 24 bytes (i.e., larger than 2 blocks) in their speed-optimized versions.

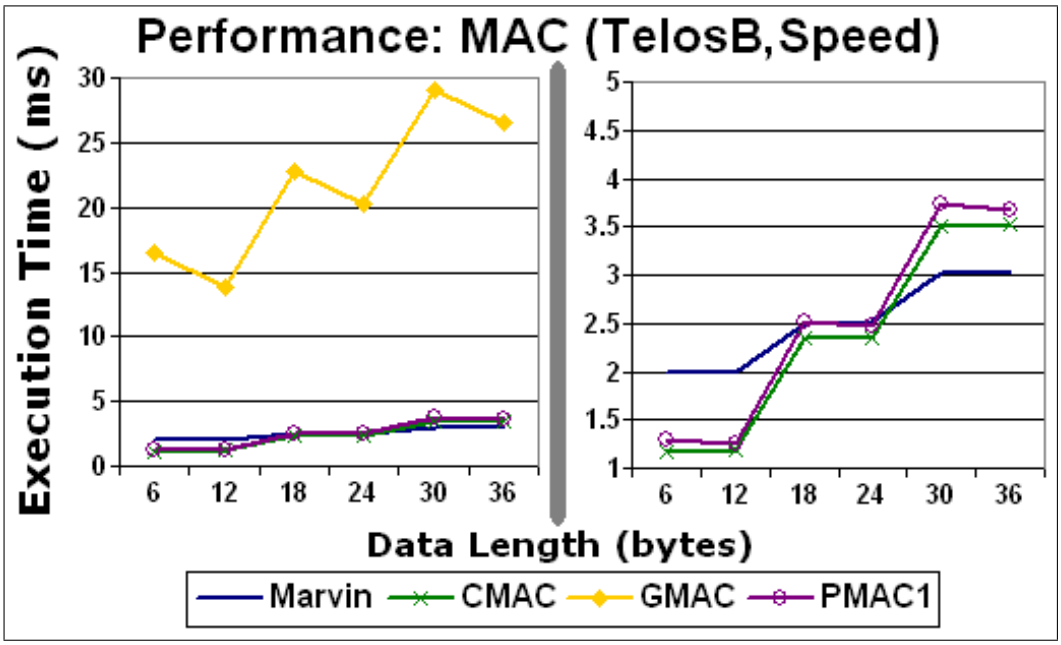

Figure 16: Performance of speed-optimized MAC algorithms on TelosB. The left side shows all algorithms, and the right side details the fastest ones.

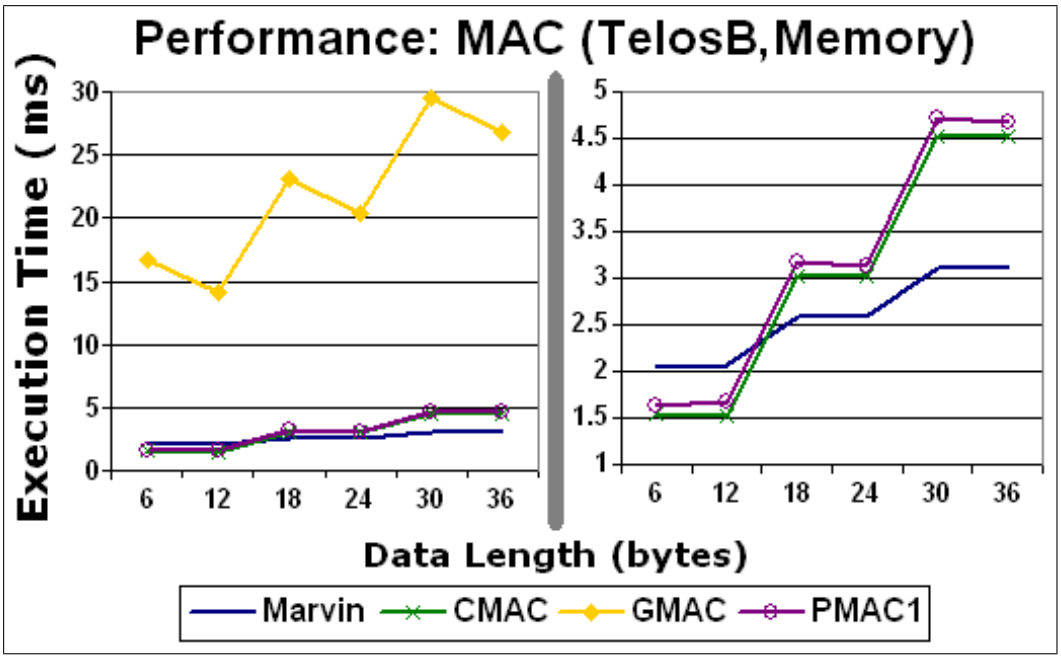

Figure 17: Performance of memory-optimized MAC algorithms on TelosB. The left side shows all algorithms, and the right side details the fastest ones.

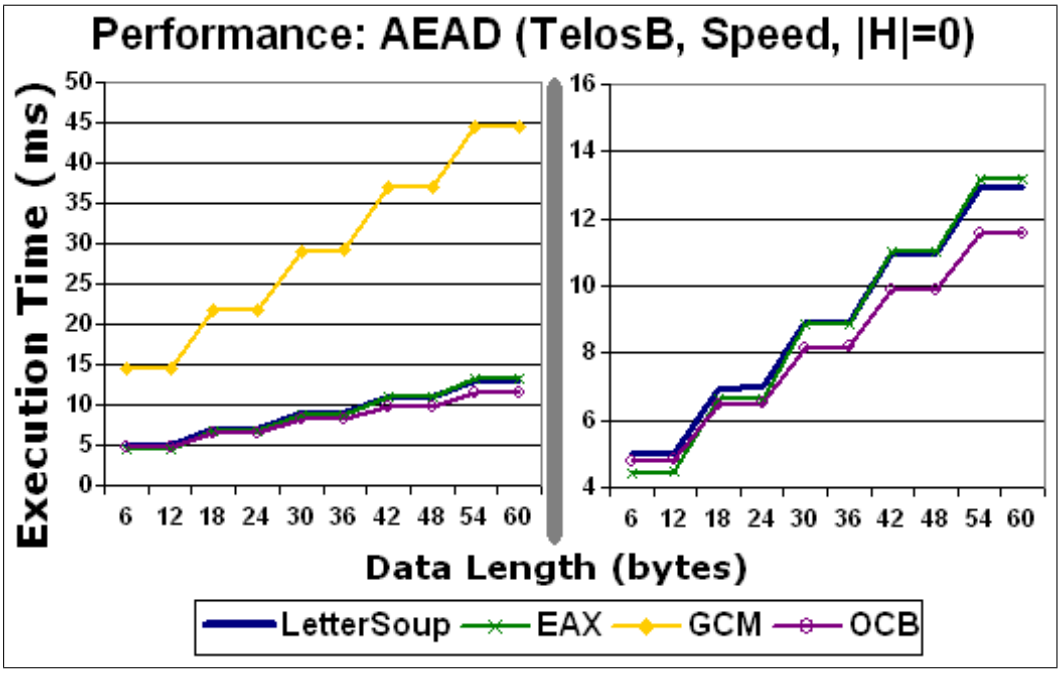

Figure 18: Performance of speed-optimized AEAD algorithms on TelosB, for $|H| = 0$. The left side shows all algorithms, while the right side details the fastest ones.

This result is in consonance with the theoretical analysis from section 6.1: MARVIN performs 1.4 and 1.8 encryptions for 1- and 2-block messages, respectively, while PMAC1 and CMAC perform 1 and 2 encryptions in such scenarios; the difference between the theoretical and practical results are due to the algorithms' ancillary processing, which is low when compared to a full encryption, but not completely negligible.

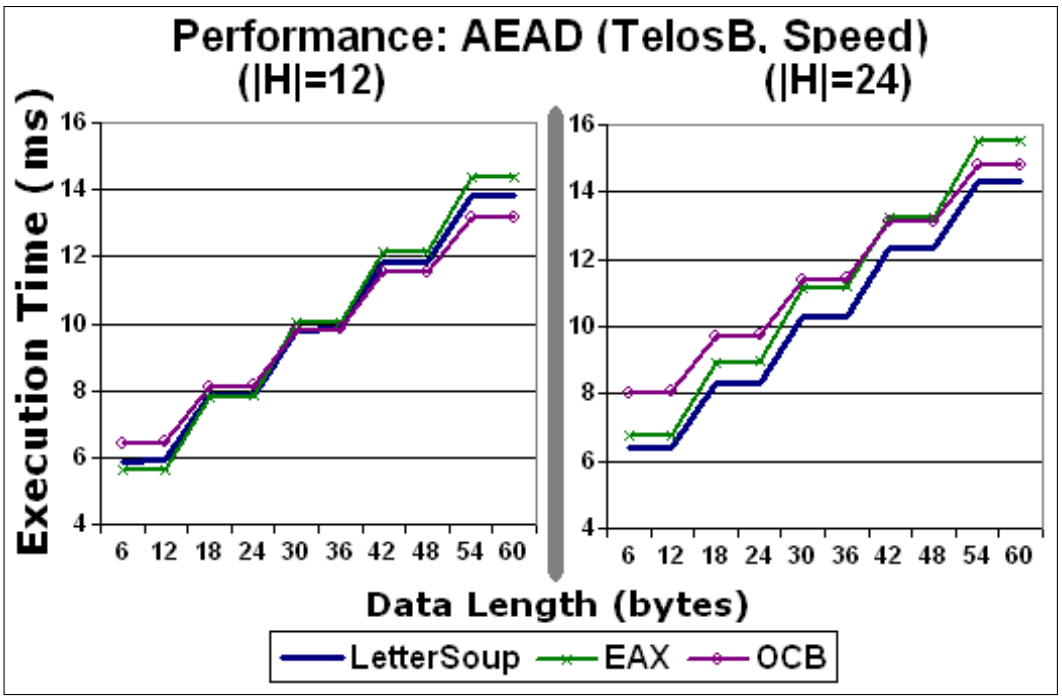

Figure 19: Performance of speed-optimized AEAD algorithms on TelosB, for $|H| = 12$ (left side) and $|H| = 24$ (right side).

The performance of the AEAD algorithms in their speed-optimized versions is shown in Figures 18 and 19, while Figures 20 and 21 do the same for their memory-optimized versions. We show only the encryption process, but the results for decryption are very similar.

Once again, the Carter-Wegman structure, represented by GCM, displays a poor performance in all scenarios. Comparing the remaining algorithms, we can see that OCB is indeed a very fast AEAD scheme for processing messages having no associated data: in this case, OCB is only surpassed by EAX in its speed-optimized version, and only when the message is shorter than 1 block. In comparison, LETTERSOUP is a more interesting solution in situations involving the presence of associated data

and typical message sizes (i.e., shorter than 60 bytes (CORDEIRO; AGRAWAL, 2006)): in its speed-optimized version, our AEAD proposal is the middle-way between OCB and EAX for message whose header has up to 1 block, and is the fastest solution for larger headers; in its memory-optimized version, LETTERSOUP's performance is similar to OCB's for 1-block or smaller headers, and is again the fastest solution for larger amounts of associated data.

Finally, the graphics for the algorithms' energy consumption are very similar in shape to the ones depicting their performances, showing that the energy efficiency of the MAC and AEAD solutions is directly related to their processing speed. Analogous to the result presented in Table 11, the mean power consumption for each algorithm's operation usually varies between 5.6 and 5.9 μW.

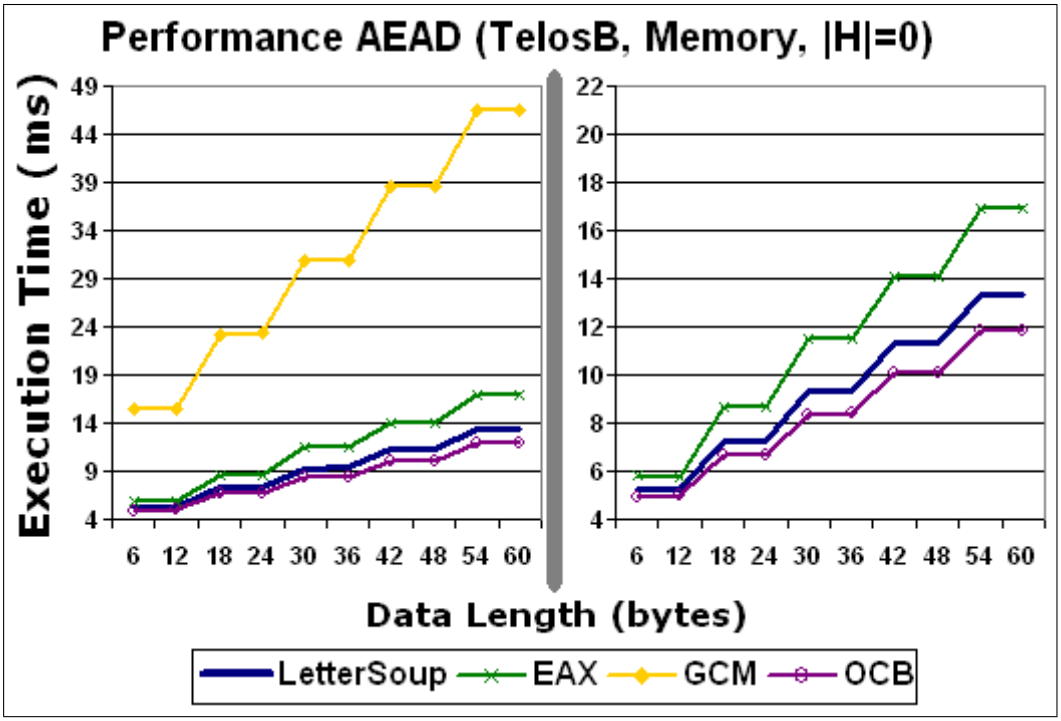

Figure 20: Performance of memory-optimized AEAD algorithms on TelosB, for $|H| = 0$. The left side shows all algorithms, while the right side details the fastest ones.

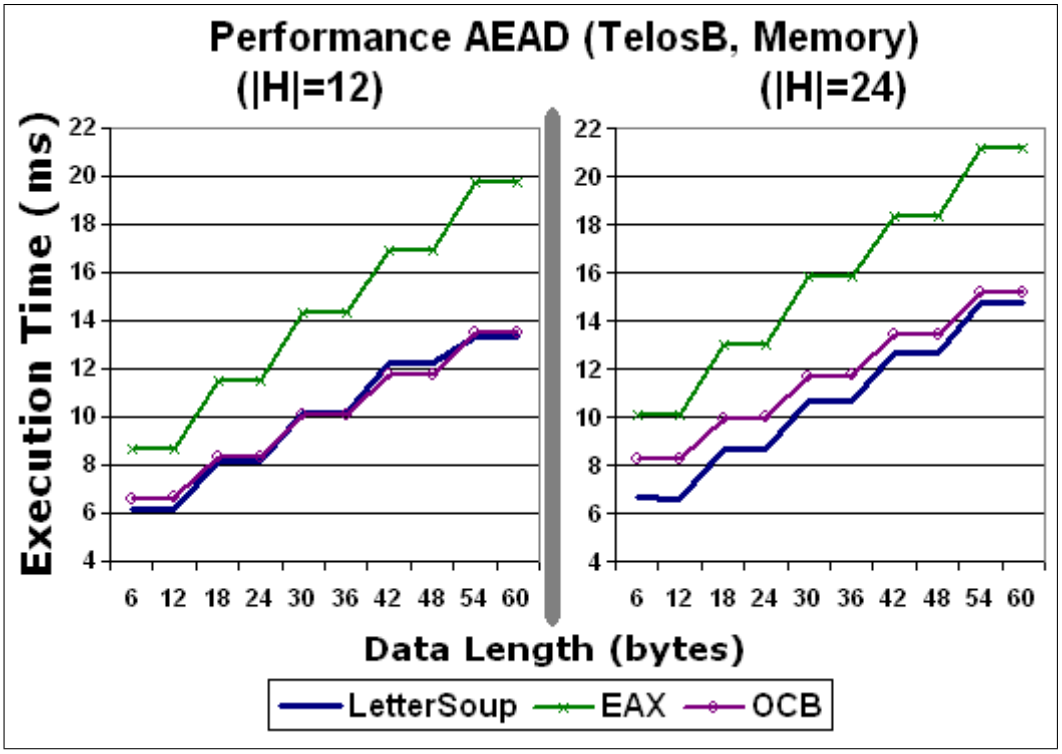

Figure 21: Performance of memory-optimized AEAD algorithms on TelosB, for $|H| = 12$ (left side) and $|H| = 24$ (right side).

6.4 Results for Powerful Platforms

The performance of the different MAC implementations in our PC testbed is displayed in Figure 22. Additionally, in order to facilitate the visualization and comparison between the algorithms, we provide a more detailed version of the same performance data in Figure 23.

This figure shows that MARVIN outperforms CMAC, PMAC and even GMAC with the increase in the message size. More specifically, as detailed in Figure 23: GMAC is faster than MARVIN only for messages shorter than ≈72 bytes (6 blocks); PMAC and CMAC, on the other hand, have a performance superior to MARVIN's only for messages shorter than ≈36 bytes (3 blocks) — note that, for this message size, the former require 3 full encryptions against ≈1.8 encryption performed by MARVIN. The result is that our MARVIN implementation achieves a throughput of approximately 500 Mbps in our platform, while GMAC, CMAC and PMAC achieve, respectively, about 375 Mbps,

260 Mbps and 235 Mbps.

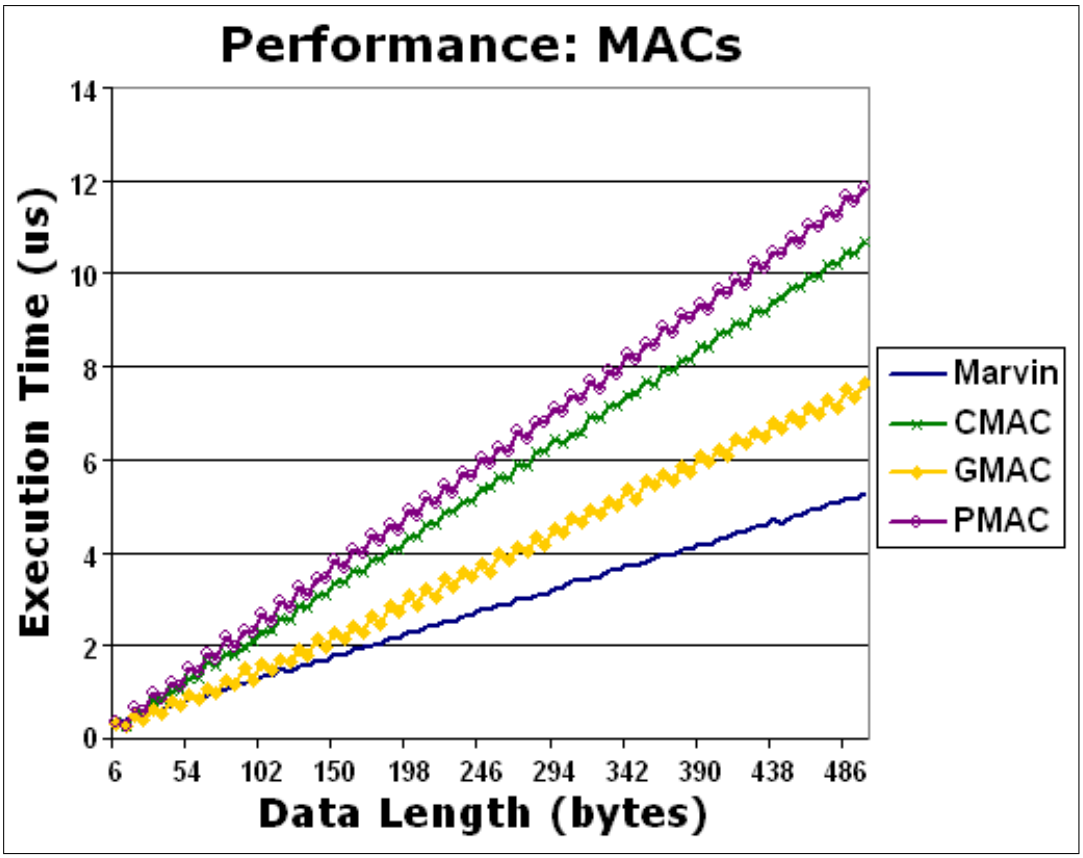

Figure 22: Performance of MAC algorithms on PC

Moreover, the initialization time of each algorithm is considerably low: about 450 ns for GMAC (mostly due to the construction of the key-dependent LUTs) and 50 ns for MARVIN, CMAC and PMAC (whose initialization involves much simpler operations than those performed in GMAC). Therefore, unless the application uses only very short messages, the adoption of MARVIN is advantageous from the point of view of performance.

The time required for encrypting and authenticating in our PC platform, using different AEAD solutions, is shown in Figure 24. The corresponding graph for the decryption process is very similar to this one and is thus omitted.

According to this figure, LETTERSOUP is faster than EAX and GCM, but not faster than OCB. More specifically, as detailed in Figure 24, EAX is slower than the other

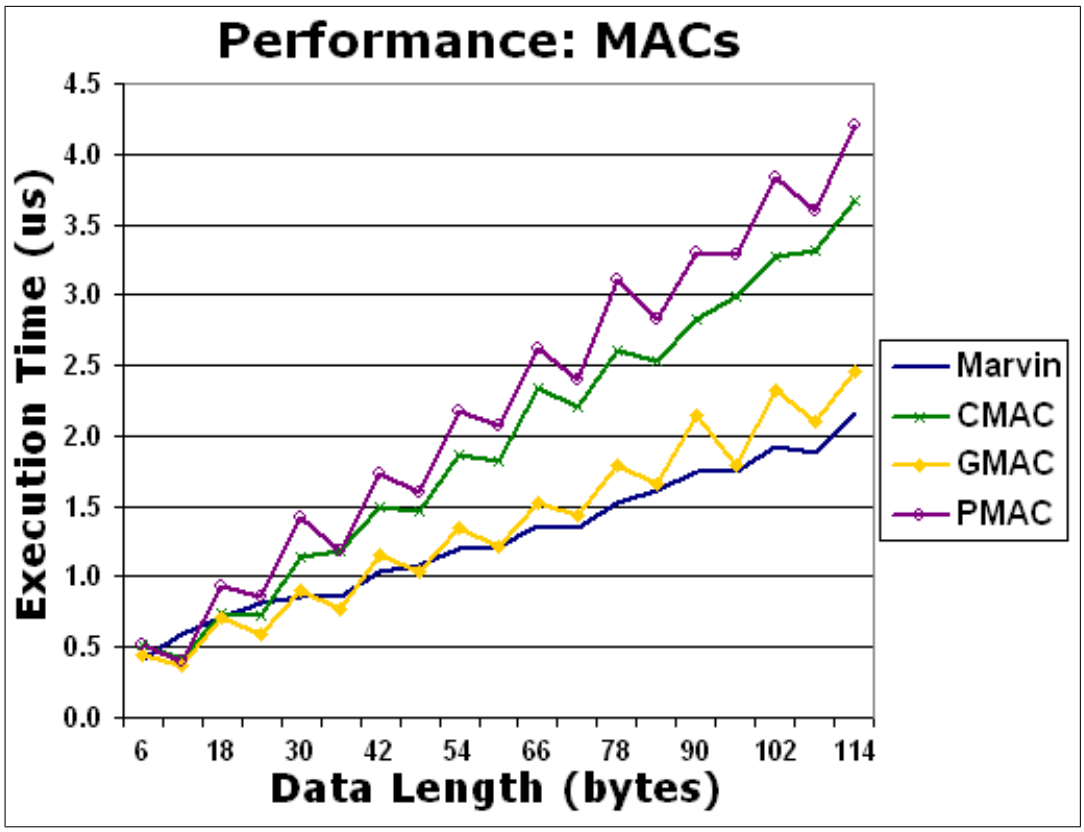

Figure 23: Performance of MAC algorithms on PC (detail)

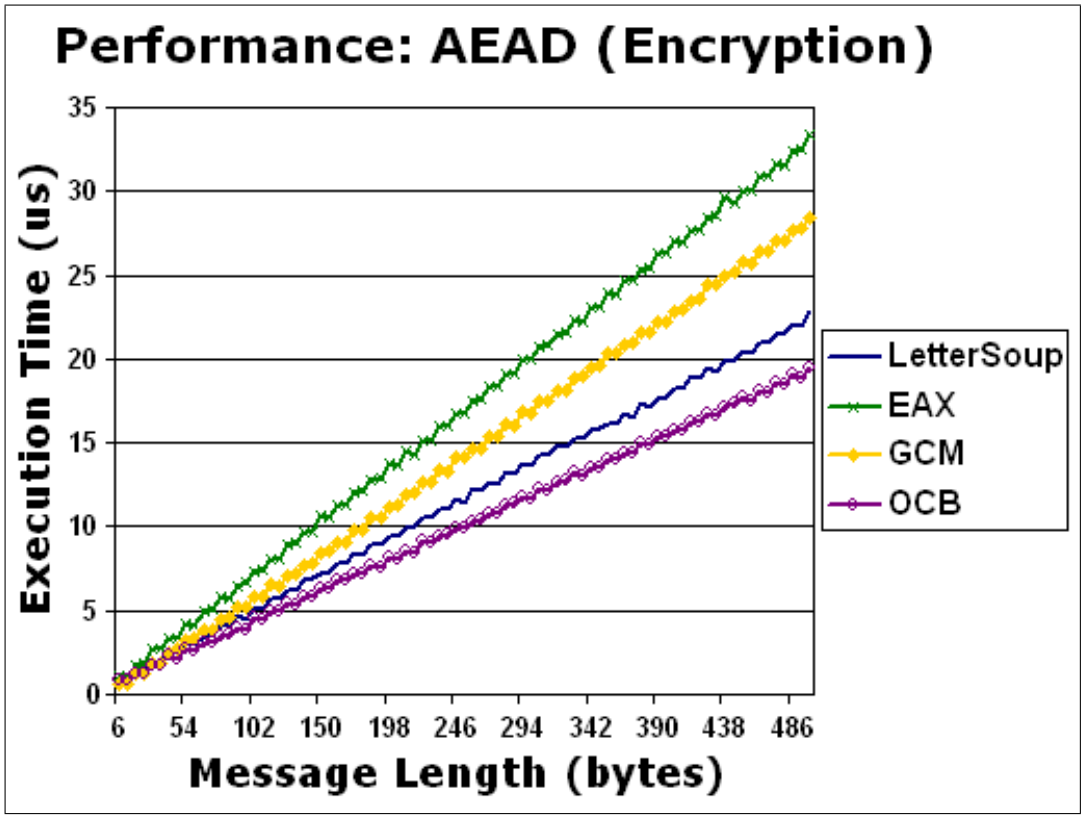

Figure 24: Performance of AEAD algorithms on PC: encryption

algorithms for all message sizes; GCM is the fastest algorithm for authenticating messages having less than 24 bytes (2 blocks), and is faster than LetterSoup for messages smaller than 48 bytes (4 blocks); finally, for messages having more than 48 bytes, OCB is the scheme that displays the best performance. In this scenario, the approximate throughputs for our implementations of LetterSoup, EAX, GCM, and OCB are, respectively, 170, 120, 140, and 200 Mbps.

This result was expected, since OCB is a one-pass scheme and requires a single encryption per block of the encrypted+authenticated message, while LetterSoup requires ≈1.4 encryption per block. Hence, in scenarios where all data to be authenticated must also be encrypted (i.e., when there is no associated data to be processed) and that can afford the adoption of patented algorithms, OCB seems to be a promising solution; in applications that require a very compact and unpatented scheme, though, the deployment of LetterSoup would be more recommended.

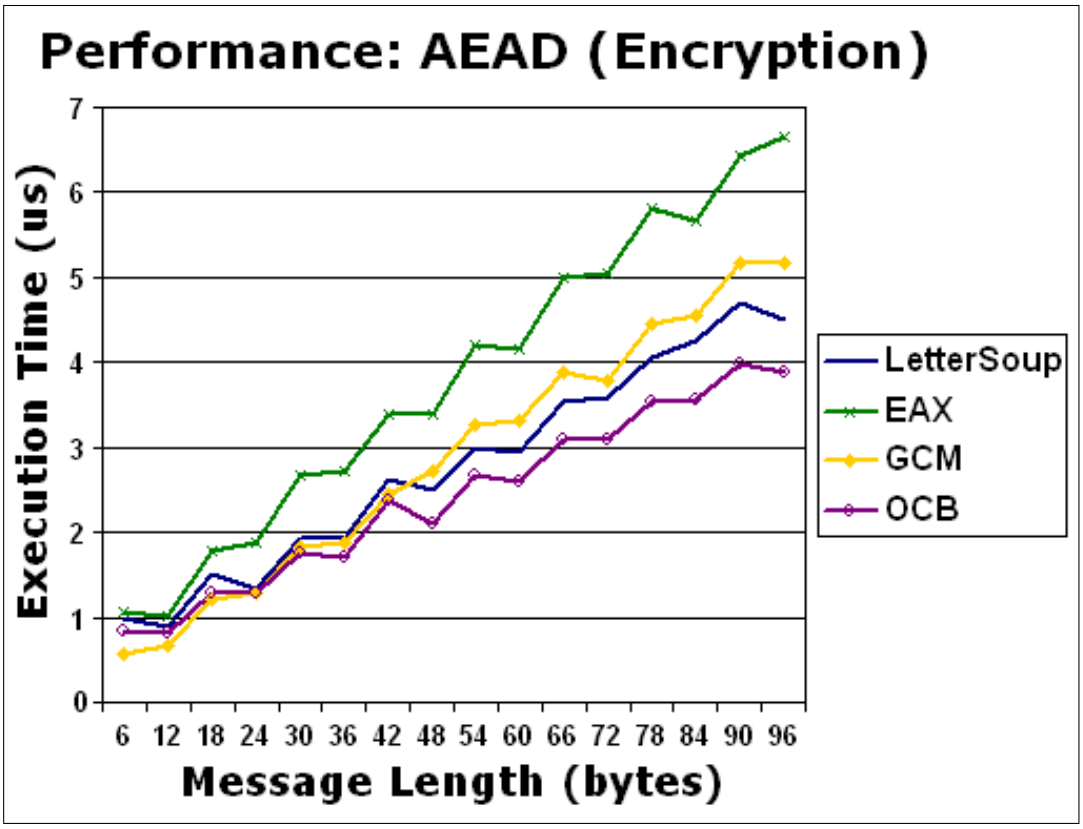

Figure 25: Performance of AEAD algorithms on PC: encryption (detail)

On the other hand, OCB requires a full encryption per block of associated data and LetterSoup requires ≈0.4 encryption for processing the same amount of data. In consequence, in scenarios where part of the message shall not be encrypted, the adoption of LetterSoup may also be advantageous in terms of performance.

As shown in Figure 26 and in more details in Figure 27, in which we consider messages where the confidential message's and the header's lengths are identical, our implementations of LetterSoup, EAX, GCM, and OCB achieve a throughput of, respectively, 260, 165, 205, and 215 Mbps. This may not be a very realistic scenario, but it is interesting to give an insight on how the header's processing impacts on the performance of OCB and LetterSoup.

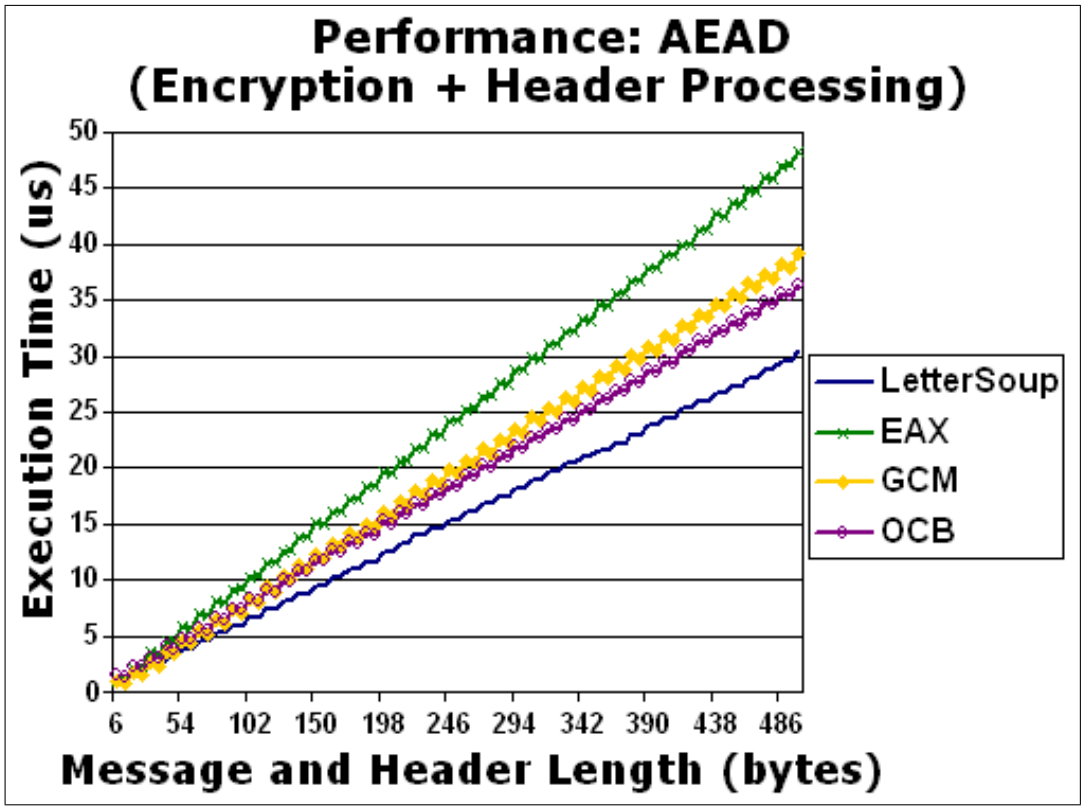

Figure 26: Performance of AEAD Algorithms on PC: encryption + associated data

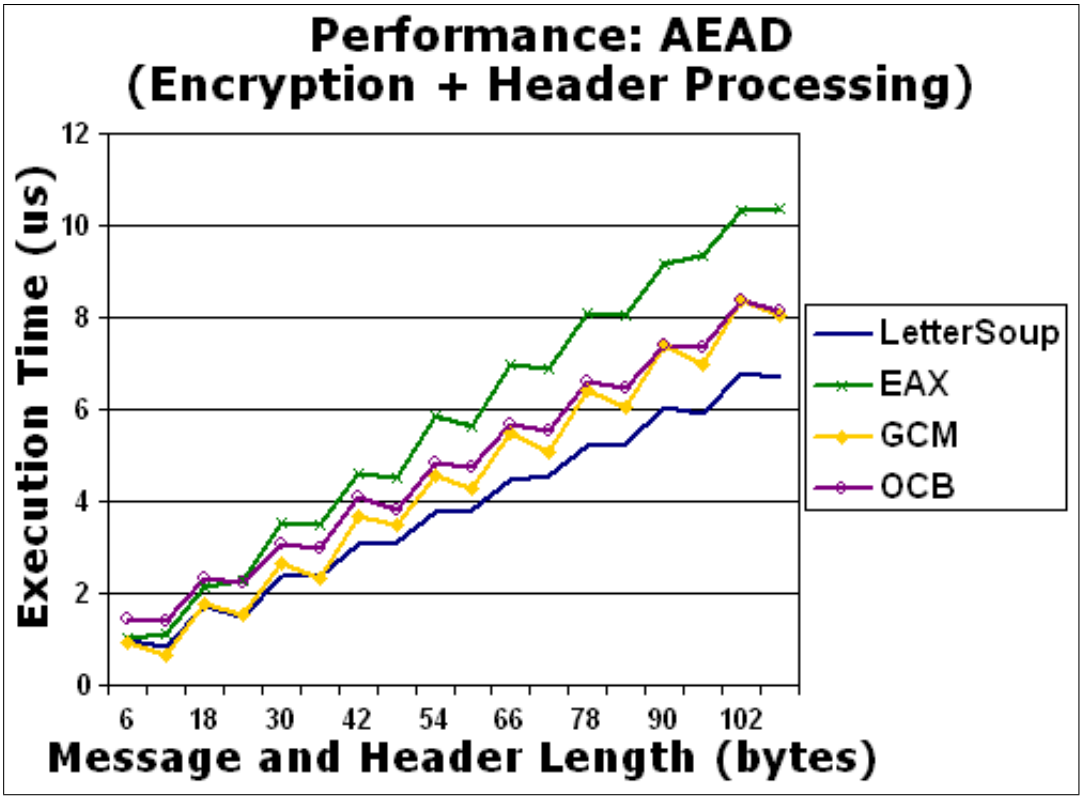

Figure 27: Performance of AEAD algorithms on PC: encryption + associated data (Detail)

6.5 Summary and Discussion

In this chapter, we evaluated the efficiency of MARVIN and LETTERSOUP both in powerful and constrained platforms.

Our theoretical and practical analyses show that MARVIN is a prominent MAC function for use in constrained platforms: in comparison with similar solutions, our proposal is very compact and displays a good performance for messages larger than 2 blocks. For very short messages (e.g., 1 byte), MARVIN would be a less appealing solution. However, there is a non-negligible overhead associated to the switching between transmit, receive and sleep states; hence, the transmission of such tiny messages (as opposed to consolidated, larger messages) in WSNs is usually discouraged (TIWARI; BALLAL; LEWIS, 2007; RUZZELLI, 2008). Furthermore, note that for ciphers whose block size is 12 bytes or lower, MARVIN would be the most efficient MAC solution —

among the ones considered in our benchmark — for authenticating a typical packet, whose length is expected to range from 30 to 60 bytes (CORDEIRO; AGRAWAL, 2006).

Our tests also show the potential of deploying LetterSoup on constrained platforms: although OCB is the most efficient solution for processing packets having no associated data (or, equivalently, when the header is fixed and can thus be precomputed), LetterSoup may be an advantageous alternative for scenarios requiring, for example, 1-block or larger headers to be processed for all packets, or that cannot afford the use of patented solutions.

Finally, the benchmark results for our powerful platform show that both Marvin and LetterSoup can take advantage of additional resources when they are available: on the one hand, Marvin surpasses the Carter-Wegman-based GMAC, known to provide a high throughput in powerful platforms; on the other hand, LetterSoup surpasses both EAX and GCM, being slower only than the (patented) OCB algorithm. Moreover, the proposed algorithms are both parallelizable, something that gains increasing importance with the continuous development of multi-core processors.

7 CONCLUSIONS

Message authentication is an important concern in any network: without this, unauthorized users could easily introduce invalid data into the system. This service is usually provided through the deployment of a secure Message Authentication Code (MAC), which simply authenticates (plain and/or encrypted) data; another possible approach is to deploy an Authenticated-Encryption with Associated Data (AEAD) scheme, which combines encryption and authentication in a single structure.

Due to the extremely constrained nature of the devices used in Wireless Sensor Networks (WSNs), most of the all-purpose cryptographic algorithms used in modern networks are not well-adapted to this particular context, which motivates the research for more efficient solutions.

In this document, in Chapter 3, we studied some modern design techniques for building MACs and AEADs, aiming to design novel cryptographic algorithms for this specific need. Among the possible choices, we selected the ALRED structure for two main reasons: first, it allows the reuse of an underlying block cipher machinery, thus introducing little impact in terms of memory occupation; second, while many similar designs require a full encryption per authenticated block, the ALRED structure requires only a Square-Complete Transform (SCT) per block, which corresponds roughly to 25%-40% of a block cipher call per block — the exact number depends on the details of the underlying cipher. Hence, algorithms developed according to this strategy can be made very compact and efficient even in highly constrained platforms.

Using the ALRED design, we proposed two novel solutions for message authentication: the MARVIN Message Authentication Code and its AEAD extension, LETTERSOUP; the latter combines MARVIN and the Linear Feedback Shift Register Counter (LFSRC) block cipher mode of operation. Both are described in Chapter 4. Despite the fact that the proposed algorithms have been tailored with constrained platforms in mind, they are flexible enough to also take advantage of the resources available on powerful platforms. Indeed, any optimization applicable to the underlying block cipher adopted can also be used to improve the performance of MARVIN and LETTERSOUP. Moreover, since both display a highly parallelizable structure, they can take further advantage of platforms where multiple processing cores are available.

Together with the specification of MARVIN and LETTERSOUP, we provided a detailed security analysis of both algorithms using the Game-Playing technique, thus bounding the amount of resources needed for attacking them. According to our results, both schemes are secure as long as (1) the underlying block cipher E adopted is a good pseudo-random permutation, and (2) the SCT created from E has a low maximum differential probability. The main limitation of the proposed solutions resides, thus, on the fact that the total number of messages that can be authenticated using MARVIN or LETTERSOUP and an n-bit underlying block cipher is lower than what is normally achieved with conventional solutions, since an attacker can explore the structure of the SCTs when making forgery attempts. Nonetheless, the security analysis of practical SCTs, constructed with ciphers based on the Wide Trail Strategy (DAEMEN; RIJMEN, 2001), shows that the level of security provided remains reasonably high: with SCTs constructed from AES (a 128-bit block cipher), then the security level provided by our solutions is comparable to that of conventional schemes adopting a 114-bit block cipher; analogously, if AES is replaced by the WSN-oriented cipher CURUPIRA-2 (which adopts 96-bit blocks), then the same comparison could be made with a conventional scheme adopting a 71-bit block cipher. Indeed, as discussed along Chapter 5, the se-

curity levels achieved are in consonance with the requirements of WSNs, since the number of messages available for attackers is usually very limited: with CURUPIRA-2 as the underlying block cipher, an attacker using the whole bandwidth of a TelosB or similar mote to check the validity of forgery attempts would still need about 550 days in order to produce a valid tag with probability higher than 50%; with AES, this number would be even higher.

Our benchmark results presented in Chapter 6 show that MARVIN and LETTERSOUP are indeed compact solutions. More precisely, an implementation of MARVIN together with the CURUPIRA-2 as underlying cipher takes about 2.5 KiB, requiring 5% to 20% less code memory than CMAC, PMAC and GMAC in similar conditions; moreover, MARVIN's operation requires less than 150 bytes of RAM, which is just slightly higher (4 to 16 bytes) than the amount used by these algorithms. Analogously, LETTERSOUP with CURUPIRA-2 fits in 3.5 KiB and requires less than 220 bytes of RAM memory for its operation, being more compact (up to 25%) than EAX, OCB and GCM and using a similar or lower (up to 32 bytes) amount of RAM.

Chapter 6 also reveals that both proposals are faster (and, thus, requires less energy to run) than similar solutions in many important scenarios, including constrained and powerful platforms. Using a TelosB mote as constrained testbed for our tests, we observed that MARVIN is better suited for authenticating messages whose length is larger than one or two blocks (e.g., larger than 13 or 25 bytes for 96-bit block ciphers such as CURUPIRA-2), while CMAC and PMAC are more interesting only for scenarios where very tiny messages are transmitted, which is is usually discouraged in the context of WSNs (TIWARI; BALLAL; LEWIS, 2007; RUZZELLI, 2008). In a PC equipped with a 32-bit 2.4 GHz Intel Pentium Quad Core processor, MARVIN achieved a throughput of approximately 500 Mbps, overcoming the performance of CMAC and PMAC1 for messages larger than 3 blocks, and also that of a fully optimized implementation of GMAC (known to provide a high throughput in powerful platforms) for messages

larger than 6 blocks. Furthermore, our tests with speed- and memory-optimized versions of GCM, EAX, OCB and LetterSoup on TelosB for message smaller than 60 bytes (which can be considered the typical case for WSNs (CORDEIRO; AGRAWAL, 2006)) show that the solution proposed is a more advantageous solution for scenarios where part of each message must remain unencrypted: in most cases, LetterSoup displays a better performance than GCM and EAX for all messages sizes, but it surpasses OCB only when there is at least one or two (possibly incomplete) blocks of associated data to be authenticated. Finally, the benchmark of the same AEAD schemes on our PC reveals that LetterSoup can also take advantage of additional resources when they are available: LetterSoup can provide a throughput ranging from 170 to 260 Mbps, surpassing both EAX and GCM (known by its high throughput), and being slower than OCB only when the amount of authenticated data that requires encryption considerably surpasses the amount of associated (plain) data to be processed.

Taking into account all the above features provided by our solutions, we expect them to be interesting alternatives for authenticating messages in a large number of practical scenarios. Additionally, aiming to stimulate the analysis and adoption of the proposed algorithms, we decided not to fill patents covering their usage.

7.1 Future Work

Quite recently, we have observed the proposal of some hardware-oriented ciphers like Present (BOGDANOV et al., 2007), HIGHT (HONG et al., 2006), Grain (HELL; JOHANSSON; MEIER, 2007) and Trivium (DECANNIÈRE, 2006), to cite a few. However, there has been little work on hardware-oriented solutions for message authentication. Although we briefly explore some issues related to the hardware implementation of our algorithms, a deeper analysis of the subject could lead to a better insight on Marvin's and LetterSoup's performance in hardware when compared to similar-purpose solutions, and possibly to enhancements in their designs. Moreover, since both Marvin

and LetterSoup require the implementation of SCTs and those are likely not available in current hardware cipher implementations (e.g., the AES encryption module available in TelosB motes), this study could also involve optimized and secure hardware implementations of ciphers having SCT interfaces.

Another interesting subject of research involves techniques for raising the security of the Alred construction and of the proposed algorithms themselves. This may involve the substitution of some SCTs by full-encryptions in the algorithm's design, as proposed in (MINEMATSU; MATSUSHIMA, 2007), which is likely to lead to new security bounds at the cost of performance. Another related task is related to the existence of replay attacks against MAC algorithms, which also applies to Marvin. Such attacks could be thwarted through the replacement of c = `0x42` by an input parameter (e.g., a sequence number), but this approach could also lead to free-start or related vulnerabilities; thus, it would be useful to assess the security of such tweaked version.

Finally, there are still many security-related areas in WSNs other than authentication. One of the most challenging involves the distribution of symmetric keys to the sensor nodes. While key distribution in modern systems usually involves asymmetric cryptography, the most widely accepted strategy for WSNs is to pre-distribute some key material prior to the nodes' deployment (ÇAMTEPE; YENER, 2005; MERWE; DAWOUD; MCDONALD, 2007). The reason for this divergent approach is that the performance of traditional public-key algorithms — e.g., RSA (RIVEST; SHAMIR; ADELMAN, 1977) — in highly constrained devices usually is not satisfactory enough for their wide adoption. More recently, though, the deployment of Identity Based (ID-based) protocols based on Elliptic Curve Cryptography (ECC) (HANKERSON; MENEZES; VANSTONE, 2003) has gained considerable attention from the research community. Indeed, some recent results — e.g., (ARANHA et al., 2009) — show that this is a prominent approach for scenarios that require a more flexible and dynamic key management than what is provided by pre-distribution schemes. Nonetheless, to date, the suitability of

this and other approaches in WSNs remains an open issue.

7.2 Publications

The following publications, submissions and envisioned papers are a direct or indirect result of the research effort carried out during this thesis:

- Conference Papers: in (SIMPLICIO et al., Submitteda), we formally prove the security of MARVIN and use these results to evaluate the security of the ALRED construction itself; in (SIMPLICIO et al., Submittedb), we survey and analyze the performance of different MAC algorithms (including MARVIN) on a TelosB mote.

- Journal Articles: in (SIMPLICIO et al., 2009), we describe MARVIN and LETTERSOUP and provide a preliminary security and performance analysis; in (SIMPLICIO et al., In production), we expand the discussion of (SIMPLICIO et al., Submitteda), including the results of the security analysis of LETTERSOUP; in in (SIMPLICIO et al., Submittedb), we expand the analysis provided in (SIMPLICIO et al., Submittedb), including the survey and evaluation of different AEAD schemes (including LETTERSOUP) and considering not only TelosB but also MicaZ in our testbed; in (SIMPLICIO et al., Submitted), we survey many different key management schemes for WSNs, focusing on pre-distribution solutions.

- Demos: in (MARGI et al., 2009a), we show that the deployment of security algorithms on a WSN testbed is possible without causing significant impact on the performance of such applications.

- Book chapters: in (MARGI et al., 2009b), we provide an overview of the area of security in WSNs, covering attack models, routing-related vulnerabilities, secure location and data-aggregation, lightweight solutions for message authentication, encryption and key distribution, as well as implementation-related issues.

REFERENCES

ADAMS, D. *The Hitchhiker's Guide to the Galaxy*. [S.l.]: Completely Unexpected Productions Ltd, 1979.

AGILENT. *E363xA Series Programmable DC Power Supplies – Data Sheet*. [S.l.], 2004. http://www.home.agilent.com/.

____. *Agilent 34401A Multimeter – Uncompromising Performance for Benchtop and System Testing*. [S.l.], 2007. http://www.home.agilent.com/.

AKYILDIZ, I. et al. Wireless sensor networks: A survey. *Computer Networks*, Elsevier, New York, NY, USA, v. 38, n. 4, p. 393–422, March 2002.

ALEMDAR, A.; IBNKAHLA, M. Wireless sensor networks: Applications and challenges. *9th International Symposium on Signal Processing and Its Applications (ISSPA 2007)*, p. 1–6, February 2007.

ARAMPATZIS, T.; LYGEROS, J.; MANESIS, S. A survey of applications of wireless sensors and wireless sensor networks. In: *Proc. of the 2005 IEEE International Symposium on Intelligent Control - Mediterrean Conference on Control and Automation*. [S.l.: s.n.], 2005. p. 719–724.

ARANHA, D. et al. NanoPBC: Implementing cryptographic pairings on an 8-bit platform. In: *Conference on Hyperelliptic curves, discrete Logarithms, Encryption, etc. – CHiLE'09*. [S.l.: s.n.], 2009.

BARRETO, P. *The JAES Library*. 2003. http://larc.usp.br/~pbarreto.

BARRETO, P.; SIMPLICIO, M. Curupira, a block cipher for constrained platforms. In: *Anais do 25º Simpósio Brasileiro de Redes de Computadores e Sistemas Distribuídos - SBRC 2007*. [S.l.]: SBC, 2007. v. 1, p. 61–74. ISBN 85-766-9106-X. http://www.larc.usp.br/~mjunior/files/en/Curupira1-extended.pdf.

BARTHE, G.; GRÉGOIRE, B.; BÉGUELIN, S. Formal certification of code-based cryptographic proofs. In: *POPL'09: Proc. of the 36th annual ACM SIGPLAN-SIGACT Symposium on Principles of programming languages*. New York, NY, USA: ACM, 2009. p. 90–101. ISBN 978-1-60558-379-2.

BELLARE, M.; KILIAN, J.; ROGAWAY, P. The security of the cipher block chaining message authentication code. *Journal of Computer and System Sciences*, Academic Press, Inc., v. 61, n. 3, p. 362–399, 2000.

BELLARE, M.; KOHNO, T.; NAMPREMPRE, C. Breaking and provably repairing the SSH authenticated encryption scheme: A case study of the Encode-then-Encrypt-and-MAC paradigm. *ACM Trans. Inf. Syst. Secur.*, ACM, New York, NY, USA, v. 7, n. 2, p. 206–241, 2004. ISSN 1094-9224.

BELLARE, M.; MICCIANCIO, D. A new paradigm for collision-free hashing: Incrementality at reduced cost. In: *Advances in Cryptology – Eurocrypt'97*. Heidelberg, Germany: Springer, 1997. (Lecture Notes in Computer Science, v. 1233), p. 163–192.

BELLARE, M.; NAMPREMPRE, C. Authenticated encryption: Relations among notions and analysis of the generic composition paradigm. In: *ASIACRYPT'00: Proc. of the 6th International Conference on the Theory and Application of Cryptology and Information Security*. London, UK: Springer-Verlag, 2000. p. 531–545. ISBN 3-540-41404-5.

BELLARE, M.; ROGAWAY, P. *Code-Based Game-Playing Proofs and the Security of Triple Encryption*. 2004. Cryptology ePrint Archive, Report 2004/331. http://eprint.iacr.org/2004/331.

BELLARE, M.; ROGAWAY, P.; WAGNER, D. The EAX mode of operation: A two-pass authenticated-encryption scheme optimized for simplicity and efficiency. In: *Fast Software Encryption - FSE'04*. [S.l.: s.n.], 2004. p. 389–407. http://www.cs.ucdavis.edu/~rogaway/papers/eax.pdf.

BELLOVIN, S.; BLAZE, M. *Cryptographic modes of operation for the internet*. 2001. Second NIST Workshop on Modes of Operation. citeseer.ist.psu.edu/bellovin01cryptographic.html.

BIHAM, E.; BIRYUKOV, A.; SHAMIR, A. Cryptanalysis of Skipjack reduced to 31 rounds using impossible differentials. In: *Advances in Cryptology – Eurocrypt'99*. [S.l.]: Springer, 1999. (Lecture Notes in Computer Science, v. 1592), p. 55–64.

BIHAM, E.; SHAMIR, A. Differential cryptanalysis of DES-like cryptosystems. In: *Crypto '90: Proc. of the 10th Annual International Cryptology Conference on Advances in Cryptology*. London, UK: Springer-Verlag, 1991. p. 2–21. ISBN 3-540-54508-5.

BLACK, J. *Authenticated Encryption*. Berlin, Germany: Springer, 2005. http://www.cs.ucdavis.edu/~rogaway/classes/227/fall03/ae.pdf.

BLACK, J.; ROGAWAY, P. A block-cipher mode of operation for parallelizable message authentication. In: *Advances in Cryptology - EUROCRYPT'02*. [S.l.]: Springer-Verlag, 2002. (Lecture Notes in Computer Science), p. 384–397.

BOGDANOV, A. et al. PRESENT: An ultra-lightweight block cipher. In: *Cryptographic Hardware and Embedded Systems – CHES'2007*. Heidelberg, Germany: Springer, 2007. (Lecture Notes in Computer Science).

BRASSARD, G. On computationally secure authentication tags requiring short secret shared keys. In: *CRYPTO'82*. [S.l.: s.n.], 1982. p. 79–86. http://www.tcs.ics.saitama-u.ac.jp/~crypto_papers/1/html/PDF/C82/79.PDF.

ÇAMTEPE, S. A.; YENER, B. *Key distribution mechanisms for wireless sensor networks: a survey*. [S.l.], 2005.

CARTER, J.; WEGMAN, M. Universal classes of hash functions. *Journal of Computer and System Sciences*, v. 18, n. 2, p. 143–154, April 1979. ISSN 00220000. http://dx.doi.org/10.1016/0022-0000(79)90044-8.

CERPA, A.; ESTRIN, D. ASCENT: Adaptive self-configuring sensor networks topologies. In: *Proc. of the Twenty First International Annual Joint Conference of the IEEE Computer and Communications Societies (INFOCOM'02)*. [S.l.: s.n.], 2002.

CHUN, K. et al. Differential and linear cryptanalysis for 2-round SPNs. *Inf. Process. Lett.*, Elsevier North-Holland, Inc., Amsterdam, The Netherlands, The Netherlands, v. 87, n. 5, p. 277–282, 2003. ISSN 0020-0190.

CORDEIRO, C.; AGRAWAL, D. *Ad Hoc & Sensor Networks: Theory And Applications*. River Edge, NJ, USA: World Scientific Publishing Co., Inc., 2006. ISBN 9812566813.

CROSSBOW. *MICA2 Datasheet*. 2008. http://www.xbow.com/Products/Product_pdf_files/Wireless_pdf/MICA2_Datasheet.pdf.

____. *MICAz Datasheet*. 2008. http://www.xbow.com/Products/Product_pdf_files/Wireless_pdf/MICAz_Datasheet.pdf.

____. *TelosB Datasheet*. 2008. http://www.xbow.com/Products/Product_pdf_files/Wireless_pdf/TelosB_Datasheet.pdf.

DAEMEN, J.; KNUDSEN, L. R.; RIJMEN, V. The block cipher Square. In: *Fast Software Encryption – FSE'97*. Haifa, Israel: Springer, 1997. (Lecture Notes in Computer Science, v. 1267), p. 149–165.

DAEMEN, J. et al. The Noekeon block cipher. In: *First open NESSIE Workshop*. Leuven, Belgium: NESSIE Consortium, 2000.

DAEMEN, J.; RIJMEN, V. The block cipher BKSQ. In: *Smart Card Research and Applications – CARDIS'98*. [S.l.]: Springer, 1998. (Lecture Notes in Computer Science, v. 1820), p. 236–245.

____. The wide trail design strategy. *Lecture Notes in Computer Science*, v. 2260, p. 222–239, 2001. ISSN 0302-9743. http://link.springer-ny.com/link/service/series/0558/papers/2260/22600222.pdf.

____. *The Design of Rijndael: AES – The Advanced Encryption Standard*. Heidelberg, Germany: Springer, 2002. ISBN 3-540-42580-2.

____. A new MAC construction Alred and a specific instance Alpha-MAC. In: *FSE*. [S.l.: s.n.], 2005. p. 1–17. http://dx.doi.org/10.1007/11502760_1.

____. *The Pelican MAC Function*. 2005. Cryptology ePrint Archive, Report 2005/088. http://eprint.iacr.org/.

____. Understanding two-round differentials in AES. In: *SCN*. [S.l.]: Springer, 2006. (Lecture Notes in Computer Science, v. 4116), p. 78–94. ISBN 3-540-38080-9. http://dblp.uni-trier.de/db/conf/scn/scn2006.html#DaemenR06.

DECANNIÈRE, C. Trivium: a stream cipher construction inspired by block cipher design principles. *Information Security*, Springer, Berlin/Heidelberg, Germany, v. 4176, p. 36–55, 2006.

DIFFIE, W.; HELLMAN, M. E. New directions in cryptography. *IEEE Transactions on Information Theory*, IT-22, n. 6, p. 644–654, 1976. citeseer.ist.psu.edu/diffie76new.html.

ELTOWEISSY, M.; MOHARRUM, M.; MUKKAMALA, R. Dynamic key management in sensor networks. *IEEE Communications Magazine*, v. 44, n. 4, p. 122–130, April 2006. ISSN 0163-6804.

FLUHRER, S.; MANTIN, I.; SHAMIR, A. Weaknesses in the key scheduling algorithm of RC4. In: *SAC'01: Revised Papers from the 8th Annual International Workshop on Selected Areas in Cryptography*. London, UK: Springer-Verlag, 2001. p. 1–24. ISBN 3-540-43066-0.

GAMAL, T. E. A public key cryptosystem and a signature scheme based on discrete logarithms. In: *Proceedings of CRYPTO 84 on Advances in Cryptology*. New York, NY, USA: Springer-Verlag, 1985. p. 10–18. ISBN 0-387-15658-5.

GAY, D. et al. *The nesC 1.1 Language Reference Manual*. May 2003. http://nescc.sourceforge.net/papers/nesc-ref.pdf.

____. The nesC language: A holistic approach to networked embedded systems. In: *In Proc. of Programming Language Design and Implementation (PLDI)*. [S.l.: s.n.], 2003. p. 1–11.

GLIGOR, V.; DONESCU, P. Fast encryption and authentication: XCBC encryption and XECB authentication modes. In: *FSE'01: Revised Papers from the 8th International Workshop on Fast Software Encryption*. London, UK: Springer-Verlag, 2002. p. 92–108. ISBN 3-540-43869-6.

GOLDREICH, O.; GOLDWASSER, S.; MICALI, S. How to construct random functions. *Journal of the ACM (JACM)*, ACM, New York, NY, USA, v. 33, n. 4, p. 792–807, 1986. ISSN 0004-5411.

HANKERSON, D.; MENEZES, A.; VANSTONE, S. *Guide to Elliptic Curve Cryptography*. Secaucus, NJ, USA: Springer-Verlag New York, Inc., 2003. ISBN 038795273X.

HELL, M.; JOHANSSON, T.; MEIER, W. Grain: a stream cipher for constrained environments. *Int. J. Wire. Mob. Comput.*, Inderscience Publishers, Geneva, Switzerland, v. 2, n. 1, p. 86–93, 2007. ISSN 1741-1084.

HEYS, H. M. A tutorial on linear and differential cryptanalysis. *Cryptologia*, v. 26, n. 3, p. 189–221, 2002. ISSN 0161–1194. http://citeseer.ist.psu.edu/443539.html.

HILL, J. et al. System architecture directions for networked sensors. In: *Architectural Support for Programming Languages and Operating Systems*. [S.l.: s.n.], 2000. p. 93–104. citeseer.ist.psu.edu/hill00system.html.

HONG, D. et al. HIGHT: A new block cipher suitable for low-resource device. In: *CHES*. [S.l.: s.n.], 2006. p. 46–59.

HONG, S. et al. Provable security against differential and linear cryptanalysis for the SPN structure. In: *FSE'00: Proc. of the 7th International Workshop on Fast Software Encryption*. London, UK: Springer-Verlag, 2001. p. 273–283. ISBN 3-540-41728-1.

IEEE. *Standard Specifications for Public-Key Cryptography – IEEE Std 1363:2000*. [S.l.], 2000.

IEEE. *802.11i-2004 Amendment to IEEE standard 802.11*. [S.l.], 2004. standards.ieee.org/getieee802/download/802.11i-2004.pdf.

INTANAGONWIWAT, C.; GOVINDAN, R.; ESTRIN, D. Directed diffusion: A scalable and robust communication paradigm for sensor networks. In: ACM. *6th Annual International Conference on Mobile Computing and Networking, (MobiCom'00)*. [S.l.], 2000. p. 56–67.

IWATA, T.; KUROSAWA, K. OMAC: One-key CBC MAC. In: *Fast Software Encryption – FSE'2003*. Heidelberg, Germany: Springer, 2003. (Lecture Notes in Computer Science, v. 2887), p. 129–153.

JUTLA, C. Encryption modes with almost free message integrity. In: *EUROCRYPT'01: Proc. of the International Conference on the Theory and Application of Cryptographic Techniques*. London, UK: Springer-Verlag, 2001. p. 529–544. ISBN 3-540-42070-3.

KANG, J.-S. et al. Practical and provable security against differential and linear cryptanalysis for substitution-permutation networks. *ETRI Journal*, v. 23, p. 158–167, 2001.

KARENOS, K.; KALOGERAKI, V. Real-time traffic management in sensor networks. In: *RTSS'06: Proc. of the 27th IEEE International Real-Time Systems Symposium*. Washington, DC, USA: IEEE Computer Society, 2006. p. 422–434. ISBN 0-7695-2761-2.

KARLOF, C.; SASTRY, N.; WAGNER, D. Tinysec: a link layer security architecture for wireless sensor networks. In: *2nd International Conference on Embedded Networked Sensor Systems – SenSys'2004*. Baltimore, USA: ACM, 2004. p. 162–175. ISBN 1-58113-879-2.

KELIHER, L. Refined analysis of bounds related to linear and differential cryptanalysis for the AES. In: *AES Conference*. [S.l.: s.n.], 2004. p. 42–57.

KELIHER, L.; MEIJER, H.; TAVARES, S. *Dual of New Method for Upper Bounding the Maximum Average Linear Hull Probability for SPNs*. 2001. Cryptology ePrint Archive, Report 2001/033. http://eprint.iacr.org/.

____. New method for upper bounding the maximum average linear hull probability for SPNs. In: *EUROCRYPT'01: Proc. of the International Conference on the Theory and Application of Cryptographic Techniques*. London, UK: Springer-Verlag, 2001. p. 420–436. ISBN 3-540-42070-3.

KELIHER, L.; SUI, J. Exact maximum expected differential and linear probability for 2-round advanced encryption standard. *Information Security, IET*, v. 1, n. 2, p. 53–57, June 2007. ISSN 1751-8709. citeseer.ist.psu.edu/738374.html.

KELLY, T. The myth of the skytale. *Cryptologia*, v. 22, n. 3, p. 244–260, 1998.

KRAWCZYK, H. LFSR-based hashing and authentication. In: *Advances in Cryptology – Crypto'94*. Heidelberg, Germany: Springer, 1994. (Lecture Notes in Computer Science, v. 839), p. 129–139. ISBN 3-540-58333-5.

KROVETZ, T.; ROGAWAY, P. *Internet Draft: The OCB Authenticated-Encryption Algorithm*. March 2005. http://www.cs.ucdavis.edu/~rogaway/papers/ocb-id.htm.

KULIK, J.; HEINZELMAN, W.; BALAKRISHNAN, H. Negotiation-based protocols for disseminating information in wireless sensor networks. *Wireless Networks*, Kluwer Academic Publishers, Hingham, MA, USA, v. 8, n. 2/3, p. 169–185, 2002. ISSN 1022-0038.

LEURENT, G. Message freedom in MD4 and MD5 collisions: Application to APOP. In: *FSE*. [S.l.]: Springer, 2007. (LNCS, v. 4593), p. 309–328. ISBN 978-3-540-74617-1. http://dx.doi.org/10.1007/978-3-540-74619-5_20.

LEVIS, P. et al. TinyOS: An operating system for wireless sensor networks. In: ____. *Ambient Intelligence*. [S.l.]: Springer-Verlag, 2004.

LI, T. et al. *SenSec Design*. [S.l.], February 2005.

LUK, M. et al. Minisec: A secure sensor network communication architecture. In: *IPSN'07: Proc. of the 6th international conference on Information processing in sensor networks*. New York, NY, USA: ACM, 2007. p. 479–488. ISBN 978-1-59593-638-X.

MACWILLIAMS, F. J.; SLOANE, N. J. A. *The theory of error-correcting codes*. [S.l.]: North-Holland Mathematical Library, 1977.

MARGI, C. et al. *Demo: Security Mechanisms Impact and Feasibility on Wireless Sensor Networks Applications*. 2009. IEEE INFOCOM 2009.

____. Mini-curso: Segurança em redes de sensores sem fio. In: ____. [S.l.]: Sociedade Brasileira de Computação (SBC), 2009. p. 149–194.

MCGREW, D.; VIEGA, J. *Flexible and Efficient Message Authentication in Hardware and Software*. 2003. http://www.zork.org/gcm/gcm-paper.pdf.

____. *The Galois/Counter Mode of Operation (GCM)*. May 2005. Submission to NIST Modes of Operation Process. http://www.cryptobarn.com/papers/gcm-spec.pdf.

MENEZES, A. J.; OORSCHOT, P. C. van; VANSTONE, S. A. *Handbook of Applied Cryptography*. Boca Raton, USA: CRC Press, 1999.

MERWE, J. V. D.; DAWOUD, D.; MCDONALD, S. A survey on peer-to-peer key management for mobile ad hoc networks. *ACM Comput. Surv.*, ACM, New York, NY, USA, v. 39, n. 1, p. 1, 2007. ISSN 0360-0300.

MINEMATSU, K.; MATSUSHIMA, T. Improved MACs from differentially-uniform permutations. *IEICE Trans. Fundam. Electron. Commun. Comput. Sci.*, Oxford University Press, Oxford, UK, E90-A, n. 12, p. 2908–2915, 2007. ISSN 0916-8508.

NAKAHARA, J. Analysis of Curupira block cipher. In: *Anais do 8º Simpósio Brasileiro em Segurança da Informação e Sistemas Computacionais*. [S.l.: s.n.], 2008.

NANO-RK. *FireFly 2.2 Datasheet*. 2007. http://www.nanork.org/wiki/FireFly.

NIST. *Federal Information Processing Standard (FIPS PUB 46-3) – Data Encryption Standard (DES)*. [S.l.], January 1977. http://csrc.nist.gov/publications/fips/fips46-3/fips46-3.pdf.

NIST. *Federal Information Processing Standard (FIPS PUB 113) – Standard on Computer Data Authentication*. [S.l.], May 1985. http://www.itl.nist.gov/fipspubs/fip113.htm.

NIST. *Federal Information Processing Standard (FIPS 197) – Advanced Encryption Standard (AES)*. [S.l.], November 2001. http://csrc.nist.gov/publications/fips/fips197/fips-197.pdf.

____. *Special Publication SP 800-38A – Recommendations for Block Cipher Modes of Operation, Methods and Techniques*. [S.l.], December 2001. http://csrc.nist.gov/publications/nistpubs/800-38a/sp800-38a.pdf.

NIST. *Federal Information Processing Standard (FIPS PUB 198) – The Keyed-Hash Message Authentication Code*. [S.l.], March 2002. http://csrc.nist.gov/publications/fips/fips198/fips-198a.pdf.

____. *Special Publication 800-38C Recommendation for Block Cipher Modes of Operation: the CCM Mode for Authentication and Confidentiality*. [S.l.], May 2004. http://csrc.nist.gov/publications/PubsSPs.html.

____. *Special Publication 800-38B Recommendation for Block Cipher Modes of Operation: the CMAC Mode for Authentication*. [S.l.], May 2005. http://csrc.nist.gov/publications/PubsSPs.html.

_____. *Announcing request for candidate algorithm nominations for a new cryptographic hash algorithm (SHA-3) Family*. [S.l.], November 2007. v. 72, 62212–62220 p. http://csrc.nist.gov/groups/ST/hash/documents/FR_Notice_Nov07.pdf.

_____. *Special Publication 800-38D – Recommendation for Block Cipher Modes of Operation: Galois/Counter Mode (GCM) and GMAC*. [S.l.], November 2007. http://csrc.nist.gov/publications/nistpubs/800-38D/SP-800-38D.pdf.

_____. *Federal Information Processing Standard (FIPS 180-3) – Secure Hash Standard*. [S.l.], October 2008. http://csrc.nist.gov/publications/nistpubs/800-107/NIST-SP-800-107.pdf.

_____. *NIST Comments on Cryptanalytic Attacks on SHA-1*. January 2009. http://csrc.nist.gov/groups/ST/hash/statement.html.

NSA. *Skipjack and KEA Algorithm Specifications, version 2.0*. [S.l.], May 1998. http://csrc.nist.gov/CryptoToolkit/skipjack/skipjack.pdf.

PARK, S. et al. On the security of rijndael-like structures against differential and linear cryptanalysis. In: *ASIACRYPT'02: Proc. of the 8th International Conference on the Theory and Application of Cryptology and Information Security*. London, UK: Springer-Verlag, 2002. p. 176–191. ISBN 3-540-00171-9.

_____. Improving the upper bound on the maximum differential and the maximum linear hull probability for SPN structures and AES. In: *Fast Software Encryption – FSE'03*. [S.l.]: Springer-Verlag, 2003. (Lecture Notes in Computer Science, v. 2887), p. 247–260. www.iacr.org/archive/fse2003/28870263/28870263.pdf.

PARK, T.; SHIN, K. LiSP: A lightweight security protocol for wireless sensor networks. *Trans. on Embedded Computing Sys.*, v. 3, n. 3, p. 634–660, 2004. ISSN 1539-9087.

PATEL, M.; VENKATESAN, S.; WEINER, D. Role assignment for data aggregation in wireless sensor networks. In: *21st International Conference on Advanced Information Networking and Applications Workshops (AINAW'07)*. [S.l.: s.n.], 2007. v. 2, p. 390–395.

PERRIG, A. et al. SPINS: security protocols for sensor netowrks. In: *Proc. of the 7th Annual International Conference on Mobile Computing and Networking*. [S.l.]: ACM Press, 2001. p. 189–199. ISBN 1-58113-422-3.

PETRANK, E.; RACKOFF, C. CBC-MAC for real-time data sources. *Journal of Cryptology*, v. 13, n. 3, p. 315–338, 2000. http://link.springer.de/link/service/journals/00145/contents/00/10009/.

RIJMEN, V. et al. The cipher SHARK. In: *Fast Software Encryption – FSE'96*. [S.l.]: Springer, 1996. (Lecture Notes in Computer Science, v. 1039), p. 99–111.

RIVEST, R. The MD4 message digest algorithm. In: *CRYPTO*. [S.l.]: Springer, 1990. (LNCS, v. 537), p. 303–311. ISBN 3-540-54508-5. http://link.springer.de/link/service/series/0558/bibs/0537/05370303.htm.

____. *The MD5 message-digest algorithm - RFC 1320*. [S.l.], 1992. http://www.ietf.org/rfc/rfc1321.txt.

RIVEST, R. L.; SHAMIR, A.; ADELMAN, L. *A Method for obtaining Digital Signatures and Public-Key Cryptosystems*. [S.l.], 1977. 15 p. citeseer.ist.psu.edu/rivest78method.html.

ROGAWAY, P. *PMAC and OCB: Patent-assurance Letter*. 2001. http://www.cs.ucdavis.edu/~rogaway/ocb/ieee.pdf.

____. *The AEM Authenticated-Encryption Mode (Specification 1.3)*. 2003. http://www.cs.ucdavis.edu/~rogaway/papers/aem.ps.

____. Efficient instantiations of tweakable blockciphers and refinements to modes OCB and PMAC. In: *Advances in Cryptology - Asiacrypt'04*. [S.l.]: Springer-Verlag, 2004. (Lecture Notes in Computer Science, v. 3329), p. 16–31. ISBN 3-540-23975-8. http://www.cs.ucdavis.edu/~rogaway/papers/offsets.pdf.

ROGAWAY, P.; BELLARE, M.; BLACK, J. Ocb: A block-cipher mode of operation for efficient authenticated encryption. *ACM Transactions on Information and System Security (TISSEC)*, ACM, New York, NY, USA, v. 6, n. 3, p. 365–403, 2003. ISSN 1094-9224.

ROGAWAY, P.; SHRIMPTON, T. Deterministic authenticated-encryption: A provable-security treatment of the key-wrap problem. In: *Advances in Cryptology - Eurocrypt'06*. Berlin / Heidelberg: Springer-Verlag, 2006. (Lecture Notes in Computer Science, v. 4004), p. 373–390.

ROGAWAY, P.; WAGNER, D. *A Critique of CCM*. 2003. Cryptology ePrint Archive, Report 2003/070. http://eprint.iacr.org/.

RUZZELLI, A. *Wireless Sensor Networks: Enhancing Performance through Integration of MAC and Routing Protocols*. Tese (Doutorado) — University College Dublin - School of Computer Science and Informatics, 2008.

SANDIA. *Submission to NIST: Cipher-State (CS) Mode of Operation for AES*. 2004. http://csrc.nist.gov/groups/ST/toolkit/BCM/documents/proposedmodes/cs/cs-spec.pdf.

SARKAR, P. *A Simple and Generic Construction of Authenticated Encryption With Associated Data*. 2009. Cryptology ePrint Archive, Report 2009/215. http://eprint.iacr.org/.

SASAKI, Y. et al. New message difference for MD4. In: *FSE*. [S.l.]: Springer, 2007. (LNCS, v. 4593), p. 329–348. ISBN 978-3-540-74617-1. http://dx.doi.org/10.1007/978-3-540-74619-5_21.

SHANNON, C. E. Communication theory of secrecy systems. *Bell System Technical Journal*, v. 28, p. 656–715, 1949.

SHOUP, V. *Sequences of games: a tool for taming complexity in security proofs*. Nov. 2004. Cryptology ePrint Archive, Report 2004/332. http://eprint.iacr.org/2004/332.

SIMPLICIO, M. *Algoritmos Criptográficos para Redes de Sensores*. Dissertação (Mestrado) — Escola Politécnica at the University of São Paulo, April 2008. http://www.teses.usp.br/teses/disponiveis/3/3141/tde-30092008-182545/.

SIMPLICIO, M. Curupira-2*: C version for 8-bit platforms (96-bit keys only)*. 2009. http://www.larc.usp.br/~mjunior/en/downloads/index.html.

____. Curupira-2*: Java version for 32-bit platforms (all key-sizes allowed)*. 2009. http://www.larc.usp.br/~mjunior/en/downloads/index.html.

SIMPLICIO, M. et al. The marvin message authentication code and the lettersoup authenticated encryption scheme. *Security and Communication Networks*, v. 2, p. 165–180, 2009.

____. The Curupira-2 block cipher for constrained platforms: Specification and benchmarking. In: *Proc. of the 1st International Workshop on Privacy in Location-Based Applications - 13th European Symposium on Research in Computer Security (ESORICS'2008)*. [S.l.]: CEUR-WS, 2008. v. 397. http://sunsite.informatik.rwth-aachen.de/Publications/CEUR-WS/Vol-397/.

____. A survey on key management mechanisms for distributed wireless sensor networks. *Elsevier Computer Networks Journal*, Submitted.

____. Revisiting the security of the Alred construction. *To be defined*, In production.

____. Revisiting the security of the Alred construction. In: *Selected Areas in Cryptography (SAC 2010)*. [S.l.: s.n.], Submitted.

____. Comparison of message authentication codes on telosb sensor nodes. In: *IEEE Sensys 2010*. [S.l.: s.n.], Submitted.

SOLIS, I.; OBRACZKA, K. The impact of timing in data aggregation for sensor networks. In: *The 2004 International Conference on Communications (ICC'04)*. [S.l.: s.n.], 2004.

STINSON, D. R. *Cryptography: Theory and Practice*. 2nd. ed. Boca Raton, USA: Chapman & Hall/CRC Press, 2002.

TIWARI, A.; BALLAL, P.; LEWIS, F. Energy-efficient wireless sensor network design and implementation for condition-based maintenance. *ACM Trans. Sen. Netw.*, ACM, New York, NY, USA, v. 3, n. 1, p. 1, 2007. ISSN 1550-4859.

WANG, X.; YIN, Y.; YU, H. Finding collisions in the full SHA-1. In: SHOUP, V. (Ed.). *CRYPTO*. [S.l.]: Springer, 2005. (LNCS, v. 3621), p. 17–36. ISBN 3-540-28114-2. http://dx.doi.org/10.1007/11535218_2.

WEGMAN, M.; CARTER, J. New hash functions and their use in authentication and set equality. *Journal of Computer and System Sciences*, v. 22, p. 265–279, 1981.

WILLIAMSON, M. J. *Non-Secret Encryption Using a Finite Field*. [S.l.], 1974. http://www.cesg.gov.uk/site/publications/media/secenc.pdf.

YAHYA, B.; BEN-OTHMAN, J. Towards a classification of energy aware MAC protocols for wireless sensor networks. *Wirel. Commun. Mob. Comput.*, John Wiley and Sons Ltd., Chichester, UK, v. 9, n. 12, p. 1572–1607, 2009. ISSN 1530-8669.

YICK, J.; MUKHERJEE, B.; GHOSAL, D. Wireless sensor network survey. *Computer Networks*, Elsevier North-Holland, Inc., New York, NY, USA, v. 52, n. 12, p. 2292–2330, 2008. ISSN 1389-1286.

YLONEN, T.; LONVICK, C. *The secure shell (SSH) transport layer protocol – RFC 4253*. 2006. http://www.ietf.org/rfc/rfc4253.txt.

ZHANG, P. et al. Hardware design experiences in zebranet. In: *SenSys '04: Proc. of the 2nd international conference on Embedded networked sensor systems*. New York, NY, USA: ACM, 2004. p. 227–238. ISBN 1-58113-879-2.

ZHANG, W.; ZHU s.; CAO, G. Pre-distribution and local collaboration-based group re-keying for wireless sensor networks. *Ad Hoc Networks*, Elsevier Science Publishers B. V., Amsterdam, The Netherlands, v. 7, n. 6, p. 1229–1242, 2009. ISSN 1570-8705.

ZIGBEE. *Zigbee Specification – Document 053474r06, Version 1.0*. [S.l.], 2005.

APPENDIX A – SOME MATHEMATICAL PROPERTIES AND DEFINITIONS

Along this thesis, we used some definitions and properties commonly used in the domain of Cryptography without further explanation, aiming to create a more concise and focused discussion. In this appendix, we aim to close this gap, presenting an overview of such concepts. Therefore, this appendix is intended mostly for the readers who are unfamiliar with some of these properties, or who are looking for a quick definition.

A.1 Finite Fields

The elements of a finite field can be represented in many different forms. For any power of a prime number, there is always a single finite field, which assures that all representations of GF(2^n) (the finite field with 2^n elements) are isomorphic. Despite this equivalence, the representation adopted has influence over its implementation. Along this document, we used the classical polynomial representation: an n-bit element, composed by bits $[u_n u_{n-1} \dots u_1 u_0]$, is interpreted as a polynomials whose coefficients belong to the set $\{0, 1\}$, i.e., $u = u_n x^n + u_{n-1} x^{n-1} + \cdots + u_1 x + u_0$.

In our examples, we assume that $n = 8$, which means that the elements we are working with are bytes. However, the same discussion could be applied to different values of n.

Example: The byte represented by the hexadecimal value `0x57` (binary value: 01010111) corresponds, in GF(2^8), to the polynomial $x^6 + x^4 + x^2 + x + 1$.

A.1.1 Addition in GF(2^n)

The addition of two polynomials is given by the addition modulo 2 of their coefficients.

Example: `0x57` + `0x83` = `0xD4` or, in polynomial notation

$$(x^6 + x^4 + x^2 + x + 1) + (x^7 + x + 1) = x^7 + x^6 + x^4 + x^2$$

It is easier to see that the addition in GF(2^n) corresponds to the bitwise XOR operation (represented as $\oplus$) when we analyze it in the binary form: $01010111 + 10000011 = 11010100$. Note that the subtraction operation also corresponds to $\oplus$, since each element is its own additive inverse (*i.e.*, $u + u \equiv u - u \equiv u \oplus u \equiv 0$).

A.1.2 Multiplication in GF(2^n)

With the polynomial notation, we represent the finite field with 2^n elements as GF(2^n) $\equiv$ GF(2)$[x]/p(x)$, for an irreducible polynomial[1] $p(x)$ of degree n, such that x is a generator[2] modulo $p(x)$. With this representation, the multiplication in GF(2^n) $\equiv$ GF(2)$[x]/p(x)$ corresponds to the multiplication of polynomials modulo $p(x)$. The following example illustrates how multiplication is performed in the field GF(2)$[x]/p(x)$ for $p(x) = x^8 + x^6 + x^3 + x^2 + 1$ (`0x14D` in hexadecimal):

Example: `0x57` $\cdot$ `0x83` = `0x43` or, in polynomial notation:

$$(x^6 + x^4 + x^2 + x + 1) \cdot (x^7 + x + 1)$$
$$= x^{13} + x^{11} + x^9 + x^8 + x^7 + x^7 + x^5 + x^3 + x^2 + x + x^6 + x^4 + x^2 + x + 1$$

[1] An irreducible polynomial is a polynomial that has no divisors except by 1 and itself (something similar to prime numbers when one is working in the domain of entire numbers)

[2] A polynomial $g(x)$ is a generator modulo $p(x)$ iff any element u of GF(2^n) can be obtained by $(u \cdot x^m)$ mod $p(x)$ for some integer m.

$$= (x^{13} + x^{11} + x^9 + x^8 + x^6 + x^5 + x^4 + x^3 + 1) \bmod (x^8 + x^6 + x^3 + x^2 + 1)$$
$$= x^6 + x + 1$$

The reduction using the n-degree polynomial $p(x)$ is necessary in order to assure that any multiplication will give a whose degree is below n — i.e., an element of GF(2^n). Unfortunately, there is no simple bitwise operation (such as a XOR) that can be used when multiplying in GF(2^n). However, the multiplication by x can be performed in a reasonably efficient way: it consists in a bitwise left-shift (represented as $\ll$) followed by a conditional XOR with the least bits of the reduction polynomial $p(x)$, as follows:

Algorithm A.1 Multiplication by x in GF(2^n)

INPUT: u ▷ n-bit value to multiply by x.
OUTPUT: $u \cdot x$.
1: $v \leftarrow (u \ll 1)$
2: **if** $testbit(u, n-1) = 1$ **then** ▷ check the most significant bit of u
3: $v \leftarrow v \oplus (p(x) \oplus x^n)$
4: **end if**
5: **return** v

Irreducible polynomials typically used in cryptographic solutions are the lexicographically first among all such polynomials. Table 12 shows some of these polynomials.

Table 12: Least irreducible polynomials for some selected degrees — adapted from (SANDIA, 2004).

Degree	Primitive Polynomial	Low-order Portion (Hex)
64	$x^{64} + x^4 + x^3 + x + 1$	`0x1B`
96	$x^{96} + x^7 + x^6 + x^4 + x^3 + x^2 + 1$	`0xDD`
128	$x^{128} + x^7 + x^2 + x + 1$	`0x87`
160	$x^{160} + x^5 + x^3 + x^2 + 1$	`0x2D`
192	$x^{192} + x^8 + x^6 + x^4 + x^3 + x^2 + 1$	`0x15D`
224	$x^{224} + x^8 + x^7 + x^5 + x^4 + x^2 + 1$	`0x1B5`
256	$x^{256} + x^{10} + x^5 + x^2 + 1$	`0x425`
512	$x^{512} + x^8 + x^5 + x^2 + 1$	`0x125`
1024	$x^{1024} + x^9 + x^8 + x^7 + x^5 + x + 1$	`0x3A3`

A.2 MDS Codes

We provide a few relevant definitions regarding the theory of linear codes. For a more extensive exposition on the subject, we refer to (MACWILLIAMS; SLOANE, 1977).

The Hamming distance between two vectors u and v from the n-dimensional vector space $\mathrm{GF}(2^p)^n$ is the number of coordinates where u and v differ. The Hamming weight $\mathrm{w}(a)$ of an element $a \in \mathrm{GF}(2^p)^n$ is the Hamming distance between a and the null vector of $\mathrm{GF}(2^p)^n$, i.e., the number of nonzero components of a.

A *linear* $[n, k, d]$ *code* over $\mathrm{GF}(2^p)$ is a k-dimensional subspace of the vector space $\mathrm{GF}(2^p)^n$, where the Hamming distance between any two distinct subspace vectors is at least d, and d is the largest number with this property.

A *generator matrix* G for a linear $[n, k, d]$ code C is a $k \times n$ matrix whose rows form a basis for C. A generator matrix is in *echelon* or *standard* form if it has the form $G = [I_{k\times k}\, A_{k\times(n-k)}]$, where $I_{k\times k}$ is the identity matrix of order k.

Linear $[n, k, d]$ codes obey the *Singleton bound*: $d \leqslant n-k+1$. A code that meets this bound, i.e., which displays $d = n-k+1$, is called a *Maximal Distance Separable (MDS)* code. A linear $[n, k, d]$ code C with generator matrix $G = [I_{k\times k}\, A_{k\times(n-k)}]$ is MDS if, and only if, every square sub-matrix formed from rows and columns of A is nonsingular — cf. (MACWILLIAMS; SLOANE, 1977, chapter 11, § 4, theorem 8).

A.3 Cryptographic Properties

A product of m distinct Boolean variables is called an m-th order product of the variables. Every Boolean function $f : \mathrm{GF}(2)^n \to \mathrm{GF}(2)$ can be written as a sum over $\mathrm{GF}(2)$ of distinct m-order products of its arguments, with $0 \leqslant m \leqslant n$; this is called the algebraic normal form of f. The *nonlinear order* of f, denoted $\nu(f)$, is the maximum order of the terms appearing in its algebraic normal form.

A *linear* Boolean function is a Boolean function of nonlinear order 1, i.e., its algebraic normal form only involves isolated arguments. Given $\alpha \in \mathrm{GF}(2)^n$, we denote by $\ell_\alpha : \mathrm{GF}(2)^n \to GF(2)$ its *parity*, i.e., the linear Boolean function consisting of the sum of the argument bits selected according to the bits of α:

$$\ell_\alpha(x) = \bigoplus_{i=0}^{n-1} \alpha_i \cdot x_i.$$

For an array $x = (x_1, \ldots, x_n)$, the pattern of x, denoted γ_x, is defined by $\gamma_x = (\gamma_1, \ldots, \gamma_n) \in Z_2^n$, where $\gamma_i = 1$ if $x_i \neq 0$, and $\gamma_i = 0$ otherwise.

A mapping $S : \mathrm{GF}(2^n) \to \mathrm{GF}(2^n), x \mapsto S[x]$, is called a *substitution box*, or S-box for short. An S-box can also be viewed as a mapping $S : \mathrm{GF}(2)^n \to \mathrm{GF}(2)^n$ and therefore described in terms of its component Boolean functions $s_i : \mathrm{GF}(2)^n \to \mathrm{GF}(2), 0 \leqslant i \leqslant n-1$, i.e., $S[x] = (s_0(x), \ldots, s_{n-1}(x))$.

The *nonlinear order* of an S-box S, denoted ν_S, is the minimum nonlinear order over all linear combinations of the components of S. Hence, it measures the difficulty of representing the (non-linear) function defined by the S-Box as a linear function:

$$\nu_S = \min_{\alpha \in \mathrm{GF}(2)^n} \{\nu(\ell_\alpha \circ S)\}.$$

Let a denote the input difference of an S-Box, and let b denote the difference between the corresponding outputs. The pair (a, b) is called a *differential*.

The *difference table* of an S-box S, built using the relationship between the differences at the input and the resulting differences at the output of that S-Box (i.e., differentials), is defined as

$$e_S(a, b) = \#\{c \in \mathrm{GF}(2^n) \mid S[c \oplus a] \oplus S[c] = b\}.$$

The *δ-parameter* of an S-box S, which indicates the maximum probability of a

given differential in table e_S, is defined as

$$\delta_S = 1/2^n \cdot \max_{a \neq 0, b} e_S(a, b).$$

The product $\delta_S \cdot 2^n$ is called the *differential uniformity* of S.

A *differential trail* is defined as a series of differentials spanning a certain number of rounds and S-Boxes. An S-Box that is part of such trail is said to be *active*. Since each differential has a probability associated to it, the higher the number of active S-Boxes in a trail, the lower the probability of that trail.

The *correlation* $c(f, g)$ between two Boolean functions f and g can be calculated as follows:

$$c(f, g) = 2^{1-n} \cdot \#\{x \mid f(x) = g(x)\} - 1.$$

The λ-*parameter* of an S-box S is defined as the maximal value for the correlation between linear functions of input bits and linear functions of output bits of S:

$$\lambda_S = \max_{(i,j) \neq (0,0)} c(\ell_i, \ell_j \circ S).$$

The *branch number* $\mathcal{B}_d$ of a linear mapping $\theta : \mathrm{GF}(2^n) \rightarrow \mathrm{GF}(2^n)$ from the point of view of differential cryptanalysis is defined as

$$\mathcal{B}_d(\theta) = \min_{a \neq 0} \{\mathrm{w}(a) + \mathrm{w}(\theta(a))\}.$$

The branch number is useful because it determines the minimum number of active S-Boxes after the application of θ, as a function of the number of active S-Boxes prior to its application (e.g., if $\mathcal{B}_d(\theta) = b$, a single active S-Box at the input of θ will lead to at least $b - 1$ active S-Boxes at its output).

APPENDIX B – ON THE SECURITY OF SOME PRACTICAL SCTS

The design of Square-Complete Transforms (SCTs) presented in this thesis is based both on the results introduced in the original ALRED specification (DAEMEN; RIJMEN, 2005a), and on some practical results concerning the security of Substitution-Permutation Networks (SPNs).

One round of SPN structures typically consists of three distinct layers, as depicted in Figure 28: the key addition, the substitution and the linear transformation. On the key addition layer, the cipher's round subkeys are XORed with the round input data. The substitution layer is composed by n non-linear, bijective S-Boxes. Finally, the linear transformation layer uses a bijective linear transformation for diffusing the cryptographic characteristics of the substitution layer.

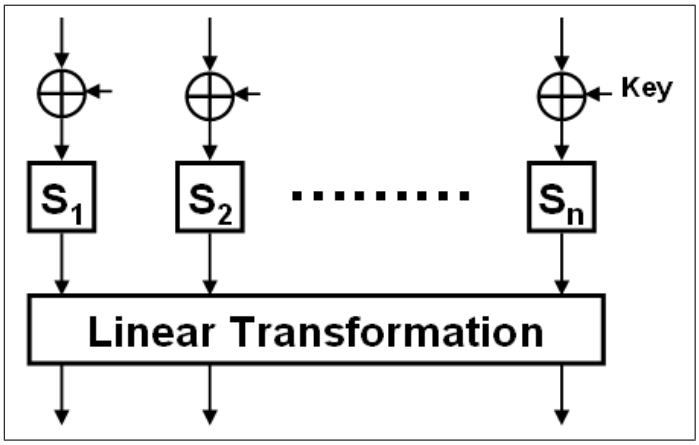

Figure 28: One round of the SPN structure

The SPN structure is among the most commonly adopted strategies for the design

of modern block ciphers, and has been target of intense scrutinity, particularly concerning powerful techniques such as linear and differential cryptanalysis (KANG et al., 2001; KELIHER; MEIJER; TAVARES, 2001b; KELIHER; MEIJER; TAVARES, 2001a; CHUN et al., 2003). Most of these analyses can be directly applied to AES and other Rijndael-like structures (PARK et al., 2002; PARK et al., 2003; KELIHER, 2004; DAEMEN; RIJMEN, 2006), which has a high practical interest since many modern ciphers follow or are at least influenced by this design.

Here, we describe some important results concerning the maximum differential probability of AES and then apply the same concepts to the CURUPIRA block cipher (BARRETO; SIMPLICIO, 2007; SIMPLICIO et al., 2008), showing the reasoning that supports our assumptions on the security of SCTs as discussed in Chapter 5.

B.1 Preliminaries

The notation henceforth used is based on the definitions given in Appendix A and on the following:

Definition 3. *Rijndael-like structures are the block ciphers composed of SPN structures satisfying the following:*

1. *Their linear transformation has the form* $(\theta_1, \theta_2, \theta_3, \theta_4) \circ \pi$
2. *(The condition of* π*) Each of the bytes of* y_i *comes from each different* x_i*, where* $x = (x_1, x_2, x_3, x_4)$ *is the input of* π *and* $y = (y_1, y_2, y_3, y_4)$ *is the output of* π*.*
3. *(The condition of* $\theta = (\theta_1, \theta_2, \theta_3, \theta_4)$*) When we consider each of* θ_i *as a linear transformation, the following hold:* $\mathcal{B}_d^{\theta_1} = \mathcal{B}_d^{\theta_2} = \mathcal{B}_d^{\theta_3} = \mathcal{B}_d^{\theta_4}$

Demonstrating a cipher's security against differential cryptanalysis involves proving that the Maximum Expected Differential Probability (MEDP) is sufficiently small

over r of its rounds. Computing the exact value of the MEDP is a difficult task and, for this reason, researchers usually focus on bounds.

To date, the best known results for Rijndael-like ciphers can be obtained from the technique described in (PARK et al., 2003), and is based in the following theorem[1]:

Theorem 5. *Let $\mathcal{B}_d$ be the branch number of the linear transformation θ from the viewpoint of differential cryptanalysis. Then, the maximum probability of a differential (a, b) for 2-rounds of SPN structure is bounded by:*

$$Pr_2^{\theta}[a,b] \leqslant \max\left\{ \max_{1\leqslant u\leqslant 255} \sum_{j=1}^{255} \{e_S(u,j)\}^{\mathcal{B}_d} \quad ,\ldots, \quad \max_{1\leqslant u\leqslant 255} \sum_{j=1}^{255} \{e_S(j,u)\}^{\mathcal{B}_d} \right\}$$

By applying Theorem 5 to Rijndael-like structures, one can show that they satisfy (PARK et al., 2002):

Theorem 6.

$$Pr_2[a,b] \leqslant \begin{cases} \delta_S^{\mathrm{w}(\gamma_{\pi(a)})(\mathcal{B}_d-1)}, & if\ \gamma_{\pi(a)} = \gamma_b, \\ 0, & otherwise. \end{cases}$$

B.2 Upper Bounding the Maximum Differential Probability of AES

AES is a block cipher that follows the SPN structure. It is easy to show that AES displays a Rijndael-like structure, where the $\pi : (Z_2^8)^{16} \rightarrow (Z_2^8)^{16}$ linear transformation is called *ShiftRows*, and the $\theta = (\theta_1, \theta_2, \theta_3, \theta_4)$ linear transformation is called *MixColumns*. Both are depicted in Figure 29.

As shown in the figure, AES operates over 4×4 byte blocks, and its x and y arrays

[1] We note that the formulation presented in (PARK et al., 2003, Theorem 1) is slightly different than the expression described here, but this simplified version is enough for our purposes.

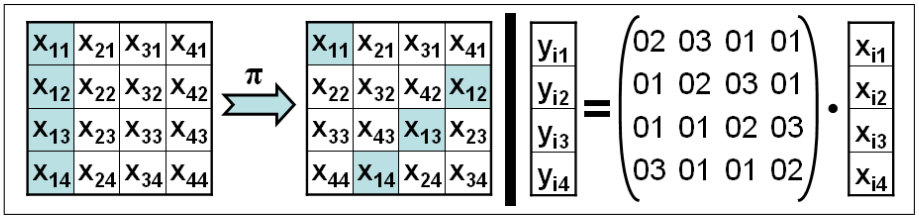

Figure 29: AES: ShiftRows (left) and MixColumns (right) transformations

assume the forms:

$$(x_1, x_2, x_3, x_4) = (x_{11}, x_{12}, x_{13}, x_{14},\ x_{21}, x_{22}, x_{23}, x_{24},\ x_{31}, x_{32}, x_{33}, x_{34},\ x_{41}, x_{42}, x_{43}, x_{44});$$

$$(y_1, y_2, y_3, y_4) = (y_{11}, y_{12}, y_{13}, y_{14},\ y_{21}, y_{22}, y_{23}, y_{24},\ y_{31}, y_{32}, y_{33}, y_{34},\ y_{41}, y_{42}, y_{43}, y_{44}).$$

By applying Theorem 5 to AES, whose θ operation displays $\mathcal{B}_d = 5$ and whose S-box is highly non-linear and well behaved, we can write (PARK et al., 2003; KELIHER; SUI, 2007): $Pr_2^{\theta}[a, b] \leqslant \sum_{j=1}^{255}\{e_S(1, j)\}^5 = 53 \times 2^{-34}$.

By combining the above result with Theorem 6, we arrive at $Pr_2[a, b] \leqslant (53 \times 2^{-34})^{\mathrm{w}(\gamma_{\pi(a)})}$ when $\gamma_{\pi(a)} = \gamma_b$. Furthermore, assuming the minimum number of active S-Boxes in these rounds (i.e., $\mathrm{w}(\gamma_{\pi(a)}) = 1$), we can rewrite this upper bound as $Pr_2[a, b] \leqslant (53 \times 2^{-34})$.

Finally, this result can be further expanded to 4 rounds of the cipher, leading to the following bound (PARK et al., 2003; KELIHER; SUI, 2007): $Pr_4[a, b] \leqslant (\max\{Pr_2^{\theta}[a, b]\})^4 \leqslant (53 \times 2^{-34})^4 \approx 1.881 \times 2^{-114}$.

B.3 Upper Bounding the Maximum Differential Probability of Curupira

The Curupira block cipher also follows the SPN structure. As such, its maximum differential probability is also bounded according to Theorem 5. The cipher displays $\mathcal{B}_d = 4$ and, using a computer program for evaluating its S-Box, we found the following upper bound for two rounds of the cipher: $Pr_2^{\theta}[a, b] \leqslant \max_{1 \leqslant u \leqslant 255} \sum_{j=1}^{255}\{e_S(j, u)\}^4 =$

$21536 \times 2^{-32} \approx 1.3145 \times 2^{-18}$.

Figure 30 illustrates CURUPIRA's linear transformation $\pi : (Z_2^8)^1 2 \rightarrow (Z_2^8)^1 2$ and a $\theta = (\theta_1, \theta_2, \theta_3, \theta_4)$. As shown in this figure, CURUPIRA's design does not strictly follow the Rijndael-like structure as given in Definition 3: it operates over 3×4 byte blocks, and is unable to satisfy the "condition of π" because each of the bytes of y_i comes from a different x_i, but not from *all* possible x_i (e.g., the bytes of y_1 come from x_1, x_2 and x_3, but not from x_4).

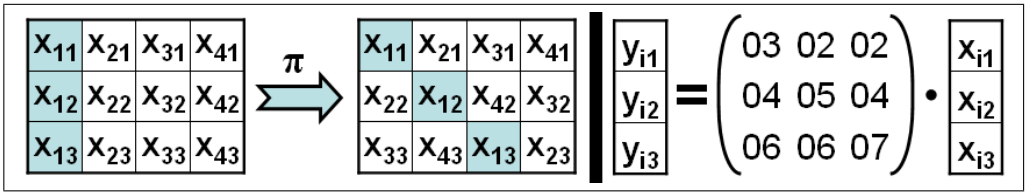

Figure 30: CURUPIRA: permutation (left) and diffusion (right) layers

Nonetheless, the proof of Theorem 6 given in (PARK et al., 2002), which builds on the work done in (KANG et al., 2001; HONG et al., 2001), applies both to ciphers that have $B_d = n + 1$ |e.g., Rijndael-like ciphers ($n = 4$, $B_d = 5$) — and to ciphers displaying $B_d = n$ — e.g., CURUPIRA and similar ciphers such as BKSQ (DAEMEN; RIJMEN, 1998) ($n = 4$, $B_d = 4$). Thus, we can also apply Theorem 6 to CURUPIRA and write $Pr_2[a, b] \leqslant (21536 \times 2^{-32})^{w(\pi(a))}$ when $\gamma_{\pi(a)} = \gamma_b$.

Finally, this result can be used for computing the maximum differential probability of 4 rounds of CURUPIRA-like ciphers. The analysis is analogous to the one given in (PARK et al., 2003, Theorem 5), for Rijndael-like block ciphers, with the main difference that we use $\mathcal{B}_d = 4$ instead of $\mathcal{B}_d = 5$. Therefore, we can write $Pr_4[a, b] \leqslant (\max\{Pr_2^{\theta}[a, b]\})^4 \leqslant (21536 \times 2^{-32})^4 \approx 1.4926 \times 2^{-71}$.

APPENDIX C - ON THE ALGORITHMS' NAMES

The name MARVIN is a tribute to the paranoid android from "The Hitchhiker's Guide to the Galaxy" series by Douglas Adam (ADAMS, 1979). As stated in Wikipedia when this document was written[1]:

"Marvin is the ship's robot aboard the starship Heart of Gold. He was built as a prototype of Sirius Cybernetics Corporation's GPP (Genuine People Personalities) technology. Marvin is apparently afflicted with severe depression and boredom, in part because he has a 'brain the size of a planet' which he is seldom able to use. Indeed, the true horror of Marvin's existence is that no task he could be given would occupy even the tiniest fraction of his vast intellect. He claims he is 50,000 times more intelligent than a human, (or 30 billion times more intelligent than a live mattress) though this is, if anything, a vast underestimation. When kidnapped by the bellicose Krikkit robots and tied to the interfaces of their intelligent war computer, Marvin simultaneously manages to plan the entire planet's military strategy, solve all of the major mathematical, physical, chemical, biological, sociological, philosophical, etymological, meteorological and psychological problems of the Universe except his own, three times over, and compose a number of lullabies. He seemed to find this last task the hardest, and only one, 'How I Hate the Night', is known".

We do not claim (well, at least for now...) that our MAC algorithm is that "powerful", though it adopts the "answer to the ultimate question of life" (ADAMS, 1979) as its

[1] http://en.wikipedia.org/wiki/Marvin_the_Paranoid_Android

constant. However, we do believe that it is much more useful, even for very paranoid users. ☺

Concerning LetterSoup, the name was chosen really for the sake of pronunciation, as stated in section 4.3: correctly writing and speaking "L-F-S-R-C-Marvin", or "L-F-S-R-C-A-E-A-D scheme" has proven to be a thorny task. In the end, we accepted that fact and decided to give a more suitable and descriptive name for the algorithm. Some time later, we noted that most designers of AEAD schemes love to name their algorithms after acronyms (OCB, GCM, EAX, CCM, IAPM, XCBC, XECB, just to cite a few that appear in this document). However, it was too late to go back, since we had already published the paper and taken the picture to represent our creation, as shown in Figure 31. ☺

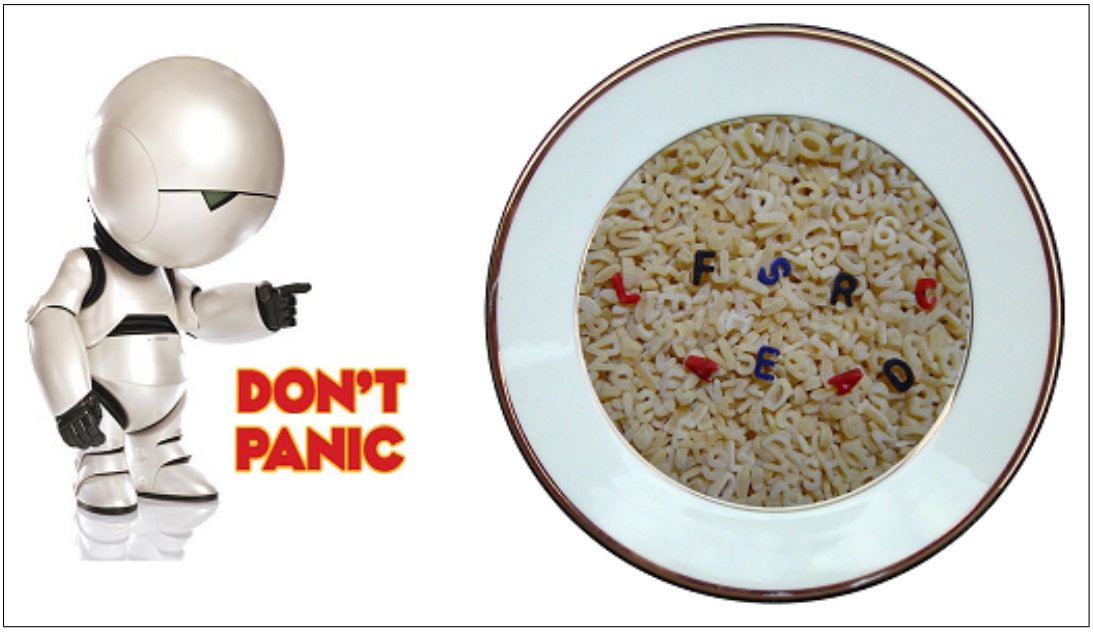

Figure 31: Marvin (left)[2] and LetterSoup (right)[3] illustrations.

[2]©Buena Vista Pictures Distribution.
[3]©Marcos and Débora Simplício — use this image at will, but beware of the amount of carbs.