Lightweight cryptography for use in low-capacity computers

Ascon

Laurits Gorm Dahl

Department of Mathematical Sciences
Skjernvej 4A
DK-9220 Aalborg Ø
http://math.aau.dk

AALBORG UNIVERSITET
STUDENTERRAPPORT

Titel
Lightweight cryptography for use in low-capacity computers

Theme
Cryptography

Project period
Sep. 01st. 2023 - June. 03rd. 2024

Project group
4.105 B

Authors
Laurits Gorm Dahl

Supervisors
Oliver Whilhelm Gnilke
Matteo Bonini

Number of pages
74

Due date
June 3, 2024

Abstract

This thesis investigates lightweight cryptography with a focus on the AEAD Ascon cipher. It offers an overview of lightweight cryptography principles and practical applications, alongside an in-depth analysis of Ascon ’s security and efficiency.
The thesis begins with foundational cryptographic concepts and key aspects of cryptanalysis, including various attack types. It proceeds to examine the design considerations and requirements of lightweight cryptography.
An exploration of the AEAD Ascon cipher follows, detailing its encryption schemes, phases, permutation process, and security properties, and comparing Ascon -128 with AES-128-GCM.
Cryptanalytic techniques, such as linear, differential and differential-linear cryptanalysis, are also explored.
Confidentiality and authenticity, particularly under state recovery, are addressed, with a focus on Ascon ’s nonce-respecting setting and the necessity of its key blinding technique for authenticity.
An experimental analysis of the randomness of Ascon ’s permutations concludes the thesis.

Unlike in ethics, two wrongs do make a right in mod-2 arithmetic.
-Langford and Hellman [14, page 22]

Acknowledgments

I will make this as short as I can. For the last 18 years of my life, mathematics has always been something I could count on.

Since I was a little boy, mathematics has always fascinated me. I fondly remember being a kindergartener, sitting with my older brother, doing his homework while our parents were not looking. Soon, they found out, and they gave me my own math book. For this, I am thankful.

I would also like to express my gratitude to Oliver and Matteo for answering all my (sometimes questionable) questions over the last couple of years. I especially thank You for the work You have done this semester. Without Your help, this thesis would not have been possible.

Finally, I want to thank my fellow students, who have been great friends for the last five years and, hopefully, for many more to come.

Introduction

This thesis explores the domain of lightweight cryptography with a specific focus on the AEAD Ascon cipher, a state-of-the-art lightweight cryptographic algorithm. The structure of this thesis is designed to provide a comprehensive understanding of the principles and practical applications of lightweight cryptography, as well as an in-depth analysis of the Ascon cipher's security and efficiency.

The foundation of this thesis is laid in Chapter 1, where we delve into essential cryptographic functions and concepts. This chapter provides key aspects of cryptanalysis and various types of attacks, such as semantic security, chosen plaintext, chosen ciphertext, known plaintext, and known ciphertext attacks. Additionally, we introduce the substitution-permutation network and a toy cipher to illustrate basic cryptographic principles. The chapter concludes with a discussion on entropy.

Chapter 2 transitions into the specific domain of lightweight cryptography. Here, we examine the considerations and requirements that drive the design of lightweight cryptographic algorithms. This chapter sets the stage for understanding the constraints and design choices that influence the development of ciphers intended for low-resource environments.

Chapter 3 provides a detailed exploration of the AEAD Ascon cipher, beginning with an overview of its encryption schemes. We break down the phases of Ascon , analyze its permutation process, and evaluate its claimed security properties. This chapter also includes a comparison of Ascon -128 and AES-128-GCM.

In Chapter 4, we delve into advanced cryptanalytic techniques. We provide a generic description of linear and differential cryptanalysis and explain the ideas of differential-linear cryptanalysis. An application of these techniques to Ascon , presenting cryptanalytic results that assess its resilience against these forms of attacks, is also included.

Chapter 5 addresses the critical aspects of confidentiality and authenticity. We comment on the confidentiality of Ascon in the nonce respecting setting. We also prove that the key blinding technique used in Ascon is necessary for authenticity under state recovery.

Chapter 6 presents an experimental analysis of the randomness of the permutations used in Ascon . This chapter details the experimental setup, methodologies, and results, providing empirical evidence of the randomness.

Contents

Chapter 1 Preliminary readings

1.1 Some functions and how they work

This section presents several logical Boolean functions. Each is equipped with a short example, and their truth tables are given at the end. We also define linear and affine functions.

AND

The bitwise Boolean function AND is equivalent to the bitwise multiplication of vectors over $\mathbb{F}_2^n$. It only outputs 1 whenever both inputs to the function are 1. For $x, y \in \mathbb{F}_2^n$, we denote the AND function by $x \cdot y$ or simply xy when there is no chance of ambiguity. The truth table for the AND function can be found in Table 1.1.

Example 1.1
Let $x = (100101)$ and let $y = (001011)$, then $x \cdot y = (00001)$, as x and y only agree on a 1 in the last position.

XOR

The bitwise Boolean function XOR stands for exclusive or, and it is equivalent to the bitwise addition of vectors over $\mathbb{F}_2^n$. It outputs 1 whenever the inputs to the function are different. For $x, y \in \mathbb{F}_2^n$, we denote the XOR function by $x \oplus y$. The truth table for the exclusive or function can be found in Table 1.1.

Example 1.2
Let $x = (100101)$ and let $y = (001011)$, then $x \oplus y = (101110)$, as x and y are different in entries $1, 3, 4$ and 5.

NOT

The bitwise Boolean function NOT function differs from the above function as it does not take multiple inputs but a single one. For $x \in \mathbb{F}_2^n$ we denote the NOT function by $\neg x$. It functions by flipping its input: an input of 0 becomes a 1, and an input of 1 becomes a 0. The truth table for NOT function function can be found in Table 1.1.

Example 1.3
Let $x = (100101)$ then $\neg x = (011010)$.

x	y	$x \cdot y$	$x \oplus y$	$\neg x$
0	0	0	0	1
0	1	0	1	1
1	0	0	1	0
1	1	1	0	0

Table 1.1: *The truth table for the Boolean functions AND, XOR and NOT*

We now give the definitions of linear and affine functions.

Definition 1.4 Linear and affine functions
A linear function is any function $f : V \to W$ that fulfills the following conditions:

- For any $x, y \in V$ it holds that $f(x) + f(y) = f(x + y)$
- For any $\lambda \in V$ it holds that $\lambda f(x) = f(\lambda x)$.

An affine function g is the composition of a linear function with a translation, that is, $f(x) + y$ is affine for any $x, y \in W$

1.2 Cryptanalysis

This section is a preliminary section, introducing, among other things, the attacks used for cryptanalysis, such as linear cryptanalysis and differential cryptanalysis. The sections on semantic security, chosen plaintext attack, chosen ciphertext attack, known plaintext attack, and known ciphertext attack are written based on [10].

We introduce the idea of types of four different cryptographic attacks. But firstly, we need a measure of security, and one measure that will be used in this thesis is semantic security, which we will now introduce.

1.2.1 Semantic security

Semantic security is a computational-complexity analog of Shannon's definition of perfect privacy, in which the ciphertext should reveal no information about the plaintext. Semantic security states that what can be effectively computed from the ciphertext can also be effectively computed by only looking at the length of the plaintext.

An encryption scheme is semantically secure if it is infeasible to derive information other than information about the length of the plaintext from the ciphertext. In the definition of semantic security, we define the information relating to the plaintext that the adversary tries to obtain by the function f, and the preexisting partial information about the plaintext is denoted by the function h.

The infeasibility of gaining information about the plaintext has to hold for any distribution of plaintexts, represented by the collection of probability distributions $\{X_m\}_{m \in \mathbb{N}}$. Finally, we only look at security for plaintext of polynomial length in a security parameter n, meaning that $|X_n|$, $X_n \in \{X_m\}_{m \in \mathbb{N}}$, has to have polynomial length of at most $\text{poly}(n)$, where $\text{poly}(n)$ is some polynomial in n of unbounded degree. We cannot provide computational security of plaintexts of unbounded length or exponential length in a security parameter n, which we will discuss later.

We also restrict the length of the functions f, h to be polynomially bounded, that is, $|f(X_n)|, |h(X_n)| \leq \text{poly}(|X_n|)$. When dealing with an asymmetric cipher, the adversary is given the encryption key, whereas when dealing with a symmetric cipher, it is not. We now define semantic security in the setting of a symmetric cipher.

Definition 1.5 Semantic security - symmetric cipher
An encryption scheme $(G, \mathcal{E}, \mathcal{D})$, consisting of a key generator G, an encryption algorithm $\mathcal{E}$ and a decryption algorithm $\mathcal{D}$, is said to be semantically secure if for every probabilistic polynomial-time algorithm A there exists a probabilistic polynomial-time algorithm A' such that for every collection of probability distributions $\{X_m\}_{m \in \mathbb{N}}$ with

$X_n \in \{X_m\}_{m\in\mathbb{N}}$ and $|X_n| \leq \mathrm{poly}(n)$, every pair of polynomially bounded functions $f, h : \mathbb{F}_2^* \to \mathbb{F}_2^*$, every positive polynomial $p(n)$ and all sufficiently large n it holds that

$$\begin{aligned}&\Pr[A(n, \mathcal{E}_{K_e}(X_n), |X_n|, h(n, X_n) = f(n, X_n)]\\ &< \Pr[A'(n, |X_n|, h(n, X_n)) = f(n, X_n)] + \frac{1}{p(n)},\end{aligned}$$

where $\mathcal{E}_{K_e}(X_n)$ is the ciphertext of X_n given an arbitrary encryption key K_e.

A probabilistic polynomial-time algorithm is an algorithm for which the running time is bounded by a polynomial in the length of the input.

Example 1.6
Suppose that the definition of semantic security is modified so that no bounds exist on the length of plaintexts X_n. Let $|X_n|$ be of exponential length in the security parameter. As the adversary runs in polynomial time of its input length $|X_n|$, it runs in $\mathrm{poly}(\exp(n))$ which is considerably larger than $\mathrm{poly}(n)$. This allows the adversary to find the key by brute force.

The function h imparts partial information about the plaintext X_n to both algorithms. Additionally, both algorithms receive the length of X_n. Subsequently, these algorithms attempt to predict the value of $f(n, X_n)$, aiming to deduce information about the plaintext X_n. In a semantically secure encryption scheme, loosely speaking, the ciphertext does not aid in this inference task. Specifically, the success probability of any efficient algorithm (denoted as A) given the ciphertext can be closely approximated, up to a negligible fraction, by the success probability of an efficient algorithm (denoted as A') that operates without access to the ciphertext at all. Now, having defined what security means, we discuss different types of attacks.

1.2.2 Chosen plaintext attack

We now present a slightly simplified version of a chosen plaintext attack in the scenario of a symmetric cipher. For further details, consult [10].

An attack in which the adversary obtains ciphertexts corresponding to plaintexts of their own choice is called a chosen plaintext attack. One way this can happen is that the adversary either directly or indirectly gains access to the encryption module. Also, if the cryptosystem the adversary is attacking is a symmetric cipher, the key used for encryption is the same as that used for decryption, which may make it more vulnerable to this type of attack. We define the attack in four steps: key generation by a legitimate party, the adversary's request for encryption, generation of a challenge ciphertext, and additional requests for encryption under the same key. There are two main types of chosen plaintext attacks: batch attacks and adaptive attacks. In a batch-chosen plaintext attack, the adversary chooses a set of plaintexts, submits them for encryption, and attempts to identify patterns or correlations between the plaintexts and ciphertexts to uncover the key. In an adaptive-chosen plaintext attack, the adversary can select plaintexts sequentially, adjusting their choices based on the feedback received from previous ciphertexts. This grants the adversary increased flexibility. We now discuss the four steps of a chosen plaintext attack in more detail.

1. *Key generation:* A key $K_e \in \mathbb{F}_2^n$ is generated, and the adversary is only given the length of the key.

2. *Encryption requests:* The adversary A requests plaintext X_n of its own choice to be encrypted, and such a request is answered with the corresponding ciphertexts $\mathcal{E}_{K_e}(X_n)$. After many such requests, we move on to the next step.

3. *Challenge generation:* The adversary A specifies a challenge template and is given an actual challenge. The challenge template is a triplet (S_m, h_m, f_m), where S_m specifies a distribution of plaintexts over $\mathbb{F}_2^m$ and $h_m, f_m : \mathbb{F}_2^m \to \mathbb{F}_2^*$. The actual challenge pair $(\mathcal{E}_{K_e}(X), h(X))$ is generated by the adversary, where X is distributed according to $S_m(U_{\text{poly(n)}})$, where $U_{\text{poly(n)}}$ is a uniform distribution of length poly(n). The adversary then succeeds if $A(n, \mathcal{E}_{K_e}(X_n), |X_n|, h(n, X_n)) = f(n, X_n)$.

4. *Additional encryption requests:* The adversary can seek the encryptions of extra plaintexts at its discretion. These inquiries are managed as outlined when requesting encryptions. Following multiple such requests, the adversary generates an output and concludes its actions.

In reality, the adversary's strategy is split into two parts, corresponding to its actions before and after the generation of the challenge. Each part will be represented by an *oracle machine*, where the *oracle* is an encryption oracle. Loosely speaking, an oracle machine is a machine that is enhanced to interact by posing queries to the outside. In our case, these queries receive consistent responses from a function $\mathcal{A} : \{0,1\}^* \to \{0,1\}^*$, known as the oracle. In other words, if the machine poses a query q, the response it receives is $\mathcal{A}(q)$. In this scenario, we say that the oracle machine can access the oracle $\mathcal{A}$. We will call these our oracle machines $\mathcal{A}_1$ and $\mathcal{A}_2$, where $\mathcal{A}_1$ presents the behavior during the encryption requests and $\mathcal{A}_2$ presents the behavior during the additional encryption requests. During the encryption requests, $\mathcal{A}_1$ is provided with the security parameter, and the output is a pair consisting of a template (S_m, h_m, f_m) generated at the beginning of the challenge generation and some additional information D that is passed on to the second oracle machine. The second part of the adversary's strategy $\mathcal{A}_2$ is given this pair, from which it produces the actual output of the adversary. We can now rewrite the steps of the attack more simply.

1. The key K_e is generated.

2. The first oracle machine $\mathcal{A}_1^{\mathcal{E}_{K_e}}$ gets the security parameter and generates a challenge template and some additional information $(S_m, h_m, f_m), D)$.

3. A challenge is generated with respect to (S_m, h_m, f_m).

4. The second oracle machine $\mathcal{A}_2^{\mathcal{E}_{K_e}}$ uses the challenge and the additional information and outputs a result.

The adversary's objective is to guess $f(X)$, and semantic security equates to stating that the adversary's success probability corresponds to another algorithm that is only provided $h(X)$ and n. Similar to the adversary, this corresponding algorithm is divided into two parts: the first $\mathcal{A'}_1$ generates a challenge template, and the second $\mathcal{A'}_2$ solves the challenge without having access to a ciphertext, but with the information passed from the first part. This gives us the following definition of semantic security for chosen plaintext attacks in the case of a symmetric cipher.

Definition 1.7 Semantic security - chosen plaintext attacks - symmetric An encryption scheme, $(G, \mathcal{E}, \mathcal{D})$, is said to be semantically secure under chosen plaintext attacks if for every pair of polynomial-time oracle machines, $\mathcal{A}_1$ and $\mathcal{A}_2$, there exists a pair of polynomial-time algorithms, $\mathcal{A'}_1$ and $\mathcal{A'}_2$, such that the following two conditions hold:

1. every positive polynomial p, and all sufficiently large n and $z \in \{0,1\}^{\text{poly}(n)}$, it holds that

$$\Pr\left(\begin{matrix}\mathcal{A}_2^{\mathcal{E}_{K_e}}(D, (\mathcal{E}_{K_e}(X), h(X))) = f(X) \text{ where } X \in S_m(U_{\text{poly}(n)}) \text{ and where} \\ K_e \in \mathbb{F}_2^n \text{ and } \mathcal{A}_1^{\mathcal{E}_{K_e}}(n, z) = (S_m, h_m, f_m), D),\end{matrix}\right)$$

   is less than

$$\Pr\left(\begin{matrix}\mathcal{A'}_2(D, n, h(X)) = f(X) \text{ where } X \in S_m(U_{\text{poly}(n)}) \text{ and where} \\ \mathcal{A'}_1(n, z) = ((S_m, h_m, f_m), D)\end{matrix}\right) + \frac{1}{p(n)}$$

   Recall that (S_m, h_m, f_m) is the template produced in Step 3 of the above description and that X is a sample from the distribution induced by S_m.

2. For every n and z, the challenge templates are identically distributed.

Given that the challenge template is not fixed, but rather chosen independently by both $\mathcal{A}_1$ and $\mathcal{A'}_1$, it is important to require that, in both scenarios, the challenge template is distributed identically. There is no point in comparing the success probability of $\mathcal{A}_1$ and $\mathcal{A'}_1$ unless these probabilities pertain to the same distribution of problems (i.e., challenge templates).

1.2.3 Chosen ciphertext attack

We now discuss chosen ciphertext attacks, which are similar to chosen plaintext attacks but differ in that the adversary can now obtain plaintexts corresponding to ciphertexts of its choice, not the other way around. We hastily discuss two types called *a priori chosen ciphertext attacks* and *a posteriori chosen ciphertext attacks.*

In the former, decryption requests must be made before the challenge ciphertext is generated. In contrast, in the latter, decryption requests can also be made after the challenge ciphertext is generated, provided that the request does not pertain to the decryption of the same challenge ciphertext. In both instances, decryption requests can also be made for strings that are not valid ciphertexts, and in such cases, the decryption module returns some error symbol.

We look at the attack in four stages that are almost analogous to those of the chosen plaintext attack, and thus, we only discuss the differences in which they differ in two distinct ways. Firstly, in step 2, the adversary does not ask for encryptions but decryptions. Secondly, in step 4, in the case of an a posteriori chosen ciphertext attack, the adversary may submit additional decryption requests with the natural restriction that it is prohibited from requesting the decryption of the challenge ciphertext generated in step 3. A similar definition of semantic security for chosen ciphertext attacks exists but is omitted.

1.2.4 Known plaintext attack

We now look into known plaintext attacks. In this scenario, the adversary can acquire information about one or more plaintext/ciphertext pairs generated using some key in addition to some extra ciphertexts. The adversary's objective is to deduce information about the corresponding plaintext of the extra ciphertexts. This is a special case of the chosen plaintext attack where the adversary chooses the plaintexts randomly. In this setting, semantic security refers to the fact that any information that can be efficiently computed about the unknown plaintexts can also be efficiently computed solely based on the length of these plaintexts.

1.2.5 Known ciphertext attack

In a known ciphertext attack, also called a ciphertext-only attack, the adversary obtains multiple ciphertexts and tries to deduce information about the underlying plaintexts only using the ciphertexts.

1.3 Substitution permutation network and a toy cipher

In this section, we introduce the notion of a substitution permutation network and a "toy" cipher, which will be used for examples throughout this thesis. We define the generic mode of operation of any substitution permutation network. We use the definition of a substitution permutation network in [2].

Definition 1.8 Round function and Substitution Permutation Network (SPN)
Let s be the number of S-box entries and m be the number of S-boxes in each layer. Let $\mathcal{S}_1, \dots, \mathcal{S}_m$ be s-bit S-boxes. Define

$$\begin{aligned} \mathcal{S}^m : (\mathbb{F}_2^s)^m &\to (\mathbb{F}_2^s)^m \\ X = (X_1, \dots, X_m) &\mapsto (\mathcal{S}_1(X_1), \dots, \mathcal{S}_m(X_m)). \end{aligned}$$

Let π be a bit permutation, and define the i'th round-function F_i by

$$F_i(K, X) = (\pi \circ \mathcal{S}^m)(X \oplus K),$$

for any round key $K \in \mathbb{F}_2^{sm}$, and for any message $X \in \mathbb{F}_2^{sm}$. The key addition is the operation $X \mapsto X \oplus K$. The functions $\mathcal{S}^m$ and π are respectively called the substitution layer and the permutation layer of the round function F. An iterated cipher having F as a round function is called a Substitution Permutation Network.

Our toy cipher is the same as used in [11]. Here, we only present the S-box, the permutation, and the key addition. For further information, consult [11]. The S-box of our toy cipher $\mathcal{S}_{toy} : \mathbb{F}_2^4 \to \mathbb{F}_2^4$ is given in Table 1.2 in hexadecimal notation. For each round, there are four S-boxes named $\mathcal{S}_{toy1}, \mathcal{S}_{toy2}, \mathcal{S}_{toy3}$ and $\mathcal{S}_{toy4}$. Each of these S-boxes has four input bits and four output bits. We use $\mathcal{S}_{toy}$ when referring to an arbitrary S-box. The permutation of our toy cipher is given in Table 1.3 in hexadecimal notation. It takes each of the bits after the S-box has been applied and sends them to a new location. For example, suppose the output of the first S-box is 0100, and the output of the rest is all zeros. Then after using the permutation, the bits have been moved around such that only the fifth bit is non-zero, as 2 is permuted to 5. The key addition consists of an XOR between the key bits k associated with the round i, known as subkeys, and the data block input to that round. We also assume that all bits of the subkeys are independently generated and unrelated.

s	0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
$\mathcal{S}(s)$	e	4	d	1	2	f	b	8	3	a	6	c	5	9	0	7

Table 1.2: *Toy cipher S-box in hexadecimal.*

input	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
output	1	5	9	13	2	6	10	14	3	7	11	15	4	8	12	16

Table 1.3: *Toy cipher permutations.*

1.4 Entropy

In this section, we introduce the information-theoretic concept of entropy. Informally, entropy quantifies the average level of "information" or "surprise" associated with the possible outcomes of a random variable. We define it using a probability mass function $p(x)$, which can either be given or found experimentally.

Definition 1.9 Entropy
Given a discrete random variable X with values in $\mathcal{X}$ and distributed according to the probability mass function $p : \mathcal{X} \to [0, 1]$, the entropy $H(X)$ is defined as:

$$H(X) = -\sum_{x \in \mathcal{X}} p(x) \log p(x)$$

One can choose whatever logarithm-base fits their desired case, and in our case, we choose the binary logarithm $\log_2$; subsequently, whenever we refer to entropy, it is the $\log_2$ entropy. In the following example, we calculate the entropy of a 10-character password.

Example 1.10 Entropy of a password
Consider a 10-character password X drawn from a set of 62 possible characters $\mathcal{X}$ (including uppercase letters, lowercase letters, and digits). We assume that the probability mass function is uniformly distributed with $p(x) = \frac{1}{62^{10}}$ for all $x \in \mathcal{X}$ (this is rarely the case with passwords, as individuals often choose passwords based on personal information such as pet names). The entropy of this password can be calculated as follows:

$$H(X) = -\sum_{i=1}^{62^{10}} \frac{1}{62^{10}} \log_2 \frac{1}{62^{10}} \approx 59.54 \text{ bits.}$$

In practical terms, an entropy value of 59.54 bits suggests that if an attacker attempted to guess the password by brute force, they would need to try $2^{58.55}$ different combinations on average before successfully guessing the correct password.

We now calculate the entropy value assuming that $p(x)$ follows a discrete uniform distribution and that the size of $\mathcal{X}$ equals 2^t.

$$\begin{aligned} H(X) &= -\sum_{i=1}^{2^t} \frac{1}{2^t} \log_2 \frac{1}{2^t} \\ &= -\frac{2^t}{2^t} \log_2 \frac{1}{2^t} \\ &= -\log_2 \frac{1}{2^t} = t \end{aligned}$$

Chapter 2 Lightweight cryptography

In recent years, the growing demand for small, resource-constrained devices has increased the interest in lightweight cryptography. These devices, such as RFID tags, wireless sensors, and IoT devices, often have limited processing power, memory, and energy resources.
As a result, traditional cryptographic algorithms designed for general-purpose desktop and server environments may not be suitable for these devices. Lightweight cryptography focuses on developing cryptographic algorithms and protocols specifically tailored to meet the constraints of these resource-limited devices. This chapter will explore the challenges and requirements of lightweight cryptography. We draw upon [16], a report on lightweight cryptography drawn up by the National Institute of Standards and Technology (NIST), as the foundation for this chapter.

2.1 Considerations

Small, resource-constrained devices such as RFID tags, wireless sensors, and IoT devices are the target devices for lightweight cryptography. These devices are often used in application-specific cases to satisfy stringent conditions.
As an example, let us look at RFID tags that are not battery-powered. Battery-free RFID sensors can be integrated into many areas of our daily lives. These areas include but are not limited to, access control systems for security purposes, allowing for keyless entry, and, in our healthcare system, helping to monitor patients' conditions. These tags use the electromagnetic field transmitted by a reader to power their internal circuits. These devices require cryptographic algorithms with a small implementation footprint, measured in gate equivalents (GE), that also meet specific timing and power requirements. Those are only a few constraints, but the example gives a good understanding of the necessity of lightweight cryptographic algorithms. The following list is not intended to be exhaustive but merely representative of some common examples.

- **Limited Processing Power**: Constrained devices often have low computational capabilities, making it difficult to handle complex cryptographic algorithms. While 8-bit, 16-bit, and 32-bit microcontrollers are widely used, 4-bit microcontrollers are also used in specific ultra-low-cost applications.

- **Low Memory Capacity**: These devices typically have very limited RAM and storage, requiring cryptographic algorithms to efficiently use memory. In particular, the RAM of some microcontrollers goes all the way down to 16 bytes.

- **Energy Availability**: As depicted earlier, some devices rely on "transferable energy." These devices have a maximum energy consumption for each interaction, which must be considered when designing the cryptographic methods. More importantly, though, are the devices that rely on battery power. Replacing the batteries once deployed might be difficult or even impossible in some applications. This necessitates cryptographic methods that consume minimal energy.

- **Size**: Physical size limitations can restrict the hardware available for cryptographic functions. The area is measured in μm^2 and is often stated in GEs. For some application-specific implementations, one GE is defined as the area required by the two input NAND gate (the opposite of an AND gate). The number of GEs is obtained by dividing the area in μm^2 by the area of the NAND gate. This makes it impossible to compare the number of GEs across different technologies. Some low-cost RFID tags might have a total of 1.000-10.000 GE, where only 20% may be used for security purposes.

- **Latency and Throughput**: Latency is the measure of time between the initial request of an operation and producing the output. Throughput is the speed at which the encryption algorithm is able to consume input and generate output. Some applications require fast encryption and decryption processes to maintain performance. For instance, components like steering, airbags, or brakes in automotive applications require extremely fast response times and the processing of large amounts of data.
- **Interoperability Needs**: Cryptographic solutions must be compatible with other devices and systems. While lightweight cryptography primarily concerns resource-constrained devices, it is important to note that these algorithms might also need to be implemented on devices such as smartphones and desktops. These devices often are on the receiving end of the information sent by resource-constrained devices.
- **Scalability Requirements**: Solutions need to be scalable to support a large number of devices without significant performance degradation. It is insufficient to make a solution that works well on specific devices but not on others.
- **Security Requirements**: Despite the constraints, the cryptographic algorithm must still provide adequate security against various threats and attacks. The minimum key size required is 128 bits.
- **Flexibility**: It's desirable to have tunable algorithms that use parameters to select properties like state size and key size, as they can support multiple options using fewer resources, thus enabling a wider range of applications.

2.2 Requirements

Since the report's publication, NIST has published a "call for algorithms"[20] that describes the requirements, selection process, and evaluation criteria. More than 50 submissions were considered during three rounds, and finally, in February 2023, a finalist was selected. The following only considers Authenticated Encryption with Associated Data (AEAD) algorithms and presents some of these requirements. Many of the terms will not be explained here, but they will be clarified later.

An AEAD algorithm takes four byte-string inputs and two byte-string outputs. The four inputs are a variable-length plaintext, variable-length associated data, a fixed-length nonce, and a fixed-length key, and the outputs are a variable-length ciphertext and a fixed-length tag. All byte-string inputs that satisfy the input length requirements must be accepted. The nonce must have a length of at least 96 bits, and the tag must have a length of at least 64 bits.

Authenticated decryption must also be supported. If associated data, nonce, and key are given, it must be possible to decrypt a ciphertext into its corresponding plaintext. The decryption-verification process shall not return plaintext if the ciphertext is invalid.

AEAD algorithms must guarantee the confidentiality of the plaintexts and the integrity of the ciphertexts. AEAD algorithms are expected to be secure as long as the nonce is not repeated under the same key. Key lengths are not allowed to be smaller than 128 bits. Cryptanalytic attacks on the AEAD algorithm must require at least 2^{112} computations in a single key setting.

Chapter 3 The Ascon Cipher

This chapter introduces the Ascon v1.2 cipher suite as submitted to NIST. We look into its phases and the encryption and decryption method, analyzing and commenting on them.

3.1 Ascon

We chose to study Ascon for various reasons, but most importantly, it has gathered much attention by being selected as the primary choice for lightweight applications in the final portfolio of the "CAESAR" competition ([19]) and being chosen for standardization by the NIST Lightweight Cryptography Team in the "Lightweight Cryptography Standardization Process" competition ([6]).

This section and subsequent subsections draw inspiration from [8]. The Ascon cipher suite consists of many schemes for authenticated encryption with associated data (AEAD) and hashing functionality. With regards to authenticated ciphers, the suite consists of Ascon -128, Ascon -128a, and Ascon -80pq, and with respect to hashing functionality, the suite consists of Ascon -HASH, Ascon -HASHA, Ascon -XOF and Ascon -XOFA. We choose to focus on the AEAD cipher Ascon -128. Common to all schemes is the underlying transformation defined on five 64-bit words only using the bitwise Boolean functions "AND, NOT, XOR" and rotations within the words. We now create an overview of Ascon , then we discuss the phases of Ascon , and finally, we discuss the permutations during the intermediate processes.

3.1.1 Overview: Ascon's authenticated encryption schemes

The authenticated encryption with associated data schemes are parameterized by the key length $0 \leq k \leq 160$ bits, the rate (data block size) $0 \leq r \leq 255$, and the internal round numbers for the transformations $1 \leq a, b \leq 255$ where $a \geq b$. Each of the schemes defines an underlying encryption- and decryption algorithm respectively denoted $\mathcal{E}_{k,r,a,b}$ and $\mathcal{D}_{k,r,a,b}$. As input, the authenticated encryption procedure $\mathcal{E}_{k,r,a,b}$ takes the secret key $K \in \mathbb{F}_2^k$, a nonce (public message number) $N \in \mathbb{F}_2^{128}$, an associated data vector A of arbitrary length over $\mathbb{F}_2$ and a plaintext vector P also of arbitrary length over $\mathbb{F}_2$. It produces an output consisting of a ciphertext vector C of the same length as the plaintext vector P together with a tag $T \in \mathbb{F}_2^{\min\{128,k\}}$, which is used for authentication of both the associated data and the encrypted message C:

$$\mathcal{E}_{k,r,a,b}(K, N, A, P) = (C, T).$$

Remark 3.1
It is important not to reuse the same nonce N for multiple encryptions under the same key. If this happens, the security of the schemes cannot be assured. We comment more on this later.

The decryption and verification procedure $\mathcal{D}_{k,r,a,b}$ takes the same secret key K, nonce N and associated data A as the encryption procedure, together with the ciphertext vector C and the produced tag T from the encryption procedure as input. It outputs either the plaintext vector P if the tag is verified or $\perp$ if the tag verification fails:

$$\mathcal{D}_{k,r,a,b}(K, N, A, C, T) \in \{P, \perp\}.$$

Many size parameters depend on the choice k of key size. It is recommended that the key size be at least 128 bits long, the rate between 64 and 128, and the round numbers a and b between 6 and 16. In table 3.1 the size parameters for Ascon is summed up.

Name	Size parameters	Guideline size parameters
Key	$0 \leq k \leq 160$	$128 \leq k \leq 160$
Initialization vector	$32 \leq \lvert IV_{k,r,a,b}\rvert \leq 192 - k$	$32 \leq \lvert IV_{k,r,a,b}\rvert \leq 192 - k$
Tag	$\min\{k, 128\}$	128
Data block/rate	$0 \leq r \leq 255$	$64 \leq r \leq 128$
Round numbers a and b	$0 \leq a, b \leq 255$	$6 \leq a, b \leq 16$
Nonce	128	128

Table 3.1: *The allowed and guideline sizes of parameters for the different variables in the AEAD* Ascon *cipher scheme.*

Common to Ascon -80pq, Ascon -128 and Ascon -128a are the fixed nonce N and tag T sizes each of 128 bits, and common to both Ascon -128 and Ascon -128a are the key size k of 128 bits. Ascon -128 and Ascon -128a differ in the size of the data blocks r and the round numbers a and b, where Ascon -128 uses 64 bits for the size of the data blocks and $a = 12$, $b = 6$ for the round numbers, while Ascon -128a uses 128 bits for the size of the data blocks and $a = 12$, $b = 8$ for the round numbers. Ascon -80pq has an increased key size of 160 bits, but the rest of the parameters are equal to that of Ascon -128. This is summed up in Table 3.2 We state the recommended parameters of Ascon , which were found experimentally to suit the cipher well.

Name	key	nonce	tag	data block	a	b
Ascon -128	128	128	128	64	12	6
Ascon -128a	128	128	128	128	12	8
Ascon -80pq	160	128	128	64	12	6

Table 3.2: *The proposed size parameters for the* Ascon *AEAD cipher suite.*

We now offer insight into the state $S \in \mathbb{F}_2^{320}$, which is the part of the cryptosystem that is transformed during encryption and decryption. The state can be updated in two ways: transforming the existing state using the predetermined transformations τ^i or adding additional data consisting of either plaintext/ciphertext or associated data to the first r entries. When transforming the state using τ^i, we do not repeat the transformation τ i times, but instead, we repeat slightly different transformations a number of times a or b dependent on which phase is currently being processed. We make it clear that

$$\tau^i \neq \underbrace{\tau \circ \ldots \circ \tau}_{i \text{ times}}.$$

We will discuss this later and give a concrete definition when explaining the permutations. We call the integers a and b the round numbers of the algorithm, where a represents the number of rounds in the initialization and finalization phases, and b represents the number of rounds of the intermediate rounds processing associated data, plaintext, and ciphertext.

The recommended values of these integers are guidelines, and in any real-world scenario, one can choose these values secretly.

We introduce notations needed to describe the Ascon cipher suite.

Definition 3.2 Most and least significant entries
Let D be a vector over $\mathbb{F}_2$. We call the first k entries of D the most significant k entries of D and denote D truncated to its k most significant entries by:

$$\lfloor D \rfloor_\kappa .$$

Likewise we call the last k entries of D the least significant k entries of D and denote D truncated to its k least significant entries by:

$$\lceil D \rceil^\kappa .$$

When discussing any vector over $\mathbb{F}_2$, we often call it a bit-string and refer to its entries as bits. Also, 8 bits is called a byte, so the first 8 entries of a vector $D \in \mathbb{F}_2^n$ for $n \geq 8$ would be called the first 8 bits of D or simply the first byte of D. For simplicity, we often refer to the "XOR" operation over $\mathbb{F}_2^n$ as addition.

The state is split into two parts $\lfloor S \rfloor_r$ consisting of the first r bits of S and $\lceil S \rceil^c$ consisting of the last $c = 320 - r$ bits of S. The rate r varies depending on the Ascon variant. If $r = 64$ and the state has not been altered, then $\lfloor S \rfloor_r$ corresponds to the initialization vector $IV \in \mathbb{F}_2^r$, and $\lceil S \rceil^c \in \mathbb{F}_2^c$ is equivalent to the secret key K concatenated with the nonce N. When examining the permutations, it becomes convenient to split the state into five 64 bit words:

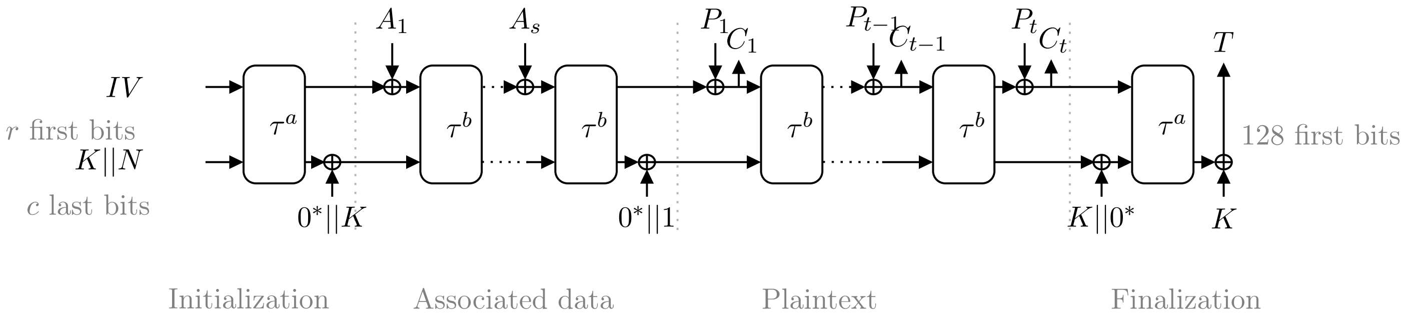
$$S = \lfloor S \rfloor_r || \lceil S \rceil^c = s_1 || s_2 || s_3 || s_4 || s_5 .$$

The symbol || is called *concatenation* and refers to joining two or more numbers to create a new, longer number. As r often equals 64 we shall sometimes refer to $\lfloor S \rfloor_r$ as s_1 and $\lceil S \rceil^c$ as $s_2 || s_3 || s_4 || s_5$

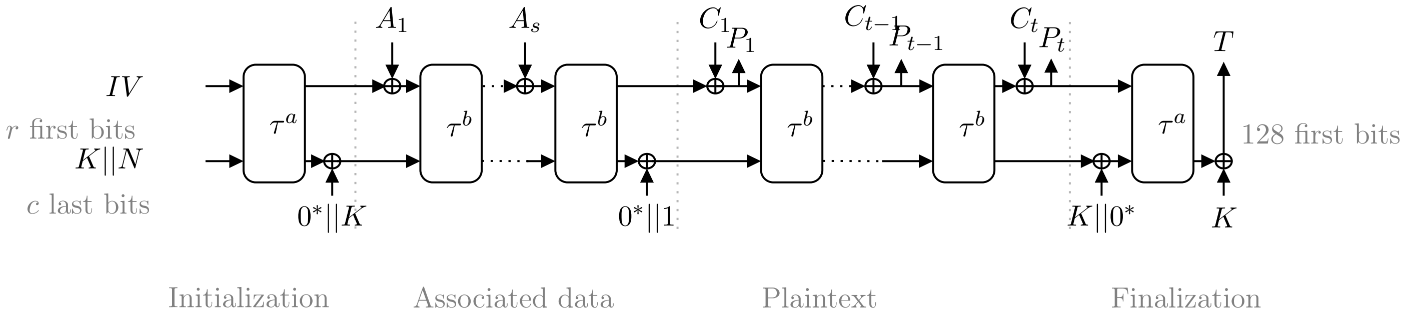
The Ascon encryption and decryption process each consists of four distinct phases: the initialization phase, the associated data phase, the plaintext/ciphertext phase, and the finalization phase. In each of these, the scheme adds data to the current state and transforms it, thus altering the state and making it computationally expensive for an adversary to backtrack changes to get previous states. Another integral part of the Ascon schemes is the *duplex-like* construction ([4]), where data can both be injected into the state and extracted from the state during the intermediate steps.

3.1.2 The phases of Ascon

We now describe the four phases of the AEAD Ascon cipher suite in depth. Below are the mode of operation of Ascon .



Encryption



Decryption

Initialization
The 320 bit initial state consists of a secret key K of k bits, nonce of 128 bits, and an initialization vector $IV \in \mathbb{F}_2^{192-k}$. The exact value of the initialization vector is given as follows:

$$IV_{k,r,a,b} = k||r||a||b||0^{160-k}.$$

The key size k, the rate r, and the round numbers a and b are all written as 8 bit binary numbers, and the exponent of 0 in the above expression is the number of zeros. The initial state is defined as

$$S \leftarrow IV_{k,r,a,b}||K||N.$$

We use the $\leftarrow$ to denote when a variable is updated. This becomes convenient as we often update the state, and alternative notation would become cumbersome.
During the initialization phase, the state S is permuted a times using the permutation τ followed by an addition of the secret key K to the least significant k bits of the state:

$$S \leftarrow \tau^a(S) \oplus (0^{320-k}||K).$$

We add the key to the state twice, once during the initialization process and once during the finalization process. This ensures that adversaries cannot forge messages, which will be made clear in Theorem 5.8 introduced later. The part of the state to which the key is added changes for which the reason will be clarified later. After the initialization phase, one can view the state as split into two parts: the first r bits and the last c bits.

Associated data
Associated data is non-confidential data added to a cipher scheme. It is data that must

be presented to both the encrypter and the decryptor. If the decryptor is presented with the wrong data, the decryption fails. It is useful for "encryption context"; it can include metadata, timestamps, identifiers, and other information that is not part of the primary data but is essential for proper interpretation.

When associated data A is introduced to the scheme, it is padded and divided into blocks of size r bits. We always pad the associated data with a single one and enough zeros to make the padded length a multiple of r. Suppose associated data of length $|A|$, where $|A|$ is divisible by r, is left unpadded. In that case, we might confuse it with another piece of associated data string A' whose length is not divisible by r but is identical to A in all but the least significant ℓ entries.

Example 3.3
Say that $|A| = nr$ and $A = a_1 \dots a_{nr-3}100$. If left unpadded, we would not be able to distinguish between A and another piece of associated data A', whose length is $nr - 3$ and whose entries are identical to A in the first $nr - 3$ indices, that is $\lfloor A \rfloor_{nr-3} = A' = a_1 \dots a_{nr-3}$, as we would have to pad A' as its length is not divisible by r.

By always padding the associated data we know to remove everything after encountering the last 1.

Definition 3.4 Padded data
Let D be any binary string, then if D is padded with a single one and κ many zeros, it is denoted by

$$D||1||0^{\kappa}.$$

The resulting division gives us s blocks $A_1, \dots, A_s$ each of length r. In the case when there is no associated data, no padding is needed, and we let $s = 0$:

$$A||1||0^{\kappa} = A_1, \dots, A_s = \begin{cases} r\text{-bit blocks of } A||1||0^{r-1-(|A| \mod r)} & \text{if } |A| > 0 \\ \varnothing & \text{if } |A| = 0. \end{cases}$$

We add each block A_i for $i = 1, \dots, s$ to the most significant r bits $\lfloor S \rfloor_r$ of the state S using the function σ_r, which is given as follows:

$$\sigma_r(D) : \mathbb{F}_2^{320} \times \mathbb{F}_2^{r} \to \mathbb{F}_2^{320}, \tag{3.1}$$

$$(S, D) \mapsto (\lfloor S \rfloor_r \oplus D)||\lceil S \rceil^{320-r}, \tag{3.2}$$

where D is some r-dimensional data and S is the most recent updated state. The function σ_r adds data D to the most significant r bits of the state and concatenates the remaining c bits, thus updating the state. The application of σ_r is followed by an application of the transformation τ^b, and we obtain the new state:

$$S \leftarrow \tau^b(\sigma_r(A_i)), \quad 1 \leq i \leq s.$$

After processing all of the associated data, we add a 1-bit domain separation constant to the state:

$$S \leftarrow S \oplus (0^{319}||1).$$

This is done to prevent attacks that switch the role of plaintext and associated data blocks ([12]).

Plaintext/ciphertext
The ASCON AEAD cipher suite processes plaintext of any length during the encryption process, and it processes ciphertext during the decryption process. When plaintext P is introduced to the scheme, it is padded and divided into blocks of r bits using the same padding rule used for associated data. The resulting division gives us t blocks $P_1, \ldots, P_t$ each of length r:

$$P||1||0^{r-1-(|P| \bmod(r))} = P_1|| \ldots ||P_t.$$

Encryption. The encryption process consists of multiple iterations. During each iteration, we add a single padded plaintext block P_i, for $i = 1, \ldots, t$, to the most significant r bits of the state $\lfloor S \rfloor_r$ followed by an extraction of a ciphertext block C_i. For each iteration, except the last, the state S is transformed using the transformation τ^b:

$$\begin{aligned} C_i &\leftarrow \lfloor \sigma_r(P_i) \rfloor_r \\ S &\leftarrow \begin{cases} \tau^b(\sigma_r(P_i)) & \text{if } 1 \leq i < t \\ \sigma_r(P_i) & \text{if } i = t. \end{cases} \end{aligned}$$

When extracting the last ciphertext block C_t, we truncate it to the length of the last unpadded plaintext block-fragment, such that its length is between 1 and $r - 1$ bits, and such that the total length of the ciphertext $C = C_1|| \ldots ||\tilde{C}_t$ is exactly the same as original plaintext message P:

$$\tilde{C}_t = \lfloor C_t \rfloor_{|P| \bmod r}.$$

Decryption. Like the encryption process, the decryption process also consists of multiple iterations. During each iteration, a single padded plaintext block P_i is computed by adding the most significant r bits of the state $\lfloor S \rfloor_r$ with the ciphertext C_i:

$$\lfloor \sigma_r(P_i) \rfloor_r = C_i \Leftrightarrow \lfloor \sigma_r(C_i) \rfloor_r = P_i.$$

This works since the initialization and associated data process of both the encryption function $\mathcal{E}_{k,r,a,b}$ and the decryption function $\mathcal{D}_{k,r,a,b}$ are equivalent and since $a \oplus b = c \Leftrightarrow c \oplus a = b$ as shown in the following proposition.

Proposition 3.5
Let $a, b, c \in \mathbb{F}_2^n$, then

$$a \oplus b = c \Leftrightarrow c \oplus a = b.$$

Proof. Assume that $a \oplus b = c$. We add a on both sides and using that each element is its own inverse, $a \oplus a = 0$, the commutativity and associativity of $\oplus$ we get

$$a \oplus b = c \Leftrightarrow a \oplus a \oplus b = a \oplus c \Leftrightarrow b = a \oplus c.$$

□

In each iteration, except the last, after recovering the plaintext, we replace the most significant r bits of S by C_i

$$S \leftarrow C_i || \lceil S \rceil^c \text{ for } 1 \leq i < t.$$

Moreover, for each ciphertext block except the last, the state is transformed using the transformation τ^b:

$$S \leftarrow \tau^b(S).$$

For the last truncated ciphertext block $\tilde{C_t}$ with $0 \leq \ell < r$ bits, the process differs, as we now only add the most significant ℓ bits of the state with the last ciphertext block to obtain the last plaintext block

$$\tilde{P_t} = \lfloor S \rfloor_\ell \oplus \tilde{C_t}.$$

After getting back the last plaintext block $\tilde{P_t}$, we need to make sure the last ciphertext block $\tilde{C_t}$ is included in the state, just like the other ciphertexts have been in the previous iterations. As the length of $\tilde{C_t}$ is not r, we alter the state slightly relative to before:

$$S \leftarrow \sigma_r(\tilde{P_t}||1||0^{r-1-\ell}))$$

where $\lfloor \sigma_r(\tilde{P_t}||1||0^{r-1-\ell})) \rfloor_r$ is the padded ciphertext of length r. The term $\tilde{P_t}||1||0^{r-1-\ell}$ can be rewritten as $\tilde{P_t}||0^{r-\ell} \oplus 0^\ell||1||0^{r-\ell-1}$, and when applying σ to both terms we include the unpadded ciphertext $\tilde{C_t} = \lfloor \sigma_\ell(\tilde{P_t}) \rfloor_\ell$:

$$\begin{aligned} &= \sigma_\ell(\tilde{P_t}) \oplus \sigma_r(0^\ell||1||0^{r-1-\ell}) \oplus \lceil S \rceil^c \\ &= \tilde{C_t}||\lceil \sigma_r(0^\ell||1||0^{r-1-\ell}) \rceil^{c-\ell} \end{aligned}$$

For the first equality to hold $\lceil S \rceil^c$ is added one additional time to combat the cancellation of it.

Finalization. During the finalization phase, we add the key to the state in the following way:

$$S \leftarrow S \oplus 0^r||K||0^{c-k}.$$

This ensures that the key additions do not cancel each other out. This could happen in the rare case when there is no associated data and only one incomplete plaintext block of length $1 \leq \ell < r$. In this case, no transformations between the initialization and the finalization phase would occur, and if we added the key to the same part of the state, they would cancel each other out, as they are their own inverse. After adding the key, the state is transformed using the permutation τ^a:

$$S \leftarrow \tau^a(S).$$

The finalization phase also produces the verification tag T, which consists of the addition of the last 128 bits of the state and the last 128 bits of the key:

$$T = \lceil S \rceil^{128} \oplus \lceil K \rceil^{128}.$$

When the guideline size of the key of at least 128 bits is not followed, one can use a truncated tag of length k. During encryption $\mathcal{E}_{k,r,a,b}$, the finalization phase always returns the tag T and the ciphertext C, but during decryption $\mathcal{D}_{k,r,a,b}$, the finalization phase only returns the plaintext P if the calculated tag value matches the received tag value. If the decryptor receives an errorful ciphertext, it computes a different tag than the tag received, and thus, the plaintext is not extracted. This is done to ensure the messages' authenticity, as an adversary could have interfered and sent the errorful ciphertext.

3.1.3 Permutation

We now look into the permutation of the Ascon cipher suite.The permutation applies a substitution permutation-based round-like transformation τ repeated a number of times. It consists of three underlying steps: an addition of constants τ_{c_i} that changes each round, a substitution layer $\tau_{\mathcal{S}}$, and a linear diffusion layer τ_{π}. Round $i+1$ takes the state of round i as input

$$\tau^a = \bigcirc_{i=1}^{a} \tau_{\pi} \circ \tau_{\mathcal{S}} \circ \tau_{c_i}.$$

The substitution permutation-based round-like transformation in Ascon differs from a substitution permutation-based round transformation by the fact that it does not involve a key schedule (an algorithm that determines the round keys using the original key); instead, it uses predetermined fixed round key values. The number parameters a and b can be tuned, and larger values offer greater security but also higher computational costs.

As mentioned earlier, it is favorable to express the state S as five 64-bit register vectors to describe the underlying steps. One can imagine this as a 5×64 matrix, where each of the entries consists of a single bit:

$$S = s_1 || s_2 || s_3 || s_4 || s_5 = \begin{pmatrix} s_1 \\ s_2 \\ s_3 \\ s_4 \\ s_5 \end{pmatrix}.$$

Hexadecimal notation becomes convenient for conserving space and facilitating a human-readable representation of binary values. Therefore, we introduced it. Hexadecimal notation is a positional numeral system used for base-16 numbers. In this system, each digit represents a four-bit binary sequence. Hexadecimal notation employs the digits 0-9 and the letters a-f to symbolize values from 0 to 15, providing a compact representation of binary data. We introduce the notation in Table 3.3.

$\mathbb{N}$	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	...	26
$\mathbb{F}_{16}$	0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f	10	11	...	1a

Table 3.3: *Conversion table between integers and hexadecimal.*

We now go into detail with each of the three underlying steps.

Addition of constants

The constant addition step $\tau_{c_i} : \mathbb{F}_2^{320} \to \mathbb{F}_2^{320}$ adds a round constant value (round key) c_i to the third register vector s_3 of the state in round i. One can think of this as the matrix

$$\begin{pmatrix} s_1 \\ s_2 \\ s_3 \\ s_4 \\ s_5 \end{pmatrix} \oplus \begin{pmatrix} 0 \\ 0 \\ c_i \\ 0 \\ 0 \end{pmatrix},$$

where the XOR of matrices works entrywise. The round constants c_i can be found in Table 3.4 and are given in hexadecimal notation to conserve space (each constant has length 64 in binary, the first 56 being all zeros). While unconventional, we choose to write the first 14 zeros of the hexadecimal number as $0^{14}||$. Again, this is done to conserve space. The values of the round constants are chosen such that there is an increasing and decreasing counter for the two halves of the affected 8 bits. This can easily be extended to 16 rounds if a higher security margin is needed, but such constants are not specified for round numbers exceeding 16.

Round numbers for different values of a or b

$a \vee b$=12	$a \vee b$ =8	$a \vee b$ =6		$a \vee b$ =12	$a \vee b$ =8	$a \vee b$ =6	
	Round numbers		constant c_i		Round numbers		constant c_i
1			0^{14}||f0	7	3	1	0^{14}||96
2			0^{14}||e1	8	4	2	0^{14}||87
3			0^{14}||d2	9	5	3	0^{14}||78
4			0^{14}||c3	10	6	4	0^{14}||69
5	1		0^{14}||b4	11	7	5	0^{14}||5a
6	2		0^{14}||a5	12	8	6	0^{14}||4b

Table 3.4: *The round constants used in each round i of τ^a and τ^b.*

We use the following rule to specify the round constant c_i used in round i of the transformations τ^a and τ^b

$$\begin{aligned} c_i &= c_i && \text{for } \tau^a \\ c_i &= c_{i+a-b} && \text{for } \tau^b. \end{aligned}$$

The constant is added to the third register vector, as whenever $r \leq 128$, the addition can be done in parallel with the addition of associated data, plaintext, or ciphertext to the state. Let $D \in \mathbb{F}_2^r$ be any data added to the state, then

$$\begin{aligned} S \leftarrow &(D \oplus \lfloor S \rfloor_r) || \lceil \lfloor S \rfloor_{128} \rceil^{128-r} || (c_i \oplus \lfloor \lceil S \rceil^{192} \rfloor_{64}) || \lceil S \rceil^{128} \\ = &\lfloor \sigma_r(D) \rfloor_{128} || (c_i \oplus s_3) || s_4 || s_5. \end{aligned}$$

Notice that although we in Section 3.1.2 explain the addition of data and the addition of round constants as two distinct steps, there is no mathematical need to do these steps separately. Hence, we do them in parallel whenever $r \leq 128$. When $r > 128$, the steps can not be done in parallel, as we would simultaneously add multiple values to the same entry.

Substitution layer

We start by introducing the notion of an S-box.

Definition 3.6 S-box
Let $m, n \in \mathbb{N}$. An S-box is a function

$$\mathcal{S} : \mathbb{F}_2^n \to \mathbb{F}_2^m$$

that performs substitutions.

An $n \times m$ S-box can be implemented as a lookup table. The S-box $\mathcal{S}_{\text{Ascon}} : \mathbb{F}_2^5 \to \mathbb{F}_2^5$ updates the columns s^j for $1 \leq j \leq 64$ of the state S with 64 simultaneous applications of the 5-bit S-box $\mathcal{S}_{\text{Ascon}}(s^j)$ as defined in Table 3.5.

$$\mathcal{S}_{\text{Ascon}} : \mathbb{F}_2^5 \to \mathbb{F}_2^5.$$

The substitution layer $\tau_{\mathcal{S}}$ is a function that applies 64 simultaneous applications of the 5-bit S-box $\mathcal{S}_{\text{Ascon}}$.

$$\tau_{\mathcal{S}_{\text{Ascon}}} : (\mathbb{F}_2^5)^{64} \to (\mathbb{F}_2^5)^{64}$$

Each application of the 5-bit S-box conceptually works by viewing each column s^j of the 5×64 "matrix" as a 5-bit string, then converting it to its corresponding value in hexadecimal, finding its value $\mathcal{S}_{\text{Ascon}}(s^j)$ in the lookup table Table 3.5, and then converting it to binary replacing the column with the new column:

$$s^j \leftarrow [\mathcal{S}_{\text{Ascon}}([s^j]_{\text{bin}\to\text{hex}})]_{\text{hex}\to\text{bin}},$$

for $1 \leq j \leq 64$, and where the functions $[s]_{\text{bin}\to\text{hex}}$ and $[s]_{\text{hex}\to\text{bin}}$ respectively refers to the transformation from binary numbers to hexadecimal and from hexadecimal to binary numbers. In reality, we would not perform these translations; instead, we would translate the lookup table once, but that would make the table more difficult to read. To illustrate how this works, we have the following example.

Example 3.7
We use the same S-box as for the Ascon cipher suite on the following bit-string of length 30:

$$100100011011010100011110101100.$$

Splitting this into five words of length six yields:

$$100100||011011||010100||011110||101100,$$

and rewritten to look like the state of Ascon when describing the transformations (matrix form):

$$\left(s^1, s^2, s^3, s^4, s^5, s^6\right) = \begin{pmatrix} 100100 \\ 011011 \\ 010100 \\ 011110 \\ 101100 \end{pmatrix}.$$

We now read the values vertically, such that the first value is 10001, which translates to 11 in hexadecimal. We now find the s value 11 in the S-box lookup table and see that it becomes 13, which translates to 10011 in binary. We do this for all the values and get:

$$\left(s^1, s^2, s^3, s^4, s^5, s^6\right) \leftarrow \begin{pmatrix} 101101 \\ 000111 \\ 010000 \\ 111001 \\ 100001 \end{pmatrix} = \left[\mathcal{S}_{\text{ASCON}}\left(\left[\left(s^j\right)\right]_{\text{bin}\to\text{hex}}\right)\right]_{\text{hex}\to\text{bin}}, \text{ for } 1 \leq j \leq 6$$

which is the updated state after using the S-box.

s	0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f	10	11	12	13	14	15	16	17	18	19	1a	1b	1c	1d	1e	1f
$\mathcal{S}_{\text{ASCON}}(s)$	4	b	1f	14	1a	15	9	2	1b	5	8	12	1d	3	6	1c	1e	13	7	e	0	d	11	18	10	c	1	19	16	a	f	17

Table 3.5: *ASCON's 5-bit S-box $\mathcal{S}$ as a lookup table. The values of s and $\mathcal{S}(s)$ should be translated to binary, but they have been written as hexadecimal to conserve space.*

One can also illustrate the S-box as the diagram in Figure 3.1 or algebraically in the following equations:

$$\left.\begin{aligned} y_{0,i} &= s_{5,i}s_{2,i} \oplus s_{4,i} \oplus s_{3,i}s_{2,i} \oplus s_{3,i} \oplus s_{2,i}s_{1,i} \oplus s_{2,i} \oplus s_{1,i} \\ y_{1,i} &= s_{5,i} \oplus s_{4,i}s_{3,i} \oplus s_{4,i}s_{2,i} \oplus s_{4,i} \oplus s_{3,i}s_{2,i} \oplus s_{3,i} \oplus s_{2,i} \oplus s_{1,i} \\ y_{2,i} &= s_{5,i}s_{4,i} \oplus s_{5,i} \oplus s_{3,i} \oplus s_{2,i} \oplus 1 \\ y_{3,i} &= s_{5,i}s_{1,i} \oplus s_{5,i} \oplus s_{4,i}s_{1,i} \oplus s_{4,i} \oplus s_{3,i} \oplus s_{2,i} \oplus s_{1,i} \\ y_{4,i} &= s_{5,i}s_{2,i} \oplus s_{5,i} \oplus s_{4,i} \oplus s_{2,i}s_{1,i} \oplus s_{2,i}, \end{aligned}\right\} \text{for all } 1 \leq i \leq 64.$$

This layer is a non-linear part of the round transformation.

Proposition 3.8
The ASCON S-box is neither linear nor affine.

Proof. The S-box cannot be linear since $\mathcal{S}_{\text{ASCON}}(0) = 4 \neq 0$, further it cannot be affine as $4 = \mathcal{S}_{\text{ASCON}}(0) \neq \mathcal{S}_{\text{ASCON}}(0) + \mathcal{S}_{\text{ASCON}}(0) = 8$ □

The S-box is also designed to fulfill important properties, such as invertibility, containing no fix-points, and each output depending on at least four input bits. The containment of no fix-points and the dependence on at least four input bits are clear, so we only need to argue why it is invertible. We notice that $S_{\text{ASCON}} : \mathbb{F}_{32} \to \mathbb{F}_{32}$ and every s is uniquely mapped to

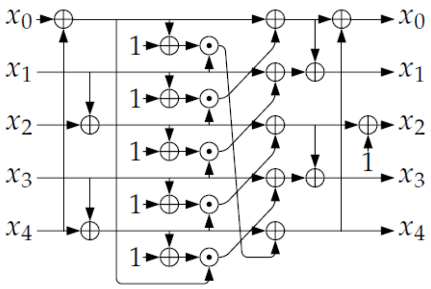


Figure 3.1: *An illustration of the* Ascon *S-box using logic gates XOR and AND.*

an element of $\mathbb{F}_{32}$ and therefore it is both injective and surjective and thereby bijective.

Linear diffusion layer/permutation layer
The linear diffusion layer $\pi : (\mathbb{F}_2^{64})^5 \to (\mathbb{F}_2^{64})^5$ adds different rotated copies of each register word to themselves. It provides this rotation by applying functions $\pi_i(s_i)$ to each register word. These functions differ for each register word and have been chosen experimentally to achieve a good diffusion within each register word. They are defined entrywise for each register word as follows:

$$\left.\begin{aligned}
\pi_0(s_1) = y_{0,i} &\quad = s_{1,i} \oplus s_{1,i+19} \oplus s_{1,i+28} \\
\pi_1(s_2) = y_{1,i} &\quad = s_{2,i} \oplus s_{2,i+61} \oplus s_{2,i+39} \\
\pi_2(s_3) = y_{2,i} &\quad = s_{3,i} \oplus s_{3,i+1} \oplus s_{3,i+6} \\
\pi_3(s_4) = y_{3,i} &\quad = s_{4,i} \oplus s_{4,i+10} \oplus s_{4,i+17} \\
\pi_4(s_5) = y_{4,i} &\quad = s_{5,i} \oplus s_{5,i+7} \oplus s_{5,i+41}
\end{aligned}\right\} \text{ for all } 1 \leq i \leq 64$$

The functions π_i are found to be linear by application of Proposition 3.5. Table 3.6 shows some diffusion properties of up to 3 rounds of the Ascon permutation. After three rounds, almost all input bits appear in the algebraic equations of each output bit due to the mixing.

round	1		2		3	
	$\tau_{\mathcal{S}}$	τ_{π}	$\tau_{\mathcal{S}}$	τ_{π}	$\tau_{\mathcal{S}}$	τ_{π}
$s_{1,i}$	5	15	51	125	219	313
$s_{2,i}$	5	15	51	115	219	308
$s_{3,i}$	4	12	41	107	218	316
$s_{4,i}$	5	15	51	130	219	305
$s_{5,i}$	4	12	43	107	193	306

Table 3.6: *The number of different variables in algebraic equation of* $s_{w,i}$

3.2 Claimed security properties of Ascon

This section comments on some of the claimed security properties of Ascon . We use [8, Chapter 3] as a reference point and focus on Ascon -128.

The Ascon cipher suite provides 128−bit security for nonce-based AEAD. That is, the

confidentiality of the plaintext, besides its length, and the integrity of ciphertext, including the associated data, is protected and has a complexity of 2^{128}. Users should add extra padding if the plaintext length needs to remain confidential. The encrypter and the decrypter should agree on this padding.
To fulfill the security claim, we require implementations to ensure that nonce is not repeated for two different encryptions under the same key. Also, we require decrypted plaintexts plaintexts only to be released after successful tag verification.
We also limit the amount of processed plaintext and associated data blocks protected by the encryption algorithm to 2^{64} blocks per key. As a data block has size 64 and one byte consists of 8 bits, this corresponds to a total of 2^{67} bytes for Ascon -128 and Ascon -80pq and a total of 2^{68} bytes for Ascon -128a (see table 3.2). The authors consider this as more than sufficient for lightweight applications in practice. We summarize these and other claims in table 3.7.

Requirement	Security in bits		
	Ascon -128	Ascon -128a	Ascon -80pq
Confidentiality of plaintext	128	128	128
Integrity of plaintext	128	128	128
Integrity of associated data	128	128	128
Integrity of public message number	128	128	128
Maximum number of bytes encrypted	2^{67}	2^{68}	2^{67}
Key recovery and forgery complexity in the case of a leaked inner state	2^{128}	2^{96}	2^{128}

Table 3.7: *Security claims for recommended parameter configurations of* Ascon .

It is also claimed that even if specific accidental implementation errors, such as the nonce being repeated a few times, the security claims stated in table 3.7 can still be fulfilled. Here, we only require the combination of nonce and associated data to be unique. Also, even if a single inner state is leaked during data processing, this does NOT imply the possibility of a global attack, such as key recovery or trivial forgeries. In the case of a leaked state, forgeries and key recoveries can be obtained with a complexity of $2^{c/2}$.

"*We do not expect that key recovery attacks for* Ascon *-128a and* Ascon *-128 can be found with complexity significantly below* 2^{96} *and* 2^{128} *even if a few internal states can be recovered. In fact, it is easy to see that the product of data and time complexity for a key-recovery attack remains above* 2^{128}*.*"[8, page 15-16]

Apart from the single-use requirement, there are no further constraints on the nonce; one may even choose a practical and simple counter.

When implementing the algorithm, one should also consider implementing a procedure that monitors and counts the number of failed verification attempts and, if necessary, limits this number such that if exceeded, the decryption algorithm rejects all tags, including the correct one.

3.3 Comparison of AES-128-GCM and Ascon -128

This section compares Ascon -128 to the current encryption standard, Advanced Encryption Standard (AES). We compare Ascon-128 to AES-128 because they have the same key size. We will not explain AES-128's mode of operation; instead, we will compare the parameters

and metrics. For many years, AES has been the dominant algorithm for symmetric encryptions. AES can be fast but also costly in terms of memory and implementation area. Secure communication involves more than just encryption; it also requires ensuring authenticity and confidentiality. Thus, comparing Ascon to AES directly would be like comparing apples to oranges. Therefore, we transform AES into an AEAD cipher utilizing the Galois/Counter Mode (GCM) algorithm, which uses a block cipher with a block size of 128 bits. We want to emphasize that GCM commonly utilizes AES-128 as its block cipher [21]. We will not discuss the GCM algorithm but merely use its ability to view AES-128 as an AEAD cipher. We call this "new" algorithm AES-128-GCM.
In table 3.8, we remind ourselves of the parameters of Ascon and give the parameters of AES-128-GCM.

	key	nonce	tag	data block	round numbers
Ascon -128	128	128	128	64	12 and 6
AES-128-GCM	128	96	128	128	10

Table 3.8: *Parameters of* Ascon *-128 and AES-128*

The following presentation on implementations of the algorithms is based on [17] and [1]. We consider throughput (measured in encrypted bits per clock cycle), area, energy (measured in joules per encrypted bit), and energy times area. The measure of the area is not given in GE as per usual but rather as the "*average scaled area*". For a definition of the average scaled area, see [1, Section 2.5]. Both area and energy are "smaller-is-better" metrics, and their product helps simplify the comparison of different cryptographic algorithms. We refer to Table 3.9 for results.

	Area	Throughput	Energy	Energy×Area
Ascon -128	1.56	16.00	0.44	0.76
AES-128-GCM	2.75	11.63	0.71	2.05

Table 3.9: *Results on comparison of* Ascon *-128 and AES-128-GCM*

We see that Ascon -128 outperforms AES-128-GCM in all metrics.
We also consider the implementation in specific technologies and the code size in these. We refer to Tables 3.10 and 3.11 for the results.

	F1	ESP	F7	R5
Ascon -128	76.7	22.3	13.8	8.5
AES-128-GCM	332.8	67.2	35.8	23.7

Table 3.10: *Time to process NIST testvectors in μs*

	F1	ESP	F7	R5
Ascon -128	2157	1120	1180	1792
AES-128-GCM	9908	14832	9836	14272

Table 3.11: *Code size in bytes*

We will not comment on the technologies, but once again, we see that Ascon outperforms AES-128-GCM in processing time and code size.

Chapter 4 Linear and differential cryptanalysis

Mainly using [11], [8], [7] and the original paper on linear cryptanalysis [15], this chapter will discuss the theory of linear and differential cryptanalysis.

4.1 A generic description of linear cryptanalysis

In this section, we aim to describe a generic version of linear cryptanalysis with an emphasis on the structure of the attack and the use of the method in general.
Linear cryptanalysis is a type of known plaintext attack where the adversary examines the probabilistic linear relationship, also referred to as *linear approximations*, among the *parity bits*[1] of the plaintext P, the cipher text C and the encryption key K inside the cipher. Given a linear approximation with high or low probability, the adversary can estimate the secret key's parity bit by examining the parity bits of the plaintext and the ciphertext. Then, using supplementary methods, the attack can be extended to find more bits of the secret key.

We now explain the above in more detail. We let $A \in \mathbb{F}_2^n$ and let A_i denote the ith entry of A. We denote the parity bit of A, $A_{i_1} \oplus \ldots \oplus A_{i_n}$, by $\bigoplus_{m=1}^{n} A_{i_m}$. Let $P, \alpha \in \mathbb{F}_2^{|P|}$, $C, \beta \in \mathbb{F}_2^{|C|}$ and $K, \gamma \in \mathbb{F}_2^{|K|}$. The purpose of linear cryptanalysis is to approximate an expression of the form:

$$\bigoplus_{m=1}^{|P|} \alpha_m P_m \oplus \bigoplus_{m=1}^{|C|} \beta_m C_m = \bigoplus_{m=1}^{|K|} \gamma_m K_m. \tag{4.1}$$

Definition 4.1 Masks
Define

$$\bigoplus_{m=1}^{|P|} \alpha_m P_m \oplus \bigoplus_{m=1}^{|C|} \beta_m C_m = \bigoplus_{m=1}^{|K|} \gamma_m K_m.$$

We refer to α, β, and γ as the masks of the equation and say that P, C, and K are masked by α, β, and γ, respectively. We call α the input mask, β the output mask, and γ the key mask of the equation.

For simple linear functions, such as $\oplus$ and permutations, uncomplicated linear expressions can be expressed such that Equation 4.1 holds with probability one. Also, when examining affine functions, we can arrive at uncomplicated linear expressions by adding one to the function if it does not pass through the origin. For non-linear operations, such as S-boxes, we try to find linear approximations that hold with probability p_L, that maximizes $|p_L - 1/2|$.

Definition 4.2 Linear probability bias
We set $|p_L - 1/2| = \varepsilon$ and call this the linear probability bias.

Whenever there is no chance of confusion, we denote the linear probability bias as the bias.

[1] The parity bit of any string is the sum of all its entries modulus 2.

The approximation in Equation 4.1 is interesting only if $p_L \neq 1/2$ as if we independently and uniformly selected random elements from $\mathbb{F}_2$ and placed them into Equation 4.1 the probability of the expression holding would be exactly $1/2$.

In linear cryptanalysis, we exploit and examine the deviation or bias from the probability of $1/2$ for the expression to hold. We refer to the linear approximation when $(p_L - 1/2)$ reaches its max as the *best approximation* and the probability p_L as the *best probability.*

4.1.1 The linear approximation table

The S-box is often the only non-affine part of a cipher; therefore, we need to find good linear approximations of it to attack it.
In the following, we draw inspiration from the lecture [13]. We have the following linear equation associated with the S-box $\mathcal{S} : \mathbb{F}_2^n \to \mathbb{F}_2^m$ with $\alpha \in \mathbb{F}_2^n$ and $\beta \in \mathbb{F}_2^m$:

$$\alpha^T x \oplus \beta^T \mathcal{S}(x) = 0. \tag{4.2}$$

We now define the solution space and the solution space cardinality of Equation 4.2.

Definition 4.3 Solution space of S-box associated linear equation
Let $\mathcal{S}$ be an S-box and let Equation 4.2 be its associated linear equation. Fix the masks of the equation. We denote the solution space of the equation by

$$\Sigma_{\alpha,\beta} = \{x \in \mathbb{F}_2^n : \alpha^T x \oplus \beta^T \mathcal{S}(x) = 0\},$$

and denote the number of solutions by

$$e_{\alpha,\beta} = |\Sigma_{\alpha,\beta}|.$$

As $\Sigma_{\alpha,\beta} \subseteq \mathbb{F}_2^n$ get the following simple bounds on the number of solutions

$$0 \leq e_{\alpha,\beta} \leq 2^n. \tag{4.3}$$

We calculate the probability of Equation 4.2 holding for a random x given α and β as

$$p_{\alpha,\beta} = \frac{|\Sigma_{\alpha,\beta}|}{|\mathbb{F}_2^n|} = \frac{e_{\alpha,\beta}}{2^n}.$$

The *bias* can be calculated as

$$\begin{aligned} \varepsilon_{\alpha,\beta} &= p_{\alpha,\beta} - \frac{1}{2} \\ &= \frac{e_{\alpha,\beta}}{2^n} - \frac{1}{2} \\ &= \frac{e_{\alpha,\beta} - 2^{n-1}}{2^n}. \end{aligned}$$

We let $e'_{\alpha,\beta} = e_{\alpha,\beta} - 2^{n-1}$ and we refer it as the *bias integer*. In the following simple way, we express the bias using the bias integer

$$\varepsilon_{\alpha,\beta} = \frac{e'_{\alpha,\beta}}{2^n}$$

By subtracting 2^{n-1} in Equation 4.3 we get the following bounds of the bias integer

$$\begin{aligned} 0 - 2^{n-1} &\leq e_{\alpha,\beta} - 2^{n-1} \leq 2^n - 2^{n-1} \\ -2^{n-1} &\leq e'_{\alpha,\beta} \leq 2 \cdot 2^{n-1} - 2^{n-1} \\ -2^{n-1} &\leq e'_{\alpha,\beta} \leq 2^{n-1}. \end{aligned} \tag{4.4}$$

The table that for any given values of α and β displays the value of the bias integer $e'_{\alpha,\beta}$ is called the linear approximation table (LAT).

Definition 4.4 Linear approximation table (LAT)
Let $\mathcal{S} : \mathbb{F}_2^n \to \mathbb{F}_2^m$ by any S-box. The LAT of $\mathcal{S}$ is a table of integers with 2^n rows indexed by the elements of $\mathbb{F}_2^n$ and 2^m columns indexed by the elements of $\mathbb{F}_2^m$. The entry at row α and column β is given as the bias integer $e'_{\alpha,\beta}$:

$$\mathcal{L}_{\mathcal{S}} = (e'_{\alpha,\beta}).$$

One can view the LAT of an S-box as a $2^n \times 2^m$ matrix of all possible bias integers $e'_{\alpha,\beta}$. Regarding the computational complexity of the LAT of an S-box, we see that there are 2^n possible choices for α and 2^m possible choices for β totaling in a combined $2^n \cdot 2^m = 2^{n+m}$ possible equations of the form $\alpha^T x \oplus \beta^T \mathcal{S}(x) = 0$. There are an additional 2^n possible inputs $x \in \mathbb{F}_2^n$, so calculating the LAT of an S-box requires a total of $2^n \cdot 2^{n+m} = 2^{2n+m}$ calculations. We now give an example wherein we calculate some of the entries of our toy cipher.

Example 4.5
In this example, we calculate some of the entries of the linear approximation table. The entire table can be found in Table 4.1.
We start by calculating the first entry, corresponding to $e'_{0,0}$ From the above discussed, we know that $e'_{\alpha,\beta} = e_{\alpha,\beta} - 2^{n-1}$, and that $e_{\alpha,\beta} = |\Sigma_{\alpha,\beta}| = |\{x \in \mathbb{F}_2^n : \alpha^T x \oplus \beta^T \mathcal{S}_{toy}(x) = 0\}|$. We note that for all $x \in \mathbb{F}_2^4$ it holds that

$$0^T x \oplus 0^T \mathcal{S}_{toy}(x) = 0.$$

Thus we get that $|\Sigma_{0,0}| = 2^5$ and thereby $e_{0,0} = 2^5 - 2^4 = 8$. Calculating the entire first column can be done by using Lemma 4.10, and the first row can be calculated using Lemma 4.11, which we will state and prove later. We now move on to calculating a random entry, say an input sum of 7 and an output sum of c. That is, we have to find the values of x for which the expression $(0111)x \oplus (1100)\mathcal{S}_{toy}(x) = 0$ holds true. To do this, we check all possible values of x. We now use hexadecimal notation

- $7^T \cdot 0 \oplus c^T \cdot \mathcal{S}_{toy}(0) = 0 \oplus 0 = 0$
- $7^T \cdot 1 \oplus c^T \cdot \mathcal{S}_{toy}(1) = 1 \oplus 1 = 0$
- $7^T \cdot 2 \oplus c^T \cdot \mathcal{S}_{toy}(2) = 1 \oplus 1 = 1$
- $7^T \cdot 3 \oplus c^T \cdot \mathcal{S}_{toy}(3) = 0 \oplus 0 = 0$
- $7^T \cdot 4 \oplus c^T \cdot \mathcal{S}_{toy}(4) = 1 \oplus 0 = 1$
- $7^T \cdot 5 \oplus c^T \cdot \mathcal{S}_{toy}(5) = 0 \oplus 0 = 0$

- $7^T \cdot 6 \oplus c^T \cdot \mathcal{S}_{toy}(6) = 0 \oplus 1 = 1$
- $7^T \cdot 7 \oplus c^T \cdot \mathcal{S}_{toy}(7) = 1 \oplus 1 = 0$
- $7^T \cdot 8 \oplus c^T \cdot \mathcal{S}_{toy}(8) = 0 \oplus 0 = 0$
- $7^T \cdot 9 \oplus c^T \cdot \mathcal{S}_{toy}(9) = 1 \oplus 1 = 0$
- $7^T \cdot a \oplus c^T \cdot \mathcal{S}_{toy}(a) = 1 \oplus 1 = 0$
- $7^T \cdot b \oplus c^T \cdot \mathcal{S}_{toy}(b) = 1 \oplus 1 = 0$
- $7^T \cdot c \oplus c^T \cdot \mathcal{S}_{toy}(c) = 1 \oplus 1 = 0$
- $7^T \cdot d \oplus c^T \cdot \mathcal{S}_{toy}(d) = 0 \oplus 1 = 1$
- $7^T \cdot e \oplus c^T \cdot \mathcal{S}_{toy}(e) = 0 \oplus 0 = 0$
- $7^T \cdot f \oplus c^T \cdot \mathcal{S}_{toy}(f) = 1 \oplus 1 = 0$

We see that the expressions hold for 12 values of x and thus the entry $\mathcal{L}_S = (e'_{7,c}) = 12 - 8 = 4$. We calculate all the entries, and the table can be found in Table 4.1.

		Output sum															
		0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
Input sum	0	8	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	1	0	0	-2	-2	0	0	-2	6	2	2	0	0	2	2	0	0
	2	0	0	-2	-2	0	0	-2	-2	0	0	2	2	0	0	-6	2
	3	0	0	0	0	0	0	0	0	2	-6	-2	-2	2	2	-2	-2
	4	0	2	0	-2	-2	-4	-2	0	0	-2	0	2	2	-4	2	0
	5	0	-2	-2	0	-2	0	4	2	-2	0	-4	2	0	-2	-2	0
	6	0	2	-2	4	2	0	0	2	0	-2	2	4	-2	0	0	-2
	7	0	-2	0	2	2	-4	2	0	-2	0	2	0	4	2	0	2
	8	0	0	0	0	0	0	0	0	-2	2	2	-2	2	-2	-2	-6
	9	0	0	-2	-2	0	0	-2	-2	-4	0	-2	2	0	4	2	-2
	a	0	4	-2	2	-4	0	2	-2	2	2	0	0	2	2	0	0
	b	0	4	0	-4	4	0	4	0	0	0	0	0	0	0	0	0
	c	0	-2	4	-2	-2	0	2	0	2	0	2	4	0	2	0	-2
	d	0	2	2	0	-2	4	0	2	-4	-2	2	0	2	0	0	2
	e	0	2	2	0	-2	-4	0	2	-2	0	0	-2	-4	2	-2	0
	f	0	-2	-4	-2	-2	0	2	0	0	-2	4	-2	-2	0	2	0

Table 4.1: *The linear approximation table of the toy cipher.*

The LAT can be used to compute the bias of a single S-box of a cipher, and if this were all we needed, we would be done. Unfortunately for the adversary, this is not the case. A cipher often consists of multiple rounds with multiple S-boxes in each round. Brute force can be used to find the most biased linear approximation on a component such as an S-box, but several problems arise when using this method on a full-sized cipher. The adversary would have to compute all possible linear approximations throughout the full cipher and then compute their biases using all possible plaintexts. This clearly is infeasible for any practical cipher. We solve this problem by making several assumptions and thereby approximate the probability. We approximate the bias integers for multiple S-boxes using the Piling-up Lemma.

4.1.2 Piling-up lemma

We aim to find an effective way to approximate the bias integers $e'_{\alpha,\beta}$, equivalent to finding effective approximations. Alternatively, one can demonstrate that effective approximations do not exist, but this is often much more difficult, as it would often require checking all possible approximations. Finding approximations has to be done explicitly for each different cipher one wishes to attack, as different ciphers should not share their inner structures. We introduce the Piling-up Lemma after a brief discussion.

Consider two random variables $X_1, X_2 \in \mathbb{F}_2$, and begin by noting that the relationship $X_1 \oplus X_2 = 0$ is a linear expression and equivalent to $X_1 = X_2$, while $X_1 \oplus X_2 = 1$ is an affine expression and equivalent to $X_1 \neq X_2$. We assume that the probability is distributed as in the following:

$$\Pr(X_1 = i) = \begin{cases} p_1, & i = 0 \\ 1 - p_1, & i = 1 \end{cases}$$

and

$$\Pr(X_2 = j) = \begin{cases} p_2, & j = 0 \\ 1 - p_2, & j = 1. \end{cases}$$

If we further assume that the variables are independent, we get the following probability distribution:

$$\Pr(X_1 = i, X_2 = j) \begin{cases} p_1 p_2, & i = 0, j = 0 \\ p_1(1 - p_2), & i = 0, j = 1 \\ (1 - p_1)p_2, & i = 1, j = 0 \\ (1 - p_1)(1 - p_2), & i = 1, j = 1. \end{cases}$$

We now examine the probability that $X_1 \oplus X_2 = 0$:

$$\begin{aligned} \Pr(X_1 \oplus X_2 = 0) &= \Pr(X_1 = X_2) \\ &= \Pr(X_1 = 0, X_2 = 0) + \Pr(X_1 = 1, X_2 = 1) \end{aligned}$$

and by independence we get

$$\begin{aligned} &= \Pr(X_1 = 0)\Pr(X_2 = 0) + \Pr(X_1 = 1)\Pr(X_2 = 1) \\ &= p_1 p_2 + (1 - p_1)(1 - p_2). \end{aligned}$$

Let $p_1 = 1/2 + \varepsilon_1$ and $p_2 = 1/2 + \varepsilon_2$, where ε_1 and ε_2 are the probability baises and $-1/2 \leq \varepsilon_1, \varepsilon_2 \leq 1/2$ we get:

$$\begin{aligned} \Pr(X_1 \oplus X_2 = 0) &= (1/2 + \varepsilon_1)(1/2 + \varepsilon_2) + (1/2 - \varepsilon_1)(1/2 - \varepsilon_2) \\ &= 1/4 + \varepsilon_1\varepsilon_2 + 1/2\varepsilon_1 + 1/2\varepsilon_2 + 1/4 - 1/2\varepsilon_1 - 1/2\varepsilon_2 + \varepsilon_1\varepsilon_2 \\ &= 1/2 + 2\varepsilon_1\varepsilon_2. \end{aligned}$$

Letting $\varepsilon_{1,2}$ denote the bias of the linear expression $X_1 \oplus X_2 = 0$ we get

$$\varepsilon_{1,2} = 2\varepsilon_1\varepsilon_2.$$

We can further extend this to n random independent variables $X_1, \ldots, X_n \in \mathbb{F}_2$, with probabilities $p_i = 1/2 + \varepsilon_i$ for $1 \leq i \leq n$. To calculate the probability that $X_1 \oplus \ldots \oplus X_n = 0$, we use the following Lemma.

Lemma 4.6 Piling-up Lemma
For n independent random variables $X_1, \ldots, X_n$ over $\mathbb{F}_2$ we have the following relation:

$$\Pr(X_1 \oplus \ldots \oplus X_n = 0) = 1/2 + 2^{n-1} \prod_{i=1}^{n} \varepsilon_i,$$

or equivalently the bias $\varepsilon_{1,\ldots,n}$ of $X_1 \oplus \ldots \oplus X_n = 0$

$$\varepsilon_{1,\ldots,n} = 2^{n-1} \prod_{i=1}^{n} \varepsilon_i.$$

Proof. We will prove the Lemma using induction over k. Let $k = 2$, and note that the Lemma holds based on the above discussion. Assuming that the Lemma holds for $k = n-1$, we will now prove it holds for $k = n$. We have the following:

$$\Pr\left(\bigoplus_{i=1}^{n-1} X_i \oplus X_n = 0\right) = \Pr\left(\bigoplus_{i=1}^{n-1} X_i = 0, X_n = 0\right) + \Pr\left(\bigoplus_{i=1}^{n-1} X_i = 1, X_n = 1\right)$$

and by assuming independence, we get

$$\begin{aligned}
&= \Pr\left(\bigoplus_{i=1}^{n-1} X_i = 0\right) \Pr(X_n = 0) + \Pr\left(\bigoplus_{i=1}^{n-1} X_i = 1\right) \Pr(X_n = 1) \\
&= \left(\frac{1}{2} + 2^{n-2} \prod_{i=1}^{n-1} \varepsilon_i\right) (\frac{1}{2} + \varepsilon_n) + \left(\frac{1}{2} - 2^{n-2} \prod_{i=1}^{n-1} \varepsilon_i\right) (\frac{1}{2} - \varepsilon_n) \\
&= \frac{1}{4} + \frac{1}{2}\varepsilon_n + \frac{1}{2}2^{n-2} \prod_{i=1}^{n-1} \varepsilon_i + \varepsilon_n 2^{n-2} \prod_{i=1}^{n-1} \varepsilon_i + \frac{1}{4} - \frac{1}{2}\varepsilon_n - \frac{1}{2}2^{n-2} \prod_{i=1}^{n-1} \varepsilon_i + \varepsilon_n 2^{n-2} \prod_{i=1}^{n-1} \varepsilon_i \\
&= \frac{1}{2} + 2\varepsilon_n 2^{n-2} \prod_{i=1}^{n-1} \varepsilon_i \\
&= \frac{1}{2} + 2^{n-1} \prod_{i=1}^{n} \varepsilon_i.
\end{aligned}$$

The bias can be calculated as the amount $\Pr(\bigoplus_{i=1}^{n} X_i = 0)$ differs from $1/2$, and we get:

$$\varepsilon_{1,\ldots,n} = 2^{n-1} \prod_{i=1}^{n} \varepsilon_i,$$

which concludes the proof. □

To estimate the probability of a linear approximation using the Piling-up Lemma, we write the approximation as a chain of connected linear approximations, each spanning a small part of the cipher. We call such a chain a *linear characteristic* of a cipher. Under the assumption that each of the biases of the partial approximations is independent, the total bias can be computed using the Piling-up Lemma. We demonstrate the above in the following simple example using three random variables.

Example 4.7
Consider three independent and random variables $X_1, X_2, X_3 \in \mathbb{F}_2$, and let $\Pr(X_1 \oplus X_2 = 0) = 1/2 + \varepsilon_{1,2}$ and $\Pr(X_2 \oplus X_3 = 0) = 1/2 + \varepsilon_{2,3}$. Consider the result of $X_1 \oplus X_3$ to be derived by adding together $X_1 \oplus X_2$ and $X_2 \oplus X_3$, we get:

$$\Pr(X_1 \oplus X_3 = 0) = \Pr([X_1 \oplus X_2] \oplus [X_2 \oplus X_3] = 0).$$

By combining linear expressions, we get a new linear expression, and as we can consider the random variables $X_1 \oplus X_2$ and $X_2 \oplus X_3$ independent, we can calculate $\Pr(X_1 \oplus X_3 = 0)$ as:

$$\Pr(X_1 \oplus X_3 = 0) = 1/2 + 2\varepsilon_{1,2}\varepsilon_{2,3}.$$

It should be noted that the biases calculated using the Piling-up Lemma are only estimates of the real biases and only hold when the assumption of independence is fulfilled. Unexpected effects may occur when the variables are not independent, and in general, the biases in these cases can both be smaller and larger than predicted by the Piling-up Lemma.

4.1.3 Properties of the LAT

We prove several properties of the LAT, ending with all entries of the LAT are even. In the following, let $\mathcal{S}$ be an S-box with range $\mathbb{F}_2^n$ and domain $\mathbb{F}_2^m$ and let $\mathcal{L}_\mathcal{S}$ be its corresponding LAT.

Lemma 4.8
The first entry of $\mathcal{L}_\mathcal{S}$ is given as $(e'_{0,0}) = (2^{n-1})$

Proof. By Equation 4.4 the bias integers are bounded by

$$-2^{n-1} \leq e'_{\alpha,\beta} \leq 2^{n-1}.$$

Let both α and β be equal to 0. Then every $x \in \mathbb{F}_2^n$ solves

$$0^T x \oplus 0^T \mathcal{S}(x) = 0,$$

and hence $\Sigma_{0,0} = \mathbb{F}_2^n$, which implies that the number of solution are

$$|\Sigma_{0,0}| = |\mathbb{F}_2^n| = 2^n = e_{0,0}.$$

By definition, the bias integer is given as

$$e'_{0,0} = e_{0,0} - 2^{n-1} = 2 \cdot 2^{n-1} - 2^{n-1} = 2^{n-1},$$

which completes the proof. □

We now prove a lemma concerning the number of solutions to the first term of Equation 4.2 whenever $\alpha \neq 0$.

Lemma 4.9
Let $\alpha \in \mathbb{F}_2^n$ be non-zero and let $n > 1$. Define $\mathcal{W} = \{x \in \mathbb{F}_2^n : \alpha^T x = 0\}$. The cardinality of $\mathcal{W}$ is equal to 2^{n-1}.

Proof. As $\alpha = (\alpha_1, \ldots, \alpha_n) \neq 0$, there exists some i_0 such that $\alpha_{i_0} = 1$. For the sake of argument let $\alpha_1 = 1$ and write $\alpha = (1, \alpha') \in \mathbb{F}_2^n$ where $\alpha' \in \mathbb{F}_2^{n-1}$, and define the following map

$$\begin{aligned} \varphi : \mathbb{F}_2^{-1} &\to \mathcal{W} \\ x &\mapsto (\alpha'^T x, x). \end{aligned}$$

The map φ is well defined as

$$\alpha^T \varphi(x) = \alpha^T(\alpha'^T x, x) = \alpha'^T x \oplus \alpha'^T x = 0,$$

where the second equality follows as $\alpha = (1, \alpha')$ and the third equality follows as any element of $\mathbb{F}_2^n$ is its own additive inverse. We now prove that φ is a bijection, which implies that $\mathcal{W}$ and $\mathbb{F}_2^{n-1}$ have the same cardinality. We start by proving that φ is injective (one-to-one), and since we are working over a finite field, the function must be a bijection. Assume that $\varphi(x) = \varphi(y)$, then

$$\varphi(x) = (\alpha'^T x, x) = \varphi(y) = (\alpha'^T y, y),$$

and we see that $x = y$. □

The next Lemma concerns the first column of the LAT.

Lemma 4.10
The last $2^n - 1$ entries of the first column of $\mathcal{L}_\mathcal{S}$ are zeros.

Proof. Let $\alpha \neq 0$ and $\beta = 0$, then Equation 4.2 is equivalent to

$$\alpha^T x = 0.$$

By Lemma 4.9, we know that $|\mathcal{W}| = 2^{n-1}$, and that this is the number of solutions to the above equation, hence

$$e_{\alpha,0} = 2^{n-1}$$

which implies that

$$e'_{\alpha,0} = e_{\alpha,0} - 2^{n-1} = 0.$$

This is true for all $0 < \alpha \leq 2^n$, and thus the last $2^n - 1$ entries of the first column of $\mathcal{L}_\mathcal{S}$ are all zeros. □

The case whenever $\beta = 0$ is not interesting since it does not involve the S-box. The next Lemma concerns bijective S-boxes.

Lemma 4.11
Let $\mathcal{S}$ be a bijection, then the LAT related to $\mathcal{S}^{-1}$ is given as the transpose of $\mathcal{S}$:

$$\mathcal{L}_{\mathcal{S}^{-1}} = (\mathcal{L}_{\mathcal{S}})^T.$$

Proof. Let $\mathcal{S}$ be a bijection, this implies that $m = n$ and

$$\begin{aligned}\mathcal{S} &: \mathbb{F}_2^n \to \mathbb{F}_2^n \\ \mathcal{S}^{-1} &: \mathbb{F}_2^n \to \mathbb{F}_2^n.\end{aligned}$$

Let $x \in \mathbb{F}_2^n$ and let $\mathcal{S}(x) = y$, then $x = \mathcal{S}^{-1}(y)$. Consider the equation

$$\alpha^T x \oplus \beta^T \mathcal{S}(x) = 0,$$

and replace x with $\mathcal{S}^{-1}(y)$ and $\mathcal{S}(x)$ with y, we get

$$\alpha^T x \oplus \beta^T \mathcal{S}(x) = \beta^T y \oplus \alpha^T \mathcal{S}^{-1}(y) = 0.$$

Notice that α and β have "switched" places. By definition 4.3 we get $e^{\mathcal{S}}_{\alpha,\beta} = e^{\mathcal{S}^{-1}}_{\beta,\alpha}$ and after calculating the bias integers $e'S_{\alpha,\beta}$ and $e'^{\mathcal{S}^{-1}}_{\beta,\alpha}$ we get

$$e'S_{\alpha,\beta} = e'^{\mathcal{S}^{-1}}_{\beta,\alpha},$$

which is equivalent to $\mathcal{L}_{\mathcal{S}^{-1}} = (\mathcal{L}_{\mathcal{S}})^T$. □

In addition to the fact that $\mathcal{L}_{\mathcal{S}^{-1}} = (\mathcal{L}_{\mathcal{S}})^T$ we also get that the first row and column of any LAT which has a bijective S-box are equal, which by Lemma 4.8 and Lemma 4.10 is a vector of the form $(2^{n-1}||0^{2^n-1})$.
We now have everything we need to prove that all the entries of any LAT are even.

Proposition 4.12
Let $\mathcal{S} : \mathbb{F}_2^n \to \mathbb{F}_2^n$ be a bijection and let $n > 1$. Then all the entries of $\mathcal{L}_{\mathcal{S}}$ are even.

Proof. The first entry is 2^{n-1}, which is even whenever $n > 1$. The other values of the first column and the first row are all zero. Let $\alpha, \beta \neq 0$, and define the following sets

$$\begin{aligned} A_0 &= \{x \in \mathbb{F}_2^n : \alpha^T x = 0\}, \qquad & B_0 &= \{x \in \mathbb{F}_2^n : \beta^T \mathcal{S}(x) = 0\} \\ A_1 &= \{x \in \mathbb{F}_2^n : \alpha^T x = 1\}, \qquad & B_1 &= \{x \in \mathbb{F}_2^n : \beta^T \mathcal{S}(x) = 1\}.\end{aligned}$$

The sets are partitions of $\mathbb{F}_2^n$ and $A_0 \cup A_1 = \mathbb{F}_2^n$ and likewise $B_0 \cup B_1 = \mathbb{F}_2^n$. By Lemma 4.9 $|A_0| = 2^{n-1}$ and hence $|A_1| = |\mathbb{F}_2^n| - 2^{n-1} = 2^{n-1} = |A_0|$. As $\mathcal{S}$ is a bijection we get that $|B_0| = |B_1| = 2^{n-1}$ by using the same "$\alpha^T x \oplus \beta^T \mathcal{S}(x)$ is equivalent to $\beta^T y \oplus \alpha^T \mathcal{S}^{-1}(y)$" argument as in Lemma 4.11. We now have that

$$|A_0| = |A_1| = |B_0| = |B_1| = 2^{n-1},$$

which are all of even cardinality. For $i, j \in \{0, 1\}$ define $k_{i,j} = |A_i \cap B_j|$.
Notice that

- $A_0 \cap B_0$ and $A_0 \cap B_1$ is a partition of A_0, which implies that $k_{0,0} \equiv k_{0,1} \mod 2$
- $A_1 \cap B_0$ and $A_1 \cap B_1$ is a partition of A_1, which implies that $k_{1,0} \equiv k_{1,1} \mod 2$
- $A_0 \cap B_0$ and $A_1 \cap B_0$ is a partition of B_0, which implies that $k_{0,0} \equiv k_{1,0} \mod 2$
- $A_0 \cap B_1$ and $A_1 \cap B_1$ is a partition of B_1, which implies that $k_{0,1} \equiv k_{1,1} \mod 2$.

By equivalence, we have that

$$k_{0,0} \equiv k_{0,1} \equiv k_{1,0} \equiv k_{1,1} \mod 2,$$

which implies that they are all even or odd. The solutions to $\alpha^T x \oplus \beta^T \mathcal{S}(x)$ can be expressed as the disjoint unions

$$\Sigma_{\alpha,\beta} = (A_0 \cap B_0) \cup (A_1 \cap B_1).$$

The number of solutions $e_{\alpha,\beta}$ then must be the number of elements of $A_0 \cap B_0$ and $A_1 \cap B_1$:

$$e_{\alpha,\beta} = k_{0,0} + k_{1,1} \equiv 0 \mod 2,$$

which implies that $e_{\alpha,\beta}$ is an even integer. Thus, the entries of the LAT $e'_{\alpha,\beta} = e_{\alpha,\beta} - 2^{n-1}$ are even as they are differences of even integers. □

4.1.4 Recovering key bits

We look into how the attacker can recover bits of the key if they have found a good linear approximation. We define the correlation of an approximation as this will be used later.

Definition 4.13 Correlation
The correlation $\text{cor}_F(\alpha, \beta)$ of an approximation (α, β) of a function $F : \mathbb{F}_2^n \to \mathbb{F}_2^m$ can be represented as:

$$\begin{aligned} \text{cor}_F(\alpha, \beta) &= 2\left(\Pr(\alpha^T P \oplus \beta^T C = 0) - \frac{1}{2}\right) \\ &= \Pr(\alpha^T x \oplus \beta^T F(x) = 0) - \Pr(\alpha^T x \oplus \beta^T F(x) = 1) \\ &= 2\varepsilon \end{aligned}$$

The correlation takes values between -1 and 1. Also, if the correlation is positive, it implies that the bias of the masked pair (α, β) is greater than 0. Likewise, if the correlation is negative, it implies that the bias of the masked pair (α, β) is less than 0. We conclude that it is more probable that $\alpha^T P \oplus \beta^T C$ equals 0 than it equals 1, whenever $\text{cor}_F(\alpha, \beta) > 0$. Likewise, we conclude that it is more probable that $\alpha^T P \oplus \beta^T C$ equals 1 than it equals 0, whenever $\text{cor}_F(\alpha, \beta) < 0$. If we consider Equation 4.1

$$\bigoplus_{m=1}^{n} (\alpha_m P_m \oplus \beta_m C_m) = \bigoplus_{m=1}^{n} \gamma_m K_m,$$

we can predict the sum $\bigoplus_{m=1}^{n} \gamma_m K_m$ by looking at the correlation of (α, β). In other words, once we arrive at an effective approximation, we can determine $\bigoplus_{m=1}^{w} K[k_m]$ using the following maximum likelihood-based algorithm.

Algorithm 4.14
Given a pool of N random plaintexts, initialize two counters T_0 and T_1. Then, for each plaintext/ciphertext pair (P_i, C_i):

$$\text{if } \alpha^T P_i \oplus \beta^T C_i = 0 \text{ increase } T_0 \text{ by } 1$$
$$\text{if } \alpha^T P_i \oplus \beta^T C_i = 1 \text{ increase } T_1 \text{ by } 1.$$

We get the following 1-bit information about the key:

$$\text{If } T_0 > T_1 \Rightarrow \gamma^T K = 0$$
$$\text{If } T_0 < T_1 \Rightarrow \gamma^T K = 1.$$

The above algorithm has its advantages and disadvantages. It requires an approximation for all rounds of the cipher, which is computationally challenging for any practical cipher. Also, we only learn about one bit of key information and need several approximations for more key information. If we learn enough key information, we can guess the key.

4.2 A generic description of differential cryptanalysis

In this section, we aim to describe a generic version of differential cryptanalysis, emphasizing the structure of the attack. Differential cryptanalysis was originally presented to attack DES, whose underlying architecture differs from that of Ascon . It was later revisited and adapted to the architecture of substitution permutation networks, which is the architecture Ascon relies on. We adopt part of the notation found in [2], from which we also draw inspiration.

Differential cryptanalysis is a type of chosen plaintext attack, meaning that the attacker can select specific inputs and examine outputs in an attempt to get information about the key. In differential cryptanalysis, the attacker examines the probabilistic occurrences of plaintext differences and differences in the later rounds of the cipher.
Let P and P' be plaintexts, and let C and C' be their corresponding ciphertexts. The main idea is to predict the effect of plaintext differences $\alpha = P \oplus P'$ on ciphertext differences $\beta = C \oplus C'$ without knowledge of the key K by tracing the effect of the difference α throughout the cipher. We explore the construction of a *differential* involving plaintext bits and the input to the later rounds of the cipher. We accomplish this by investigating highly probable *differential characteristics.*

We now explain the above in more detail. For now, we assume that we are working with an SPN. We now define the notion of a difference and a differential.

Definition 4.15 Difference and differential
Let $P, P' \in \mathbb{F}_2^n$ be plaintext into the i'th round and let $C, C' \in \mathbb{F}_2^n$ be their corresponding ciphertexts from the i'th round. We define the difference of the plaintexts and the ciphertexts as

$$\alpha^i = P \oplus P' \text{ and } \beta^i = C \oplus C'$$

and they are respectively called the input difference and the output difference of the system, and (α^i, β^i) is called a differential.

We write α and β whenever the round number is unimportant. A differential is used to predict that if plaintexts P and P' have a difference α, then the corresponding ciphertexts C and C' have a difference β with a certain probability. We denote the probability of an input difference having a certain output difference by

$$\Pr(\alpha \to \beta).$$

For any intermediate data x and x' that occurs during encryption, we denote the difference of these using equivalent notation

$$\alpha = x \oplus x'.$$

By x^i_t and x'^i_t, we respectively mean the input and output of the t'th S-box in the i'th round. Whenever the round is unimportant, we omit the subscript i; whenever the S-box number is unimportant, we omit the superscript t. Whenever it is not important whether a difference is an input or output difference, we denote it by γ. Differences from the i'th round is referred to as $\alpha^i = (a^i_1, \dots, a^i_m)$, $\beta^i = (b^i_1, \dots b^1_m)$ and $\gamma = (c^i_1, \dots, c^i_m)$. The index refers to the S-box. We now provide a theorem regarding the effect of linear and affine functions on differences.

Theorem 4.16
Linear and affine operations either do not affect the differences or affect the differences predictably.

Proof. Let $\alpha = x \oplus x'$.
Linear functions:
A linear function in $\mathbb{F}_2^n$ is represented as $L(x) = Ax$, where $A \in \mathbb{F}_2^{n\times n}$ and $x \in \mathbb{F}_2^n$. Consider $\alpha = x \oplus x'$. Applying the linear function to α, we get:

$$L(\alpha) = A(\alpha) = A(x \oplus x') = Ax \oplus Ax' = L(x) \oplus L(x')$$

Thus, the linear function L does not affect the difference $x \oplus x' = \alpha$.
Affine Functions:
An affine function in $\mathbb{F}_2^n$ is of the form $A(x) \oplus b$, where $A \in \mathbb{F}_2^{n\times n}$ and $x, b \in \mathbb{F}_2^n$. Applying the affine function to α, we get

$$A(\alpha) \oplus b = A(x \oplus x') \oplus b = Ax \oplus Ax' \oplus b = A(x) \oplus b \oplus A(x')$$

The effect of the affine function on the difference $x \oplus x'$ is predictable and is given by the constant term b. □

We give a few, but relevant, examples to illustrate the effect of linear or affine operations:

- **Bit permutations**, such as in the linear diffusion layer of Ascon , affects the differences by reordering them in the same way. To see this, let π be a permutation and define

$$\begin{aligned}\pi :&\mathbb{F}_2^n \to \mathbb{F}_2^n\\ &x \mapsto \pi(x).\end{aligned}$$

  Now see that $\pi(\alpha) = \pi(x) \oplus \pi(x')$, as permutations are linear maps.

- **Additions** of two values $\alpha \oplus \beta$, also adds the differences of the values to $\alpha \oplus \beta = (x \oplus y) \oplus (x' \oplus y')$.

- **Intermediate additions** can be ignored by means of differences. If data D is added to intermediate data x such that $y = x \oplus D$, and also the second intermediate data x', we get $y' = x' \oplus D$. The output differences $\beta = x \oplus D \oplus x' \oplus D = x \oplus x' = \alpha$ does not depend on D.

When looking at non-affine operations, such as some S-boxes, we study how the differences evolve throughout the cipher. Of course, whenever there is an input difference of 0, that is, whenever the inputs are equal, then the outputs must also be equal, and the output difference must be 0. The output difference cannot always be predicted for non-zero input differences as there can exist many different output differences for each input difference. We illustrate this in the following example.

Example 4.17
Let $\mathcal{S}$ be the S-box of the toy cipher introduced in the preliminary readings section. The S-box table can be found in Table 1.2. For an input difference of 5, we have many input pairs, such as 0 and 5, 1 and 6, and so on. For ease of understanding, we rewrite these from hexadecimal to binary in the following. For the input pair $(0000, 0101)$, the output pair is $(1110, 1111)$ with an output difference of 0001. The input pair $(0001, 0110)$ has an output pair $(0100, 1011)$ with an output difference of 1111. Even though the input differences are equal, the output differences are not.

While we cannot predict the exact output difference, it is possible to predict some statistical information on the output difference given the input difference. This is summed up in a *differential distribution table (DDT)*. In the following, we introduce relevant notation to define the DDT.

Let R be the total number of rounds, and let $k_1, \ldots, k_R$ denote round keys used for encryption. For each round i, $1 \leq i \leq R$, we define the input value of the $i+1$'th round as

$$\begin{aligned}x_{i+1} &= F_i(k_i, x_i)\\ x'_{i+1} &= F_i(k_i, x'_i).\end{aligned}$$

We also define the difference $\alpha_i = x_i \oplus x'_i$ as the input difference to the i'th round. Next, let

$$\begin{aligned}y_i &= \mathcal{S}^m(x_i \oplus k_i)\\ y'_i &= \mathcal{S}^m(x'_i \oplus k_i),\end{aligned}$$

and define the output difference of the i'th substitution layer as $\beta_i = y_i \oplus y'_i$. To sum up, y_i and y'_i are the output of the i'th substitution layer, while x_i and x'_i are the inputs of the

i'th round. The following relation is due to the linearity of the permutation layer π

$$\alpha_{i+1} = x_{i+1} \oplus x'_{i+1} = \pi(y_i) \oplus \pi(y'_i) = \pi(y_i \oplus y'_i) = \pi(\beta_i).$$

From this, we see that the input difference of round $i+1$ only depends on the output difference of round i. Assuming that the round keys are independent and uniformly distributed, the probability that a difference $a_i \in \mathbb{F}_2^s$ produces $b_i \in \mathbb{F}_2^s$ by the i'th S-box is given by

$$\Pr(a_i \to b_i) = \frac{|\{x \in \mathbb{F}_2^s | \mathcal{S}_i(x) \oplus \mathcal{S}_i(x \oplus a_i) = b_i\}|}{2^s}.$$

Calculating the above for all possible a's and b's results in a $2^s \times 2^s$ matrix of probabilities, called the *differential table* of the i'th S-box. The $2^s \times 2^s$ matrix consisting only of the numerators

$$|\{x \in \mathbb{F}_2^s | \mathcal{S}_i(x) \oplus \mathcal{S}_i(x \oplus a) = b\}|$$

is called the differential distribution table, abbreviated as DDT.

Definition 4.18 Differential distribution table
The DDT of an S-box $\mathcal{S}$ is the table that lists the number of elements $x \in \mathbb{F}_2^s$ such that

$$\mathcal{S}(x) \oplus \mathcal{S}(x \oplus a) = b$$

holds. The rows denote all the possible input differences a, and the columns denote all the possible output differences b.

We stress that

$$\Pr_i(0 \to 0) = 1,$$

as $\mathcal{S}_i(x) \oplus \mathcal{S}_i(x \oplus 0) = 0$.

Example 4.19
In this example, we calculate some of the entries of the differential distribution table. The full table can be found in Table 4.2. Our calculations are done in hexadecimal. We start by calculating the first entry, that is, we have to find the number of elements $x \in \mathbb{F}_2^4$ that fulfill

$$\mathcal{S}_{toy}(x) \oplus \mathcal{S}_{toy}(x+0) = 0.$$

Clearly, all 16 elements fulfill this requirement, as $\mathcal{S}_{toy}(x+0) = \mathcal{S}_{toy}(x)$ and as every element in $\mathbb{F}_2^4$ is its additive inverse. We can also quickly calculate the entire first row of the differential distribution table, as whenever the input difference is 0 and the output difference is non-zero, we have that

$$\mathcal{S}_{toy}(x) \oplus \mathcal{S}_{toy}(x+0) = 0.$$

We can also calculate the first column using a similar argumentation. Whenever the output difference is zero, but the input difference is non-zero, we require that

$$\mathcal{S}_{toy}(x) \oplus \mathcal{S}_{toy}(x+a) = 0,$$

which cannot be the case, as $\mathcal{S}_{toy}$ is injective. This will, in general, be true for any injective S-box. We often cannot argue as above; rather, we have to compute all possible combinations. We give a single example of this, namely whenever we have an input difference of 9 and an output difference of 7. We get the following

- $\mathcal{S}_{toy}(0) \oplus \mathcal{S}_{toy}(0+9) = \mathcal{S}_{toy}(0) \oplus \mathcal{S}_{toy}(9) = e + a = 4$
- $\mathcal{S}_{toy}(1) \oplus \mathcal{S}_{toy}(1+9) = \mathcal{S}_{toy}(1) \oplus \mathcal{S}_{toy}(8) = 4 + 3 = 7$
- $\mathcal{S}_{toy}(2) \oplus \mathcal{S}_{toy}(2+9) = \mathcal{S}_{toy}(2) \oplus \mathcal{S}_{toy}(b) = d + c = 1$
- $\mathcal{S}_{toy}(3) \oplus \mathcal{S}_{toy}(3+9) = \mathcal{S}_{toy}(3) \oplus \mathcal{S}_{toy}(a) = 1 + 6 = 7$
- $\mathcal{S}_{toy}(4) \oplus \mathcal{S}_{toy}(4+9) = \mathcal{S}_{toy}(4) \oplus \mathcal{S}_{toy}(d) = 2 + 9 = b$
- $\mathcal{S}_{toy}(5) \oplus \mathcal{S}_{toy}(5+9) = \mathcal{S}_{toy}(5) \oplus \mathcal{S}_{toy}(c) = f + 5 = a$
- $\mathcal{S}_{toy}(6) \oplus \mathcal{S}_{toy}(6+9) = \mathcal{S}_{toy}(6) \oplus \mathcal{S}_{toy}(f) = b + 7 = c$
- $\mathcal{S}_{toy}(7) \oplus \mathcal{S}_{toy}(7+9) = \mathcal{S}_{toy}(7) \oplus \mathcal{S}_{toy}(e) = 8 + 0 = 8$
- $\mathcal{S}_{toy}(8) \oplus \mathcal{S}_{toy}(8+9) = \mathcal{S}_{toy}(8) \oplus \mathcal{S}_{toy}(1) = 3 + 4 = 7$
- $\mathcal{S}_{toy}(9) \oplus \mathcal{S}_{toy}(9+9) = \mathcal{S}_{toy}(9) \oplus \mathcal{S}_{toy}(0) = a + e = 4$
- $\mathcal{S}_{toy}(a) \oplus \mathcal{S}_{toy}(a+9) = \mathcal{S}_{toy}(a) \oplus \mathcal{S}_{toy}(3) = 6 + 1 = 7$
- $\mathcal{S}_{toy}(b) \oplus \mathcal{S}_{toy}(b+9) = \mathcal{S}_{toy}(b) \oplus \mathcal{S}_{toy}(2) = c + d = 1$
- $\mathcal{S}_{toy}(c) \oplus \mathcal{S}_{toy}(c+9) = \mathcal{S}_{toy}(c) \oplus \mathcal{S}_{toy}(5) = 5 + f = a$
- $\mathcal{S}_{toy}(d) \oplus \mathcal{S}_{toy}(d+9) = \mathcal{S}_{toy}(d) \oplus \mathcal{S}_{toy}(4) = 9 + 2 = b$
- $\mathcal{S}_{toy}(e) \oplus \mathcal{S}_{toy}(e+9) = \mathcal{S}_{toy}(e) \oplus \mathcal{S}_{toy}(7) = 0 + 8 = 8$
- $\mathcal{S}_{toy}(f) \oplus \mathcal{S}_{toy}(f+9) = \mathcal{S}_{toy}(f) \oplus \mathcal{S}_{toy}(6) = 7 + b = c$

From this, we find that whenever the input difference is 9, the elements $1, 3, 8, a$ yield an output difference of 7. Since there are 4 such elements the entry of the differential distribution table corresponding to an input difference of 9 and an output difference of 7 is 4, as can be verified in Table 4.2.

Under the assumption that the S-boxes are independent, the probability that a difference $\alpha \in (\mathbb{F}_2^s)^m$ produces a difference $\beta \in (\mathbb{F}_2^s)^m$ by the substitution layer is the product of the probabilities of each of S-boxes producing the corresponding difference b_i given input a_i.

$$\Pr(\alpha \to \beta) = \prod_{i=1}^{m} \Pr(a_i \to b_i).$$

We now define the notion of an active S-box.

		Output difference															
		0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
Input difference	0	16	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	1	0	0	0	2	0	0	0	2	0	2	4	0	4	2	0	0
	2	0	0	0	2	0	6	2	2	0	2	0	0	0	0	2	0
	3	0	0	2	0	2	0	0	0	0	4	2	0	2	0	0	4
	4	0	0	0	2	0	0	6	0	0	2	0	4	2	0	0	0
	5	0	4	0	0	0	2	2	0	0	0	4	0	2	0	0	2
	6	0	0	0	4	0	4	0	0	0	0	0	0	2	2	2	2
	7	0	0	2	2	2	0	2	0	0	2	2	0	0	0	0	4
	8	0	0	0	0	0	0	2	2	0	0	0	4	0	4	2	2
	9	0	2	0	0	2	0	0	4	2	0	2	2	2	0	0	0
	a	0	2	2	0	0	0	0	0	6	0	0	2	0	0	4	0
	b	0	0	8	0	0	2	0	2	0	0	0	0	0	2	0	2
	c	0	2	0	0	2	2	2	0	0	0	0	2	0	6	0	0
	d	0	4	0	0	0	0	0	4	2	0	2	0	2	0	2	0
	e	0	0	2	4	2	0	0	0	6	0	0	0	0	0	2	0
	f	0	2	0	0	6	0	0	0	0	4	0	2	0	0	2	0

Table 4.2: *The differential distribution table of the toy cipher.*

Definition 4.20 Active S-box

Let $\gamma = (c_1, \ldots, c_m)$ be any difference. The i'th S-box is said to be activated by γ if $c_i \neq 0$. The function

$$\begin{aligned} \#\text{SB} :& (\mathbb{F}_2^s)^m \to [\![1, m]\!] \\ & \gamma \mapsto |\{i \in [\![1, m]\!] | c_i \neq 0\}| \end{aligned}$$

relates a difference γ to the number of S-boxes it activates.

We say an output difference β is a *candidate* for an input difference α if $\Pr(\alpha \to \beta) \neq 0$. The following Lemma connects the idea of a candidate with an active S-box.

Lemma 4.21

If β is a candidate for α, then $b_i = 0$ if and only if $a_i = 0$ for each $1 \leq i \leq m$, that is, they activate the same S-boxes. In this case,

$$\Pr(\alpha \to \beta) = \prod_i \Pr(a_i \to b_i).$$

Proof. Let β be a candidate for α, and let $1 \leq i \leq m$. As S-boxes are one-to-one, and as the probability $\Pr(\alpha \to \beta) \neq 0$ there exists some i for which the probability $\Pr(x_i \to b_i) \neq 0$. This corresponds to a non-zero input difference in the i'th S-box having a non-zero output difference. Also, if for some $1 \leq i' \leq m$ the probability $\Pr(a_{i'} \to 0) \neq 0$ holds, then $a_{i'} = 0$, must hold and likewise $\Pr(0 \to b_{i'}) \neq 0$ holds, then $b_{i'} = 0$ must hold, and the result follows. □

In an ideal cipher, the probability of any candidate β for the input difference α should be $1/2^n$, where n is the length of the x. In differential cryptanalysis, we seek to exploit a scenario where a particular β occurs given a particular input difference α with a very high probability p_D much larger than $1/2^n$. To find this probability, we look at a differential characteristic.

Definition 4.22 Differential characteristic
Let R be a non-negative integer. A R-round differential characteristic is an element $\mathcal{T} = ((\alpha_1, \beta_1), \ldots, (\alpha_R, \beta_R)) \in ((\mathbb{F}_2^{sm})^2)^R$, satisfying $\alpha_{i+1} = \pi(\beta_i)$ for all $1 \leq i < R$. The 0-round differential characteristic is denoted $()$. For each $0 \leq i \leq j \leq R$ let $\mathcal{T}_{[i,j]}$ denote the characteristic $((\alpha_i, \beta_i), \ldots, (\alpha_j, \beta_j))$.

As the round keys are assumed independently and uniformly distributed, a R-round differential characteristic can be computed as

$$\Pr(\mathcal{T}) = \prod_{i=1}^{R} \Pr(\alpha_i \to \beta_i) = \prod_{i=1}^{R} \left(\prod_{j=1}^{m} \Pr(a_j^i \to b_j^i) \right)$$

We call the differential characteristic with the highest probability after R-rounds the best R-round characteristic, and its probability is denoted $p_{\text{best}(R)}$. We can also extend characteristics to further rounds.

Definition 4.23 Extension
Let r, r' be integers such that $0 \leq r \leq r'$, and let $\mathcal{T}$ and $\mathcal{T}'$ respectively be r and r'-round characteristic. The characteristic $\mathcal{T}'$ extends $\mathcal{T}$ if the r first input and output differences are equal

$$\mathcal{T}'_{[1,r]} = \mathcal{T}.$$

We give a simplified example of a 3-round differential characteristic of the toy S-box under the assumption that there are no permutations and no constant addition layer, that is, each entry of an S-box is sent directly into the next S-box

Example 4.24
Let $\mathcal{S}$ be the S-box of the toy cipher as depicted in Table 1.2. Let the input difference to $\mathcal{S}_{toy}$ be 1011, then the output difference of the first round is 0010 with probability $8/16 = 1/2$. This can be seen in the DDT of the toy cipher Table 4.2. This then becomes the input difference to the second round. The output difference of the second round then becomes 0101 with a probability of 6/16, and the output difference of the third round then becomes 0001 with a probability of $4/16 = 1/4$. The differential characteristic here is $[(1011), (0010), (0101), (0001)]$ with a total probability of 0001 being the difference after the third round of $1/2 \cdot 6/16 \cdot 1/4 = 3/64$.

We now look into how to find the best characteristic using Matsui's algorithm for "The Search for the Best Probability" rewritten and optimized to target SPN ciphers [2]. This algorithm is called *OptTrailEst*.

Let $\tilde{R}$ denote the actual number of rounds. The algorithm presented will compute an optimal $\tilde{R}-$round characteristic without requiring any a priori knowledge. As input, it accepts any integer $R \geq 2$, the probabilities $(p_{\text{best}(i)})_{1 \leq i < R}$, and an estimate p_{est} of $p_{\text{best}(R)}$. The estimate is chosen such that $p_{\text{est}} \leq p_{\text{best}(R)}$. It returns an optimal $R-$round characteristic with its probability $p_{\text{best}(R)}$. Of course, the better the estimates are, the faster the algorithm is.

```
**Algorithm 4.25   OptTrailEst**
Let $\mathcal{E}$ be the set of saved characteristics.
**Algorithm** OptTrailEst
For each non-zero output difference $\beta_1$
        - go to "Round 1"
If a characteristic has been found ($\mathcal{E}$ is not empty), return $\mathcal{E}$ and $p_{\text{est}}$
    End algorithm

**Function** Round 1
$p_{\text{round }1} \leftarrow \max_\alpha \Pr(\alpha \to \beta_1)$
$\alpha_1 \leftarrow \alpha$ such that $\Pr(\alpha \to \beta_1) = p_{\text{round }1}$
$\mathcal{T} \leftarrow (\alpha_1, \beta_1)$
$\alpha_2 \leftarrow \pi(\beta_1)$
if $R > 2$, go to "Round 2", else go to "Last round"
    End of function (continue main loop)

**Function** Round $i$ $(2 \leq i < R)$
For each candidate $\beta_i$ for $\alpha_i$
        -$p_{\text{round }i} \leftarrow \Pr(\alpha_i \to \beta_i)$
        -$\mathcal{T} \leftarrow ((\alpha_1, \beta_1), \ldots, (\alpha_i, \beta_i))$
        - if $\prod_{j=1}^{i} p_{\text{round }j} \geq \frac{p_{\text{est}}}{p_{\text{best}(R-i)}}$ (i.e. it does not exceed the rank-$i$ bound), then
                -$\alpha_{i+1} \leftarrow \pi(\beta_i)$
                - if $i + 1 < R$ go to round $i + 1$
                - else, go to "Last round"
    End of function (continue Round $(i - 1)$ or Round 1() if $r = 2$)

**Function** Last round
$p_{\text{round }R} \leftarrow \max_\beta \Pr(\alpha_R \to \beta)$
$\beta_R \leftarrow \beta$ such that $\Pr(\alpha_R \to \beta) = p_{\text{round }R}$
If $\prod_{j=1}^{R} p_{\text{round }j} \geq p_{\text{est}}$, then
        - $\mathcal{T} \leftarrow ((\alpha_1, \beta_1), \ldots, (\alpha_R, \beta_R))$
        - $\mathcal{E} \leftarrow \mathcal{T}$ (the characteristics is saved)
        - $p_{\text{est}} = \prod_{j=1}^{R} p_j = \Pr(\mathcal{E})$
    End of function (continue "Round $(R - 1)$" or "Round 1()" if $R = 2$)
```

We now explain the ideas of the OptTrailEst algorithm.

Let us suppose that the condition on the rank-i bound in round i holds. Under this assumption, the algorithm goes through the tree of all round-R characteristics and saves the one with the highest probability in $\mathcal{E}$. When the program reaches the function "Round (i)" the current characteristic is $\mathcal{T} = ((\alpha_1, \beta_1), \ldots, (\alpha_{i-1}, \beta_{i-1}))$ and its probability is

$$\Pr(\mathcal{T}) = \prod_{j=1}^{i-1} (\alpha_j \to \beta_j) = \prod_{j=1}^{i-1} p_{\text{round }(j)}.$$

The input difference α_i for this round is equal to $\pi(\beta_{i-1})$ and for each candidate β_i for α_i, we extend $\mathcal{T}$ by (α_i, β_i) and we go to round $(i+1)$. When we reach the "Last round", it is easy to compute the output β_R that maximizes the probability of the last round. We save the characteristic only if it is better than $\mathcal{E}$. Let us now define what is meant by exceeding the rank-i bound.

Definition 4.26 Exceeding the bound
Let $\mathcal{T}$ be an i round characteristic with $i < R$. The probability $\Pr(\mathcal{T})$ of the characteristic **exceeds** the rank-i bound if

$$\Pr(\mathcal{T}) < \frac{p_{\text{est}}}{p_{\text{best}(R-i)}}$$

Rewriting "exceeding the bound" as $\Pr(\mathcal{T}) \cdot p_{\text{best}(R-i)} < p_{\text{est}}$ shows that even if the characteristic $\mathcal{T}$ is extended by an optimal $(R-i)-$round characteristic, its probability will still be lower than the estimate. This both shows the importance of the estimate and the pruning ability of the algorithm. If the estimate is chosen larger than the largest probability characteristic, $p_{\text{est}} > p_{\text{best}(R)}$, a characteristic expandable to an optimal R-round characteristic could be cut, but more importantly, no characteristic would be saved in "Last round". While we do not want to choose it larger than $p_{\text{best }(R)}$, choosing it too small allows for many characteristics to exceed the bound, making the algorithm noticeably slower. We want to choose it close to $p_{\text{best }(R)}$, allowing only a few characteristics to exceed the bound, lowering the computational complexity of the algorithm. With regard to pruning, we remove characteristics without removing any optimal characteristic. We now prove that the characteristic returned by the algorithm are optimal.

Lemma 4.27
Let $R \in \mathbb{N}$ and $r \in \mathbb{N}_{<R}$. Let $\mathcal{T}$ be a r-round characteristic whose probability exceeds the rank-r bound. Then there does not exist any R-round characteristic extension $\mathcal{T}'$ of $\mathcal{T}$ of probability greater than or equal to p_{est}.

Proof. Assume that $\mathcal{T}'$ extends $\mathcal{T}$ such that $\Pr(\mathcal{T}') \geq p_{\text{est}}$. Then, the probability of the $(R-r)$-round characteristic $\mathcal{T}'_{[r+1,R]}$ is

$$\begin{aligned}\Pr(\mathcal{T}'_{[r+1,R]}) &= \frac{\Pr(\mathcal{T})\Pr(\mathcal{T}'_{[r+1,R]})}{\Pr(\mathcal{T})} \\ &= \frac{\Pr(\mathcal{T}||\mathcal{T}'_{[r+1,R]})}{\Pr(\mathcal{T})} \\ &= \frac{\Pr(\mathcal{T}')}{\Pr(\mathcal{T})},\end{aligned}$$

where the second equality follows by the assumption of independence of the S-boxes. By assumption $\Pr(\mathcal{T}) < p_{\text{est}}/p_{\text{best }(R-r)}$ holds, and the strict inequality implies that $p_{\text{est}} > 0$. It now follows that

$$\frac{\Pr(\mathcal{T}')}{\Pr(\mathcal{T})} \geq \frac{p_{\text{est}}}{\Pr(\mathcal{T})} > \frac{p_{\text{est}}}{p_{\text{est}}/p_{\text{best }(R-r)}} = p_{\text{best }(R-r)}.$$

As by definition $p_{\text{best }(R-r)}$ is the highest probability any $(R-r)$-round characteristic can achieve, we end up with a contradiction. □

We now prove the validity of the algorithm.

Theorem 4.28
The algorithm OptTrailsEst returns a characteristic $\mathcal{E}$ such that $\Pr(\mathcal{E}) = p_{\text{best }(R)}$ if there exists an R-round characteristic of probability greater than p_{est}. In other words, if

$$p_{\text{est}} \leq p_{\text{best }(R)}$$

the algorithm returns an optimal characteristic, and if

$$p_{\text{est}} > p_{\text{best }(R)}$$

the algorithm does not return anything.

Proof. Suppose that the condition on the bound is removed, that is, $\Pr(\mathcal{T}) < p_{\text{est}}/p_{\text{best }(R)}$ does not have to hold. If $p_{\text{est}} < p_{\text{best }(R)}$, an optimal characteristic is saved in $\mathcal{E}$ in "Last Round", else $\mathcal{E}$ remains empty. The previous Lemma ensures that the pruning condition avoids only characteristics with probability strictly lower than p_{est}. The result still holds. □

There are further optimizations for the OptTrailsEst algorithm, these are described in [2], and we will now touch on one of them, namely the complexity of "Round 1".
The first step in "Round 1" requires that the algorithm goes through all non-zero output differences β_1, for which there are $2^{sm}-1$ possible. For ASCON , the lower bound complexity of the whole algorithm would then be $2^{5\cdot 64}-1 = 2^{320}-1$, which is slower than just checking all the 2^{128} possible keys. To solve this problem, we define a partition of the set of all non-zero differences and provide an effective way to test whether no difference in one part can be the beginning of an optimal characteristic.

Definition 4.29 Partitioned maximum S-box probabilities
The maximum probability of the n'th S-box is given as

$$p_{\text{SB}(n)} = \max_{a,b} \Pr(a \to b),$$

for $a, b \in \mathbb{F}_2^s \setminus \{0\}$. For every $n \in \mathbb{N}_{\leq m}$ sort the maximal probabilities of the S-boxes in decreasing order, equivalent to using a permutation ρ of $[\![1, m]\!]$ such that

$$p_{\text{SB}(\rho(n))} \geq p_{\text{SB}(\rho(n+1))}.$$

We omit the cases whenever a or b equals 0, as by Lemma 4.21 we have that $\Pr(0 \to 0) = 1$ and these would "cut in queue". We define the maximum probability of a one-round characteristic activating i S-boxes.

Definition 4.30
Let $p_{[n]\text{-SB}}$ denote the maximum probability of a one-round characteristic activating n S-boxes

$$p_{[n]\text{-SB}} = \max_{\alpha,\beta} \Pr(\alpha \to \beta),$$

for $\#\text{SB}(\alpha) = n$ and $\alpha, \beta \in \mathbb{F}_2^{sm} \setminus \{0\}$.

These definitions invite the following proposition.

Proposition 4.31
Let $n \in \mathbb{N}_{\leq m}$, then

$$p_{[n]\text{-SB}} = \prod_{i=i}^{n} p_{\text{SB}(\rho(i))}.$$

Proof. Let α be an input difference that activates n S-boxes, and let β be an output difference. We will prove that $Pr(\alpha \to \beta) \leq \prod_{i=i}^{n} p_{\text{SB}(\rho(i))}$. For each i in $[\![1, m]\!]$ define $q_i = \Pr(a_i \to b_i)$. By definition, we have

$$\Pr(\alpha \to \beta) = \prod_{i=1}^{m} \Pr(a_i \to b_i) = \prod_{i=1}^{m} q_i.$$

Let ρ' be a permutation of $[\![1, m]\!]$ such that

$$q_{\rho'(i)} \geq q_{\rho'(i+1)}.$$

As α activates n S-boxes, it must hold that $q_{\rho'(i)} = 0$ for all $i > n$, we have the following relation

$$\Pr(\alpha \to \beta) = \prod_{i=1}^{m} q_i = \prod_{i=1}^{m} q_{\rho'(i)} = \prod_{i=1}^{n} q_{\rho'(i)}.$$

We have that

$$\prod_{i=1}^{n} p_{\text{SB}(\rho(i))} \geq \prod_{i=1}^{n} p_{\text{SB}(\rho'(i))},$$

as $\prod_{i=1}^{n} p_{\text{SB}(\rho(i))}$ is the product of the n best probabilities for any $\alpha \in \mathbb{F}_2^{(sm)} \setminus \{0\}$, whereas $\prod_{i=1}^{n} p_{\text{SB}(\rho'(i))}$ relates to the specific α chosen. Also, using the same permutation ρ', we have that

$$\prod_{i=1}^{n} p_{\text{SB}(\rho'(i))} \geq \prod_{i=1}^{n} q_{\rho'(i)},$$

as $p_{\text{SB}(i)} \geq q_i$ for all $i \in [\![1, m]\!]$. The result follows as have proven that $\prod_{i=1}^{n} p_{\text{SB}(\rho(i))} \geq \Pr(\alpha \to \beta)$, which holds for arbitrary α whenever $\#\text{SB}(\alpha) = n$, so it must hold for $\max_{\alpha,\beta} \Pr(\alpha \to \beta)$ whenever $\#\text{SB}(\alpha) = n$. But as the maximum must have the highest probability, it holds that

$$\prod_{i=1}^{n} p_{\mathrm{SB}(\rho(i))} = \max_{\alpha,\beta} \Pr(\alpha \to \beta) = p_{[n]\text{-SB}}.$$

□

The inequalities $p_{[n]\text{-SB}} \leq \ldots \leq p_{[1]\text{-SB}}$, clearly holds, and the probability of an optimal one-round characteristic is given as

$$p_{\mathrm{best}(1)} = \max_{\alpha,\beta} \Pr(\alpha \to \beta) = p_{[1]\text{-SB}} = p_{\mathrm{SB}(\rho(1))}.$$

We precompute the probabilities $p_{\mathrm{SB}(i)}$ and $p_{[n]\text{-SB}}$ to optimize the search. The following theorem states that whenever $p_{[n]\text{-SB}}$ exceeds the rank-one bound, we only have to test the output differences β_1 for at most $n-1$ S-boxes. Before stating and proving the theorem, we argue the number of elements to be checked. Assuming the theorem is true and as $p_{[n]\text{-SB}}$ exceeds the rank-one bound, the output difference can activate any number from 1 to $n-1$ of m S-boxes. There are $\binom{m}{i}$ ways to choose i S-boxes and a total of $\sum_{i=1}^{n-1}\binom{m}{i}$ ways to choose any number less than n S-boxes. For each S-box, there are $2^s - 1$ possible elements to choose from. In total, there are

$$\sum_{i=1}^{n-1} \binom{m}{i} (2^s - 1)^i,$$

such differences, compared to the $2^{sm} - 1$ otherwise.

Theorem 4.32
Let n and n' be integers such that $1 \leq n \leq n' \leq m$. If $p_{[n]\text{-SB}}$ exceeds the rank-one bound, then there exists no R-round characteristic activating n' S-boxes in the first round with probability greater than or equal to p_{est}.

Proof. Assume that $p_{[n]\text{-SB}}$ exceeds the rank-one bound. Let $\mathcal{T}$ be a one-round characteristic activating n' S-boxes. By definition, we have that

$$\Pr(\mathcal{T}) \leq p_{[n']\text{-SB}}.$$

By definition of n and n' and by the discussion following proposition 4.31 it follows that $p_{[n']\text{-SB}} \leq p_{[n]\text{-SB}}$, and hence

$$\Pr(\mathcal{T}) \leq p_{[n]\text{-SB}}.$$

Then, if $\Pr(\mathcal{T})$ exceeds the rank-one bound, Lemma 4.27 ensures that there exists no R-round characteristic extending $\mathcal{T}$, with probability greater than or equal to p_{est}. □

By choosing $n' = n$, the result of the output difference activating at most $n-1$ S-boxes follows. The authors of [2] have run the improved algorithm for several SPNs, having a bit permutation as its linear layer. They have found that for $m = 16$ and $s = 4$, $p_{[4]\text{-SB}}$ always exceeds the bound. Also, they have found that, at most, only 2^{21} differences are to be tested compared to 2^{64} differences. We now present the optimized algorithm.

Algorithm 4.33 Optimized OptTrailEst

```
For $n$ form 1 to $m$
    - if $p_{[n]\text{-SB}}$ exceeds the rank-one bound, then exit the loop
    - else
        - for each output difference $\beta_1$ activating $n$ S-boxes
            - Go to "Round 1"
If a characteristic has been found ($\mathcal{E}$ is not empty), return $\mathcal{E}$ and $p_{\text{est}}$
  End algorithm
```

We have not yet explored how to find the p_{est}, and we now give a naive, and in no way optimal way to find p_{est}. The idea is to find the best $r-$round probability conditioned on the best $(r-1)-$round probability. It is easy to find the best $1-$round probability. With this naive estimate, we can run the algorithm.

4.3 Recovering key bits

We quickly touch on how to recover bits of the key. Assume that we have found a good differential characteristic for almost all of the rounds, say $r-1$ out of r rounds

$$\Pr(\alpha \to \beta) = p \text{ much greater than } 2^{-n}.$$

We can now encrypt many plaintext pairs with difference α, which in turn also gives us many ciphertext pairs. We guess part of the key for each ciphertext pair and compute backward from round r using this key. Then, we check whether the ciphertext pairs produce a pair of intermediate values with the predicted difference β. We give the key candidate a "thumbs up" if it has the predicted difference value. As this is repeated for many of the messages, the different key candidates get different numbers of "thumbs up". This happens as the intermediate results can happen either because it was the correct key, and we were looking at the actual intermediate value corresponding to the high probability difference, or because we randomly observed the correct difference. That is, we have a "good" and a "bad" scenario, one where we accidentally choose the wrong key and one where we choose the correct key. We choose the key with the most "thumbs ups". It turns out that the correct amount of pairs to check is approximately $1/p$. We have to run through all key candidates for each pair. This also means that we have to have a very good differential characteristic.

4.4 Differential-linear cryptanalysis

In this section, we touch on the subject of differential-linear cryptanalysis, a closely related attack to those of both differential cryptanalysis and linear cryptanalysis. While differential cryptanalysis examines the development of differences between two encrypted plaintexts through the encryption process and linear cryptanalysis examines the development of parities of subsets of the state bits through the encryption process of a single plaintext, differential-linear cryptanalysis combines these ideas. This holds under some *randomness assumptions* to be discussed later. We present the main ideas and leave the intricacies to the reader. We make use of the original paper on the subject [14] and a newer paper on the subject [3].

The idea of the attack is as follows. We exploit a differential characteristic with a high probability over a portion of the cipher (this probability would, of course, be significantly lower for the entire cipher). In the subsequent rounds, a linear approximation is applied, and it is expected that for each chosen plaintext pair, the probability of the linear approximation being valid for one plaintext but not the other will decrease for the correct key.

In less broad terms, it works as follows. Assuming we have a cipher E that can be decomposed into E_0 and E_1, then a high-probability differential for E_0 and a high-bias linear approximation for E_1 can be merged to form an effective distinguisher for the complete cipher E. Also, assume we have a differential $\Pr(\alpha^{\text{diff}} \to \beta^{\text{diff}}) = p$ for E_0 and a linear approximation

$$\bigoplus_{m=1}^{|P|} \alpha_m^{\text{lin}} P_m = \bigoplus_{m=1}^{|C|} \beta_m^{\text{lin}} C_m$$

with bias q for E_1. We remark that this notation is different from that used earlier when describing linear cryptanalysis, but the rewriting is easy to explain: rather than looking at how the masked plaintext and ciphertext influence the masked key, we look at how the masked plaintext directly influences the ciphertext.

Denote plaintexts by P, P', ciphertexts by C, C', and the values inbetween E_0 and E_1 by X, X'. Three approximations are combined for the differential-linear attack. The values $C\beta^{\text{lin}}$ and $C'\beta^{\text{lin}}$ correlates to $X\alpha^{\text{lin}}$ and $X'\alpha^{\text{lin}}$ by the linear approximation of E_1 and the values $X\alpha^{\text{lin}}$ and $X'\alpha^{\text{lin}}$ are correlated as consequence of the differential for E_0. This render $C\beta^{\text{lin}}$ correlated to $C'\beta^{\text{lin}}$.

We now state and discuss the earlier-mentioned randomness assumptions.

- In the cases where the differential is not satisfied, $X\alpha^{\text{lin}} = X'\alpha^{\text{lin}}$ holds in half of these (the cipher behaves randomly).

- The parts of the decomposed cipher, E_0 and E_1, are independent, and the bias of the linear approximations in E_1 is not affected by the fact that they are applied to two intermediate values that correspond to plaintexts with a fixed difference.

The first assumption often fails, so it is suggested that the bias of the whole approximation be checked experimentally whenever possible. There exist methods to express the exact bias of the approximation under the sole assumption that the two parts of the decomposed cipher are independent. We omit it here and refer the reader to [5].

Nonetheless, under these randomness assumptions, we can calculate the bias of $C\beta^{\text{lin}} = C'\beta^{\text{lin}}$ as follows. As $X\alpha^{\text{lin}} = X'\alpha^{\text{lin}}$ holds in half of the cases, this holds with a probability of $2p$. By the independence of the parts E_0 and E_1 and the Piling up Lemma 4.6, we end up with a bias of $2pq^2$.

Without going into any detail about the DES cipher or the exact method, we quickly account for the effectiveness of differential-linear cryptanalysis on a round-reduced version (8 rounds) of the DES cipher.

In their original paper [14] on differential-linear cryptanalysis, Langford and Hellman presented a method to recover 10-bits of the secret key using only 512 chosen plaintexts with a success probability of 80%. This success rate increases to 95% when using 768 chosen plaintexts.

4.5 Linear, differential and differential-linear cryptanalytic results of Ascon

This section presents the linear cryptanalysis, differential cryptanalysis, and differential-linear cryptanalysis results of the Ascon cipher. We comment on the results of [7] and [8]. Both the LAT and the DDT of Ascon can be found in [8].

4.5.1 Linear cryptanalysis and differential cryptanalysis

Firstly, they describe how to minimize the number of active S-boxes in differential characteristics for round-reduced versions of the Ascon permutation and argue that the model for linear cryptanalysis is "(...) essentially identical" [7, page 381]. Different models are made, and using these, they prove that the 3−round Ascon permutation has at least 15 differentially active S-boxes with a probability of $\leq 2^{-30}$ and at least 13 linearly active S-boxes with bias $\leq 2^{-14}$. The bounds on the number of active S-boxes are tight, but not necessarily those on probabilities. Using this, we can easily argue that the full 12−round initialization or finalization has at least 60 differentially active S-boxes with a probability of $\leq 2^{-120}$ and at least 52 linearly active S-boxes with bias $\leq 2^{-53}$. These bounds are almost certainly NOT tight, but it was not possible to derive bounds for more than 3 rounds applying the methods used.

"We could not find any differential and linear characteristics for more than 4 rounds with less than 64 active S-boxes." [8, page 31]

The non-tightness is also apparent in table 4.3 where the, at the time, best-known differential and linear characteristics for different round numbers are presented. When comparing the differential and linear results in table 4.3, we notice that for more than 2 rounds of the Ascon permutation, there are fewer linearly active S-boxes than differentially active S-boxes. This might infer that Ascon is more susceptible to linear cryptanalysis. Even so, the best 5−round linear characteristic found has more than 64 active S-boxes, and assuming the best possible bias of 2^{-2} for all active S-boxes, the attack complexity is already larger than 2^{128}. The differing methods in examining the cipher's behavior, one through differences and the other through linear approximations, result in varying numbers of activated S-boxes.

We refer to the paper for further clarification and the models used.

result	rounds	differential	linear
proof	1	1	1
	2	4	4
	3	15	13
heuristic	4	≤ 44	≤ 43
	≥ 5	≤ 78	≤ 67

Table 4.3: *Minimum number of active S-boxes for the* Ascon *permutation.*

4.5.2 Differential-linear cryptanalysis

When using differential-linear cryptanalysis for Ascon -128, we focus on the initialization phase, why the differences are only allowed in the nonce (s_4, s_5). Since the linear active bits must be observable, they have to appear in (s_0). We look at a round-reduced initialization phase (4 rounds).

Differential part

For the differential part, two differences are placed in the same S-box of round 1. With a probability of 2^{-2}, we have one active bit at the output of this S-box, and the linear layer ensures that 3 S-boxes are active in the second round. The differences in these three S-boxes occur at the same bit position in their inputs. In round 2, all three active S-boxes have the same output pattern, with two active bits, with a probability of 2^{-3}. After the linear layer, this leads to differences in 11 S-boxes in round 3.

Linear part

For the linear characteristic, we use a characteristic with one active S-box in round 4 and five active S-boxes in round 3. The bias of this linear characteristic is 2^{-8}. Additionally, we arrange the S-boxes so that the active S-boxes in round 3 do not overlap with the 11 S-boxes that have input differences. The resulting bias of the combined differential-linear characteristic is $2pq^2 = 2^{-20}$.

> *In practice, we are only interested in the bias of the output bit for the specific differences at the input. Due to the vast amount of possible combinations of differential and linear characteristics that achieve these requirements, we expect a much better bias.*
> [7, page 383-384]

Chapter 5 Confidentiality and authenticity under state recovery of Ascon

In this chapter, we state, analyze, and discuss results regarding the *confidentiality* and *authenticity* of Ascon in different settings. These results, as well as their proofs, can be found in [18]. Also, to underline the importance of the multiple key additions, we will state and prove the BAD Ascon theorem.
We start by defining the security model that will be used for this analysis.

5.1 Security model

We investigate the security of Ascon in a model where the transformations τ^a and τ^b are to be considered random permutations, that is $\tau^a, \tau^b \in \text{perm}(n)$.

> *"This is, as a matter of fact, the most crucial difference between our description and that of the actual* Ascon *: our analysis will demonstrate only resistance against generic attacks; actual attacks on* Ascon *may use internal properties of p,q and these are not captured by our security analysis."*
> [18, Page 6]
> (Note: p and q are the notation used for the permutations in the paper.)

We consider a multi-user setting of Ascon ; that is, an adversary can query up to $\mu \geq 1$ versions of the scheme simultaneously. We limit the adversary's ability, such that it cannot make a decryption query on any input of any result of any earlier encryption query.
We bound the complexity of the adversary, limiting it to $Q_{\mathcal{E}}$ encryption queries to encryption oracles, with a total amount of $\xi_{\mathcal{E}}$ blocks, $Q_{\mathcal{D}}$ decryption queries to decryption oracles, with a total amount of $\xi_{\mathcal{D}}$ *blocks*, Q_{τ^a} primitive queries to $\tau^{\pm a}$ and Q_{τ^b} primitive queries to $\tau^{\pm b}$

Definition 5.1 Construction queries and primitive queries
Construction queries involve interactions with the actual implementation of the cryptographic scheme, typically through oracles provided by the scheme.
Primitive queries involve the adversary interacting with the underlying cryptographic primitives directly, often modeled as ideal objects (like random oracles) in theoretical analysis.

Construction queries, such as encryption/decryption queries, rely on real-time responses ("online") from the oracles representing the cryptographic scheme. In these cases, the adversary sends a query and waits for an immediate response.
On the other hand, primitive queries do not require real-time interaction ("offline") with an oracle. The adversary can perform these queries independently, often assuming they can access the primitives as idealized objects (e.g., random functions or permutations).
The symbol $^{\pm}$ refers to the bidirectional query access.

Definition 5.2 Bidirectional query access
By bidirectional query access, we refer to the ability to obtain a state of the cipher either through an inverse query τ^{-a}, obtaining the state before the permutation, or through a forward query τ^a, obtaining the state after the permutation.

A block is counted as the number of τ^b evaluations that would be induced in the real world. This might be confusing at first, as a single evaluation request consisting of empty associated data and a single padded plaintext block $P_1||1||0^\kappa$ has zero τ^b evaluations, but for use in security proofs, this is the most logical definition.

For two randomized oracles $\mathcal{O}$ and $\mathcal{P}$, we define the advantage $\Delta_A(\mathcal{O};\mathcal{P})$ of an adversary A to distinguish between them as

$$\Delta_A(\mathcal{O};\mathcal{P}) = |\Pr(A^{\mathcal{O}} \to 1) - \Pr(A^{\mathcal{P}} \to 1)|,$$

where $\Pr(A^{\mathcal{P}} \to 1)$ means that the adversary succeeds.

Definition 5.3 Multi user security of Ascon

We define the multi-user security of Ascon against an adversary A as

$$\mathbf{Adv}^{\mu-\mathrm{ae}}_{\textsc{Ascon}}(A) = \Delta_A\left((\mathcal{E}^{\tau^a,\tau^b}_{K_j}, \mathcal{D}^{\tau^a,\tau^b}_{K_j})^{\mu}_{j=1}, \tau^{\pm a}, \tau^{\pm b}; (\$_j, \perp)^{\mu}_{j=1}, \tau^{\pm a}, \tau^{\pm b})\right),$$

where keys $K_1, \ldots, K_\mu \in \mathbb{F}_2^k$ and $\$_1, \ldots, \$_\mu$ are random function that for each new tuple (N, A, P) generates a random string of size $|P| + |N|$. The function $\perp$ returns a failure symbol $\perp$ for each query.

In the multi-user security of Ascon the two oracles are the encryption/decryption oracle $\left((\mathcal{E}^{\tau^a,\tau^b}_{K_j}, \mathcal{D}^{\tau^a,\tau^b}_{K_j})^{\mu}_{j=1}, \tau^{\pm a}, \tau^{\pm b}\right)$ and the random oracle $\left((\$_j, \perp)^{\mu}_{j=1}, \tau^{\pm a}, \tau^{\pm b}\right)$ consisting of random functions $\$_j$.

As commented on before, the nonce also plays a part in the security of Ascon , and if the nonce is repeated, security cannot be guaranteed.

Definition 5.4 Nonce-respecting

We say that an adversary A is nonce-respecting if every encryption query is made for a nonce N different from all nonces used in earlier encryption queries under the same key.

Note that the adversary is allowed to reuse a nonce in a decryption query or to reuse a nonce in an encryption query that was used in an earlier decryption query. We say that a scheme that is not nonce-respecting is nonce-misusing.

To analyze authenticity against both nonce-respecting and nonce-misusing adversaries, we separate the multi-user security into confidentiality and authenticity.

Definition 5.5 Seperated multi-user security of Ascon

We define the separated multi-user security of Ascon as

$$\mathbf{Adv}^{\mu-\mathrm{conf}}_{\textsc{Ascon}}(A) = \Delta_A\left((\mathcal{E}^{\tau^a,\tau^b}_{K_j})^{\mu}_{j=1}, \tau^{\pm a}, \tau^{\pm b}; (\$_j)^{\mu}_{j=1}, \tau^{\pm a}, \tau^{\pm b})\right),$$

$$\mathbf{Adv}^{\mu-\mathrm{auth}}_{\textsc{Ascon}}(A) = \Pr\left(A^{(\mathcal{E}^{\tau^a,\tau^b}_{K_j}, \mathcal{D}^{\tau^a,\tau^b}_{K_j})^{\mu}_{j=1}, \tau^{\pm a}, \tau^{\pm b}} \text{ forges}\right).$$

The adversary forges if it ever makes a query to one of its learning decryption oracles that is successful and that is not the result of an earlier encryption query.

5.2 Confidentiality of Ascon

We now present the results on the confidentiality of Ascon in the nonce-based setting, where the adversary cannot reuse nonce for different calls to a single encryption oracle, but it can reuse nonce under different keys.

Theorem 5.6
Let $a, b, k, m, c, r, \mu \in \mathbb{N}$ with $c + r = n$, $k + m \leq n$ and $m \leq k$, and consider the mode $\textsc{Ascon} = (\mathcal{E}, \mathcal{D})$. For any nonce-respecting adversary A making at most $Q_{\mathcal{E}}$ encryption queries with a total amount of $\xi_{\mathcal{E}}$ blocks, Q_{τ^a} primitive queries to τ^a, and Q_{τ^b} primitive queries to τ^b, such that $Q_{\tau^a} \leq \min\{2^{k-1}, 2^{c-1}\}$ and $Q_{\tau^b} \leq 2^{c-1}$,

$$\begin{aligned}\mathrm{Adv}^{\mu\text{-conf}}_{\mathrm{Ascon}}(A) \leq & \binom{\mu}{2}\frac{1}{2^k} + \frac{2\mu Q_{\tau^a}}{2^k} + \frac{2\mu Q_{\mathcal{E}}}{2^{n-m}} + \frac{(6m+8)Q_{\tau^a}}{2^{n-m}} \\ & + \frac{(\xi_{\mathcal{E}} + 2Q_{\mathcal{E}})^2}{2^n} + \frac{12Q_{\tau^b}(Q_{\mathcal{E}} + \xi_{\mathcal{E}})}{2^n} + \frac{(2Q_{\mathcal{E}} + Q_{\tau^a})^2}{2^n} \\ & + \frac{(\xi_{\mathcal{E}} + Q_{\tau^b})^2}{2^n} + \frac{16Q_{\tau^a}Q_{\mathcal{E}}}{2^n} + \frac{(6r+8)Q_{\tau^b}}{2^c} + \frac{(12r+16)Q_{\tau^a}}{2^c}.\end{aligned}$$

The proof of this theorem is long and cumbersome; therefore, we choose to explain some of the ideas of the proof, and we refer the reader to [18] Theorem 1, Lemma 2, and Lemma 3 for a comprehensive proof.
Idea of the proof.
The structure of the proof is as follows:

- Assumptions
- Replacement of permutations
- Definition of worlds
- Definition of transcript
- Definition of BAD events
- Indistinguishability of the ideal world and the real world
- Upper bounding the probability that any BAD event happens in the real world

We now explain the ideas of each of these.

Assumptions:
We start by making assumptions about the key, the permutations, and the adversary. Let $K_1, \ldots, K_\mu$ be μ randomly selected keys from $\mathbb{F}_2^k$, and let τ^a, τ^b be randomly selected permutations from $\mathrm{perm}(n)$, the set of all permutations on $\{0,1\}^n$. Also, let $\$_1, \ldots, \$_\mu \in \mathrm{func}(n)$, the set of all functions from $\{0,1\}^n$ to $\{0,1\}^n$ and let A be a nonce-respecting adversary.

Replacement of permutations:
The permutations τ^a and τ^b are replaced with random functions $f^{\pm}_{\tau^a}$ and $f^{\pm}_{\tau^b}$. This is done by considering them as lists of pairs of plaintexts and ciphertexts (X, Y). Whenever there

is a forward query X the response is Y if (X, Y) appears in any of the lists. If (X, Y) does not appear, some Y is randomly selected form $\{0,1\}^n$. The functions operate equivalently for inverse queries. It is then argued that if no collisions appear in these lists, that is, if there are not multiple answers for forward or inverse queries, then the permutations and the random functions are indistinguishable. To make this replacement there are $2Q_\varepsilon + Q_{\tau^a}$ queries to f_{τ^a} and $\xi_\varepsilon + Q_{\tau^b}$ queries to f_{τ^b} resulting in

$$\mathrm{Adv}^{\mathrm{conf}}_{\textsc{Ascon}}(A) \leq \Delta_A\left((\mathcal{E}_{K_j}^{f^{\pm}_{\tau^a}, f^{\pm}_{\tau^b}})_{j=1}^{\mu}, f^{\pm}_{\tau^a}, f^{\pm}_{\tau^b}; (\$_j)_{j=1}^{\mu}, f^{\pm}_{\tau^a}, f^{\pm}_{\tau^b}\right) + \frac{(2Q_\varepsilon + Q_{\tau^a})^2}{2^n} + \frac{(\xi_\varepsilon + Q_{\tau^b})^2}{2^n}.$$

Definition of worlds:
We denote the real world $W_R = \left((\mathcal{E}_{K_j}^{f^{\pm}_{\tau^a}, f^{\pm}_{\tau^b}})_{j=1}^{\mu}, f^{\pm}_{\tau^a}, f^{\pm}_{\tau^b}\right)$, consisting of the encryption structure associated with Ascon and the ideal world $W_I = \left((\$_j)_{j=1}^{\mu}, f^{\pm}_{\tau^a}, f^{\pm}_{\tau^b}\right)$ where the encryption function is random.
Transcript:
We define a transcript $\mathcal{T}$ which can be viewed as a log, saving the evaluations of $f^{\pm}_{\tau^a}$ and $f^{\pm}_{\tau^b}$.

Bad events:
We define the event **BAD** over the transcript $\mathcal{T}$. **BAD** splits into two separate events **GUESS** and **COL** which later are split further into multiple sub-events. We refrain from explicitly defining these events; instead, we explain their purpose. The exact definitions of the events can be found in [18] in the proof of Theorem 1.

We split **GUESS** into **GUESS**= $\mathbf{GUESS^{key}} \vee \mathbf{GUESS}_{\tau^a} \vee \mathbf{GUESS}_{\tau^b}$. The purpose of the event **GUESS** is to capture the case where the adversary *guesses* an intermediate state that was generated during a construction query. Event $\mathrm{GUESS}^{\mathrm{key}}$ corresponds to guessing the key, while for the other bad events, the subscript indicates if this guess is a guess for the outer primitive τ^a or the inner primitive τ^b.

COL handles collisions between the keys or between intermediate states in construction queries. It guarantees that permutation queries in the real world are unique and have not been previously seen or interacted with in the system (provided that no such state is *guessed* in a primitive query). **Col** is further split into sub-events **COL**= $\mathbf{COL^{key}} \vee \mathbf{COL^{aux}} \vee \mathbf{COL^{st}}$. $\mathbf{COL}^{\mathrm{key}}$ ensures that no collisions between two initial states $IV||K_j||N$ and $IV||K_{j'}||N'$ occurs, $\mathbf{COL}^{\mathrm{aux}}$ prevents collisions between an initial state and a state before the last f_{τ^a}-evaluation, and $\mathbf{COL}^{\mathrm{st}}$ handles the remaining collisions.

Indistinguishability of worlds:
We have the following statement:

*As long as **BAD** does not occur W_R and W_I are indistinguishable.*

We formally describe this as

$$\Delta_A(W_R; W_I) \leq \Pr(A^{W_R} \textit{ sets } \mathbf{BAD}).$$

The proof of this can be found in [18] Lemma 2.

Upper bound on the probability that a BAD event happens in the real world:
In this portion of the proof, each BAD event discussed above is upper bounded separately. We notice that each event, besides $\mathbf{COL}^{\text{key}}$, can happen at any point during any of the $Q_{\mathcal{T}^a} + 2Q_{\varepsilon} + Q_{\mathcal{T}^b} + \xi_{\varepsilon}$ $f_{\mathcal{T}^a}$ and $f_{\mathcal{T}^b}$ evaluations. It is necessary to condition each of the probabilities on the fact that any BAD event has not happened before. We will only argue some of the upper bounds and refer to [18] Lemma 3 for the rest.

Upper bounding $\mathbf{COL}^{\mathbf{key}}$:
For each of the μ versions of the scheme queried if 2 of the keys are the same, $\mathbf{COL}^{\text{key}}$ happens. There are $\binom{\mu}{2}$ ways this can happen and there are 2^k different keys, this gives us a probability of $\mathbf{COL}^{\text{key}}$ happening of

$$\Pr(\mathbf{COL}^{\text{key}}) = \binom{\mu}{2}\frac{1}{2^k}.$$

Upper bounding $\mathbf{GUESS}^{\mathbf{key}}$:
The second term $\frac{2\mu Q_{\mathcal{T}^a}}{2^k}$ corresponds to $\mathbf{GUESS}^{\text{key}}$. This event can only happen during an $f_{\mathcal{T}^{\pm a}}$ query. In the forward direction, this happens whenever the adversary guesses one of the μ random keys during any of the $Q_{\mathcal{T}^a}$ queries. Each failed guess eliminates one state of the set of possible candidates. This results in a probability of

$$\frac{\mu}{2^k - Q_{\mathcal{T}^a}}.$$

In the inverse direction, hitting $IV||K_j$ on its left most $n-m$ bits "*sets*" **BAD**, this happens with probability at most $\frac{\mu}{2^{n-m}}$. As there is a maximum of $Q_{\mathcal{T}^a}$ primitive queries to $f_{\mathcal{T}^a}$, the probability that the **BAD** event $\mathbf{GUESS}^{\text{key}}$ happens is:

$$Q_{\mathcal{T}^a}\left(\frac{\mu}{2^{n-m}} + \frac{\mu}{2^k - Q_{\mathcal{T}^a}}\right) \leq Q_{\mathcal{T}^a}\frac{2\mu}{2^k}.$$

Here, we use the fact that $Q_{\mathcal{T}^a} \leq 2^{k-1}$ and that $k+m \leq n$, such that the case of the inverse direction is also taken care of.

$\approx \square$

We now discuss the upper bound of $\mathrm{Adv}^{\mu\text{-conf}}_{\mathrm{Ascon}}(A)$ by examining some of the terms of the upper bound. Examining all the terms does not seem relevant to our discussion. We expect the upper bound to be close to zero, as our model of ASCON should appear random. The first term corresponds to the bound on $\mathbf{COL}^{\text{key}}$. While there are no restrictions on μ, it does not seem to matter as 2^k grows much faster than $\binom{\mu}{2}$. In particular, a quick calculation shows that for $\binom{\mu}{2}$ to equal 2^{128}, we would have to query more than $3.4 \cdot 10^{38}$ different schemes. We can safely imagine that we query nowhere near that many schemes.
For the second term $\frac{2\mu Q_{\mathcal{T}^a}}{2^k}$ we have the restriction $Q_{\mathcal{T}^a} \leq \min\{2^{k-1}, 2^{c-1}\}$. For the ASCON -128 scheme $2^{k-1} \leq 2^{c-1}$ ($c = 256$ and $k = 128$). At first glance, this might seem worrying as if $Q_{\mathcal{T}^a} = 2^{k-1}$ the second term equals μ. However, this scenario is mitigated by several factors.

- Practical Limits on Queries:
  In practical cryptographic applications, the number of queries $Q_{\mathcal{T}^a}$ is much smaller than 2^{k-1}. This means that in realistic scenarios, the value of $Q_{\mathcal{T}^a}$ is unlikely to

approach 2^{k-1}, thereby keeping the term $\frac{2\mu Q_{\tau^a}}{2^k}$ much smaller than μ.

- Adversary Resources:
  For an adversary to reach $Q_{\tau^a} = 2^{k-1}$, they would require an infeasible amount of computational resources and time, which makes it impractical for the adversary to achieve such a high number of queries.

We could discuss the terms of the upper bound for a long time, but we choose to stop and state that the remaining terms are all close to zero.

5.3 Authenticity in different nonce settings

We note that there exist results on the authenticity of Ascon in the nonce-misuse and nonce-respecting settings, but we choose to omit these for brevity. However, interested readers can refer to [18] Theorems 2 and 3 and their related Lemmas.

5.4 Authenticity Under State Recovery

A claim made by the Ascon architects is that even if an inner state of Ascon is leaked (the adversary recovers a state), mounting forgeries or recovering the key will still be hard. The idea is that whenever an inner state S is leaked to the adversary A, it may evaluate it in a forward/inverse direction using evaluations of τ^b. Still, it cannot go beyond the outer permutation evaluations due to the key addition mechanism. The secret values, i.e., the key and the tag, lie outside of these permutation evaluations, and hence, the architects' claim is valid. Authenticity is a weaker property than key recovery security as if an attacker can successfully recover the secret key, they can also forge messages. Therefore, to define security under state recovery, we aim for authenticity. Aside from oracle access, we assume that the adversary also has access to a leaky version of the scheme, where it not only gets the actual inputs but also some leakage function of each permutation call. The adversary succeeds if it can forge the challenge version of the scheme.
Leakages can only happen for the permutations τ^b, and the leakage function leaks the entire input and output of those permutations. Firstly, the fact that leakage only happens for permutations τ^b is supported by the observation that the permutations τ^a are stronger than the inner ones due to them being masked by the key additions. Secondly, the fact that the function leaks the entire input and output of the permutations is given out of generosity; that is, the adversary learns all the state information contrary to only part of it. We now define authenticity under state recovery using learning oracles $\mathcal{LE}$ and $\mathcal{LD}$, which respectively is an encryption oracle and a decryption oracle that additionally leak all input/output values of the evaluations of permutation τ^b.

Definition 5.7 Authenticity under state recovery
Let A be an adversary and let $\mathcal{LE}$ and $\mathcal{LD}$ be its leaky encryption and decryption learning oracles. The adversary forges if it ever makes a query to one of its learning decryption oracles that is successful and that is not the result of an earlier encryption query.

$$\mathbf{Adv}^{\mu\text{−sr−auth}}_{\textsc{Ascon}}(A) = \Pr\left(A^{(\mathcal{LE}^{\tau^a,\tau^b}_{K_j}, \mathcal{LD}^{\tau^a,\tau^b}_{K_j})^{\mu}_{j=1}, \tau^{\pm a}, \tau^{\pm b}} \text{ forges} \right).$$

5.5 Authenticity under state recovery of BAD Ascon

BAD Ascon refers to a modified version of Ascon that, among other things, omits all key additions except the first one (the addition of K to the state when adding the initialization vector IV). We will prove that this construction fails to achieve authenticity under state recovery.

Theorem 5.8
Let $a, b, k, m, c, r, \mu \in \mathbb{N}$ with $c + r = n$, $k + m \leq n$ and $m \leq k$ and consider BAD Ascon as described earlier. There exists an adversary A making $Q_{\mathcal{E}} = 1$ encryption query with a total amount of $\xi_{\mathcal{E}} = 0$ blocks, $Q_{\mathcal{D}} = 1$ decryption query with a total amount of $\xi_{\mathcal{D}} = 0$, $Q_{\tau^a} = 0$ primitive queries to τ^a, and $Q_{\tau^b} = 3$ primitive queries to τ^b, such that

$$\mathbf{Adv}^{\mu\text{−sr−auth}}_{\text{BAD Ascon}}(A) = 1.$$

Through a single learning query, one can obtain the key, and once the key is obtained, it becomes possible to execute a forgery. We include the proof below.

Proof. Consider an adversary A that recovers the key K_1 and uses it to forge a tag using this key. It operates as follows

1. A makes any encryption learning query with empty associated data and with a single padded plaintext block $\mathcal{LE}^{\tau^a,\tau^b}_{K_1}(N, P)$. It obtains (C, T) and the state S right after absorption of the plaintext P and before applying permutation τ^a.
2. The adversary queries $\tau^{-a}(S \oplus (P||0^{c-1}||1))$ to obtain $IV||K_1||N$ and extracts K_1 from it.
3. It then selects any tuple $(N', P') \neq (N, P)$, and computes $\mathcal{E}^{\tau^a,\tau^b}_{K_1}(N', P') = (C'; T')$ offline with two calls to τ^a.
4. It outputs forgery (N', C', T').

The forgery succeeds with probability 1. □

This theorem shows the importance of the additional key additions. Without these, it would be possible for an adversary to forge messages, breaking the authenticity of the scheme.

5.6 Authenticity under state recovery of Ascon

The main difference between the security of Ascon and BAD Ascon , concerning achieving authenticity under state recovery, is that the calls to the outer permutation τ^a are masked by the key K on both sides. Thus, even if an adversary learns all intermediate states, they cannot "pass through" the outer permutations τ^a. We now present the security of Ascon under state recovery.

Theorem 5.9
Let $a, b, k, m, c, r, \mu \in \mathbb{N}$ with $c + r = n$, $k + m \leq n$ and $m \leq k$, and consider the mode $\textsc{Ascon} = (\mathcal{E}, \mathcal{D})$. For any possibly nonce-misusing adversary A making at most $Q_\mathcal{E}$ encryption queries with a total amount of $\xi_\mathcal{E}$ blocks, $Q_\mathcal{D}$ decryption queries with a total amount of $\xi_\mathcal{D}$ blocks, Q_{τ^a} primitive queries to τ^a, and Q_{τ^b} primitive queries to τ^b such that $2Q_\mathcal{E} + 2Q_\mathcal{D} + Q_{\tau^a} \leq 2^{c-1}$, $\xi_\mathcal{E} + \xi_\mathcal{D} + Q_{\tau^a} \leq 2^{c-1}$ and $Q_{\tau^a} \leq 2^{k-1}$,

$$\begin{aligned}\mathrm{Adv}^{\mu\text{-sr-auth}}_{\textsc{Ascon}}(A) \leq & \frac{2Q_\mathcal{D}}{2m} + \binom{\mu}{2}\frac{1}{2k} + \frac{2\mu(Q_{\tau^a} + Q_\mathcal{E} + Q_\mathcal{D})}{2^k} \\ & + \frac{(12(c-k)+16)Q_{\tau^a}}{2k} \\ & + \frac{Q_{\tau^a}(6m+8)}{2^{n-m}} + \frac{4Q_{\tau^a}(Q_\mathcal{E} + Q_\mathcal{D})}{2n} \\ & + \frac{8(2Q_\mathcal{E} + 2Q_\mathcal{D} + 8Q_{\tau^b} + \mathcal{E} + \mathcal{D})^2}{2^c} \\ & + \frac{12Q_{\tau^a}(8Q_\mathcal{E} + 8Q_\mathcal{D} + Q_{\tau^b} + \mathcal{E} + \mathcal{D})}{2^c}.\end{aligned}$$

As for Theorem 5.6, this theorem is also long and cumbersome, but the ideas and the structure of the proof are much like that for Theorem 5.6. Therefore, we ignore discussing this and refer the reader to [18] Theorem 4 for a comprehensive proof.
The discussion of the upper bound is much like that for Theorem 5.6, so we omit it here.

Chapter 6 Experiment - the randomness of the permutations

This chapter consists of an experiment on the Ascon -128 permutations, checking whether they appear random.

To check randomness, we encrypt the same plaintext under several different keys. We choose these keys in a very specific way: we ONLY alter the first t bits of the key and go through all possible 2^t different keys. We then check whether changing the key changes the first t bits of the ciphertext. We do this by calculating the entropy of the distribution of the first t bits of the ciphertexts.
This gives us a number $0 \leq H(X) \leq t$, according to equation 3.1, which we can use to measure the uncertainty or randomness of the permutations. The experiment code has been written in Python 3.11 and can be found in Appendix A. The majority of the code consists of implementing the Ascon -128 cryptosystem. The implementation is based on [9], but multiple functions have been altered, and several functionalities have been added. The original work done in [9] was for several of the Ascon schemes, but the code included here has been altered to consider Ascon -128 only.

6.1 The experiment

This section explains the code and can be omitted.

The first section of the experiment defines "helper functions," which are used to shorten the code. Next, the Ascon -128 scheme is added, together with a function "demo_aead". The "demo_aead" function creates a random key and a random nonce, for which it encrypts the plaintext "ascon" with associated data "ASCON". It then prints, in byte from, the key, the nonce, the plaintext, the associated data, the ciphertext, the tag and the recieved message and also the byte-length of each of them. This function is added for convenience, making it easy to check whether the implementation works when altering the code.

The function "experiment(key_random,testlength,variant):"

Next, the function "experiment" is added. It takes as input a key, the amount of bits t which are to be altered during the key-altering process, and a variant of Ascon . It outputs the key, the ciphertext truncated to the first t bits, a list of the frequency of each ciphertext, the percentage of the amount of the 2^t possible ciphertexts that appear when trying all the 2^t different keys, and the entropy.

When writing the code, an issue converting different number bases arose, for which no solution has been found. This issue results in t having to be a multiple of 8 (the binary length of a byte). If this issue were solved, we would be able to calculate the entropy and the percentage of elements appearing for any number t less than or equal to the key length. As t has to be a multiple of 8, the largest value tested is 16, this, of course, is quite the limitation, but as it stands now, it is the largest possible number.

We now comment on what exactly happens in the function "experiment". We start by assigning the nonce a randomly generated value, which is kept constant during the experiment, such that we only check for one variable, which is the amount of bits of the key that we are altering. We choose to encrypt a single plaintext block and assign it the ASCII (American

Standard Code for Information Interchange) value "$b'\backslash x0eP\backslash xe3\backslash x8b;\backslash xd5\backslash x86'$". The plaintext is padded accordingly.
Next, the key, which is given as a byte object written in ASCII, is converted to a binary string using two functions key_random_hex, which converts the key from ASCII to hexadecimal, and then key_random_bin, converts the key from hexadecimal to its binary representation. An ordered list of all binary numbers "binary_numbers" less than 2^t is created together with an empty list "ciphertext_list", which will be used to store the ciphertexts.
In a loop of length 2^t, in round i, we replace the first t bits of the key with a corresponding binary number from "binary_list". The new key is converted back into bytes and used for encryption to calculate a ciphertext. Only the first t bits are considered and appended to the list "ciphertext_list". We use the function "ascon_encrypt_experiment" which differs from the ordinary encryption model by not calculating the tag, as this does not influence the ciphertext. This is done to reduce runtime.
After iterating over all possible 2^t keys, we find the frequency of each element in "ciphertext_list" and calculate the percentage of elements that appear. For this purpose, we convert the elements in "ciphertext_list" to integers and create a zero-list "frequency" of length 2^t. In a loop over "ciphertext_list", in iteration i we add one the i'th entry of "frequency". This leaves us with a list of the number of occurrences of each element in "ciphertext_list". Now, in a loop of length 2^t, whenever the entry of "frequency" is not equal to zero, we add one to a counter "amount_hit", and we find the percentage of elements hit by dividing this by 2^t. We then print the values established earlier and finally calculate the entropy in the function "entropy", which works by using the frequency list as its probability distribution.

We also define a function called "iterated_experiment", which, as the name suggests, is just the experiment iterated a number of times for multiple random keys.

6.2 Results

We now look at the results and analyze them. We test several things and add the results to tables for an easy overview.
First, we test for $t = 8$ and $t = 16$, keeping the key (the last $128 - t$ bit), the nonce, and the plaintext the same.

Unaltered key (16 bytes)	$b'\backslash xe50\backslash x00\backslash xdd\backslash x1ee\backslash x15\_K\backslash x91<I\backslash x0e\backslash xa5\backslash xa2\&'$	
Nonce (16 bytes)	$b'\backslash xff\backslash x1b\backslash xe0\backslash xb1\backslash xc1L\backslash x03;NI,;\hat{}\backslash\backslash\backslash x00\backslash x17'$	
Plaintext (7 bytes)	$b'\backslash x96\sim\backslash x16!\backslash x8ey\backslash x9e'$	
Percentage elements hit	$t = 8$ 64.453125%	$t = 16$ 63.200378%
Entropy	$t = 8$ 7.206259	$t = 16$ 15.171805

We notice that for both $t = 8$ and $t = 16$, the entropies do not attain theoretical value, and thus, we can conclude that the permutations are not truly random. If one would have to guess the key using a brute force method, they would still have to check about $2^{7.206259}$ and $2^{15.171805}$ different keys for $t = 8$ and $t = 16$ respectively. As the values are close to the theoretical value, it seems fair to conclude that the permutations appear random.
Next, we look at the percentage of elements hit, which is about $63\% - 64\%$ for both. At first glance, this might seem worrisome, but there is a logical explanation. Indeed, different keys generate different ciphertexts, but if you only look at the first portion, you should

expect to see the same sometimes. Let us look at the theoretical value with $t = 8$. One can view this as throwing 2^8 balls at 2^8 boxes under the assumption that the probability of hitting any two boxes is equal. We have the following problem

For a given box, what is the probability of hitting it?

When throwing a ball at the boxes, we have a $\frac{2^8-1}{2^8}$ of NOT hitting the desired box. Repeating this experiment 2^8 total times, we end up with

$$\left(\frac{2^8-1}{2^8}\right)^{2^8},$$

which is the total probability of NOT hitting any box. In turn, due to the law of total probability, the probability of hitting the box then becomes

$$1-\left(\frac{2^8-1}{2^8}\right)^{2^8} \approx 0.633.$$

As this holds for all boxes, we should assume to hit about 63% of the boxes when throwing 2^8 balls at 2^8 boxes. For $t = 16$ we have

$$1-\left(\frac{2^{16}-1}{2^{16}}\right)^{2^{16}} \approx 0.632.$$

These theoretical values lie close to those we observe. The following example also highlights this.

Example 6.1
In this example, we look at the first 2 bits of a bit-string. The first 2 bits can attain four different values: 00, 01, 10, and 11. Comparing this to the experiment, here we have four boxes and four balls. We have the following different possible distributions;

- Each box is hit once (1).
- One box is hit twice, and two boxes are hit once (12).
- Two boxes are hit twice (6).
- One box is hit thrice, and one box is hit once (12).
- One box is hit four times (4).

The number in parenthesis is the total number of different possible outcomes the scenario has. For example, in the scenario with two boxes hit twice, we can hit box number one and box number three twice, but we can also hit box number two and box number three twice, and so on. In the table below, we calculated the entropy for each scenario.

Scenario	Number of possibilities	Entropy
Each box is hit once	1	2
One box is hit twice, and two boxes are hit once	12	1.5
Two boxes are hit twice	6	≈ 0.8112
One box is hit thrice, and one box is hit once	12	1
One box is hit four times	4	0

One of the most probable scenarios also has the (next) highest entropy, namely, one box being hit twice and two boxes being hit once, with an entropy of 1.5.

Appendices

Appendix A Ascon -128 code

This chapter consists of the code used in Chapter 6 for experiments.

```
# === helper functions ===

def get_random_bytes(num):
    import os
    return to_bytes(os.urandom(num))

def zero_bytes(n):
    return n * b"\x00"

def to_bytes(l): # where l is a list or bytearray or bytes
    return bytes(bytearray(l))

def bytes_to_int(bytes):
    return sum([bi << ((len(bytes) - 1 - i)*8) for i, bi in enumerate(to_bytes(
        bytes))])

def bytes_to_state(bytes):
    return [bytes_to_int(bytes[8*w:8*(w+1)]) for w in range(5)]

def int_to_bytes(integer, nbytes):
    return to_bytes([(integer >> ((nbytes - 1 - i) * 8)) % 256 for i in range(
        nbytes)])

def rotr(val, r):
    return (val >> r) | ((val & (1<<r)-1) << (64-r))

def bytes_to_hex(b):
    return b.hex()

def entropy(frequency,testlength):
    import math
    entropy=0
    for i in range (2**testlength):
        if frequency[i] != 0:
            entropy += frequency[i]/2**testlength*math.log2(frequency[i]/2**
                testlength)
    print("Entropy: ", -entropy)
    # return entropy

# === Ascon AEAD encryption and decryption ===

def ascon_encrypt(key, nonce, associateddata, plaintext, variant="Ascon-128"):
    """
    Ascon encryption.
    key: a bytes object of size 16 (for Ascon-128)
    nonce: a bytes object of size 16 (must not repeat for the same key!)
    associateddata: a bytes object of arbitrary length
    plaintext: a bytes object of arbitrary length
    variant: "Ascon-128" (specifies key size, rate and number of rounds)
    returns a bytes object of length len(plaintext)+16 containing the
        ciphertext and tag
    """
    S = [0, 0, 0, 0, 0]
    k = len(key) * 8   # bits
    a = 12   # rounds
    b = 6 # rounds
    rate = 8 # bytes

    ascon_initialize(S, k, rate, a, b, key, nonce)
    ascon_process_associated_data(S, b, rate, associateddata)
    ciphertext = ascon_process_plaintext(S, b, rate, plaintext)
    tag = ascon_finalize(S, rate, a, key)
    return ciphertext + tag

def ascon_encrypt_experiment(key, nonce, associateddata, plaintext, variant="
    Ascon-128"):
    """
    Ascon encryption.
    key: a bytes object of size 16 (for Ascon-128)
    nonce: a bytes object of size 16 (must not repeat for the same key!)
    associateddata: a bytes object of arbitrary length
    plaintext: a bytes object of arbitrary length
    variant: "Ascon-128" (specifies key size, rate and number of rounds)
    """
    S = [0, 0, 0, 0, 0]
    k = len(key) * 8   # bits
    a = 12   # rounds
    b = 6 # rounds
    rate = 8 # bytes

    ascon_initialize(S, k, rate, a, b, key, nonce)
    # ascon_process_associated_data(S, b, rate, associateddata)
    ciphertext = ascon_process_plaintext(S, b, rate, plaintext)
    #tag = ascon_finalize(S, rate, a, key)
    return ciphertext


def ascon_decrypt(key, nonce, associateddata, ciphertext, variant="Ascon-128"):
    """
    Ascon decryption.
    key: a bytes object of size 16 (for Ascon-128y)
    nonce: a bytes object of size 16 (must not repeat for the same key!)
    associateddata: a bytes object of arbitrary length
    ciphertext: a bytes object of arbitrary length (also contains tag)
    variant: "Ascon-128"(specifies key size, rate and number of rounds)
    returns a bytes object containing the plaintext or None if verification
        fails
    """

    S = [0, 0, 0, 0, 0]
    k = len(key) * 8 # bits
    a = 12 # rounds
    b = 6 # # rounds
    rate = 8 # bytes

    ascon_initialize(S, k, rate, a, b, key, nonce)
    ascon_process_associated_data(S, b, rate, associateddata)
    plaintext = ascon_process_ciphertext(S, b, rate, ciphertext[:-16])
    tag = ascon_finalize(S, rate, a, key)
    if tag == ciphertext[-16:]:
        return plaintext
    else:
        return None

def ascon_initialize(S, k, rate, a, b, key, nonce):
    """
    Ascon initialization phase - internal helper function.
    S: Ascon state, a list of 5 64-bit integers
    k: key size (bits)
    rate: block size (bytes) (8 for Ascon-128)
    a: number of initialization/finalization rounds for permutation
    b: number of intermediate rounds for permutation
    key: a bytes object of size 16 (for Ascon-128)
    nonce: a bytes object of size 16
    returns nothing, updates S
    """
    iv_zero_key_nonce = to_bytes([k, rate * 8, a, b]) + zero_bytes(20-len(key))
         + key + nonce
    S[0], S[1], S[2], S[3], S[4] = bytes_to_state(iv_zero_key_nonce)

    ascon_permutation(S, a)

    zero_key = bytes_to_state(zero_bytes(40-len(key)) + key)
    S[0] ^= zero_key[0]
    S[1] ^= zero_key[1]
    S[2] ^= zero_key[2]
    S[3] ^= zero_key[3]
    S[4] ^= zero_key[4]


def ascon_process_associated_data(S, b, rate, associateddata):
    """
    Ascon associated data processing phase - internal helper function.
    S: Ascon state, a list of 5 64-bit integers
    b: number of intermediate rounds for permutation
    rate: block size in bytes (8 for Ascon-128, 16 for Ascon-128a)
    associateddata: a bytes object of arbitrary length
    returns nothing, updates S
    """
    if len(associateddata) > 0:
        a_padding = to_bytes([0x80]) + zero_bytes(rate - (len(associateddata) %
             rate) - 1)
        a_padded = associateddata + a_padding

        for block in range(0, len(a_padded), rate):
            S[0] ^= bytes_to_int(a_padded[block:block+8])
            ascon_permutation(S, b)

    S[4] ^= 1


def ascon_process_plaintext(S, b, rate, plaintext):
    """
    Ascon plaintext processing phase (during encryption) - internal helper
        function.
    S: Ascon state, a list of 5 64-bit integers
    b: number of intermediate rounds for permutation
    rate: block size in bytes (8 for Ascon-128)
    plaintext: a bytes object of arbitrary length
    returns the ciphertext (without tag), updates S
    """
    p_lastlen = len(plaintext) % rate
    p_padding = to_bytes([0x80]) + zero_bytes(rate-p_lastlen-1)
    p_padded = plaintext + p_padding

    # first t-1 blocks
    ciphertext = to_bytes([])
    for block in range(0, len(p_padded) - rate, rate):
        if rate == 8:
            S[0] ^= bytes_to_int(p_padded[block:block+8])
            ciphertext += int_to_bytes(S[0], 8)

        ascon_permutation(S, b)

    # last block t
    block = len(p_padded) - rate
    if rate == 8:
        S[0] ^= bytes_to_int(p_padded[block:block+8])
        ciphertext += int_to_bytes(S[0], 8)[:p_lastlen]
    return ciphertext


def ascon_process_ciphertext(S, b, rate, ciphertext):
    """
    Ascon ciphertext processing phase (during decryption) - internal helper function.
    S: Ascon state, a list of 5 64-bit integers
    b: number of intermediate rounds for permutation
    rate: block size in bytes (8 for Ascon-128)
    ciphertext: a bytes object of arbitrary length
    returns the plaintext, updates S
    """
    c_lastlen = len(ciphertext) % rate
    c_padded = ciphertext + zero_bytes(rate - c_lastlen) #zeropadding ot make sizes fit

    # first t-1 blocks
    plaintext = to_bytes([])
    for block in range(0, len(c_padded) - rate, rate):
        if rate == 8:
            Ci = bytes_to_int(c_padded[block:block+8])
            plaintext += int_to_bytes(S[0] ^ Ci, 8)
            S[0] = Ci

        ascon_permutation(S, b)

    # last block t
    block = len(c_padded) - rate
    if rate == 8:
        c_padding1 = (0x80 << (rate-c_lastlen-1)*8)
        c_mask = (0xFFFFFFFFFFFFFFFF >> (c_lastlen*8))
        Ci = bytes_to_int(c_padded[block:block+8])
        plaintext += int_to_bytes(Ci ^ S[0], 8)[:c_lastlen]
        S[0] = Ci ^ (S[0] & c_mask) ^ c_padding1
    return plaintext


def ascon_finalize(S, rate, a, key):
    """
    Ascon finalization phase - internal helper function.
    S: Ascon state, a list of 5 64-bit integers
    rate: block size in bytes (8 for Ascon-128)
    a: number of initialization/finalization rounds for permutation
    key: a bytes object of size 16 (for Ascon-128)
    returns the tag, updates S
    """
    S[rate//8+0] ^= bytes_to_int(key[0:8])
    S[rate//8+1] ^= bytes_to_int(key[8:16])
    S[rate//8+2] ^= bytes_to_int(key[16:] + zero_bytes(24-len(key)))
    ascon_permutation(S, a)

    S[3] ^= bytes_to_int(key[-16:-8])
    S[4] ^= bytes_to_int(key[-8:])
    tag = int_to_bytes(S[3], 8) + int_to_bytes(S[4], 8)
    return tag


def ascon_permutation(S, rounds=1):
    """
    Ascon core permutation for the sponge construction - internal helper
       function.
    S: Ascon state, a list of 5 64-bit integers
    rounds: number of rounds to perform
    returns nothing, updates S
    """
    for r in range(12-rounds, 12):
        # --- add round constants ---
        S[2] ^= (0xf0 - r*0x10 + r*0x1)
        # --- substitution layer (according to figure 4  (a))---
        S[0] ^= S[4]
        S[4] ^= S[3]
        S[2] ^= S[1]
        T = [(S[i] ^ 0xFFFFFFFFFFFFFFFF) & S[(i+1)%5] for i in range(5)]
        for i in range(5):
            S[i] ^= T[(i+1)%5]
        S[1] ^= S[0]
        S[0] ^= S[4]
        S[3] ^= S[2]
        S[2] ^= 0XFFFFFFFFFFFFFFFF
        # --- linear diffusion layer ---
        S[0] ^= rotr(S[0], 19) ^ rotr(S[0], 28)
        S[1] ^= rotr(S[1], 61) ^ rotr(S[1], 39)
        S[2] ^= rotr(S[2],  1) ^ rotr(S[2],  6)
        S[3] ^= rotr(S[3], 10) ^ rotr(S[3], 17)
        S[4] ^= rotr(S[4],  7) ^ rotr(S[4], 41)


# === some demo if called directly ===

def demo_print(data):
    maxlen = max([len(text) for (text, val) in data])
    for text, val in data:
        print("{text}:{align} 0x{val} ({length} bytes)".format(text=text, align
          =((maxlen - len(text)) * " "), val=bytes_to_hex(val), length=len(
          val)))

def demo_aead(variant):
    keysize = 16
    print("=== demo encryption using {variant} ===".format(variant=variant))

    # choose a cryptographically strong random key and a nonce that never
       repeats for the same key:
    key   = get_random_bytes(keysize) # zero_bytes(keysize)
    nonce = get_random_bytes(16)      # zero_bytes(16)

    associateddata = b"ASCON"
    plaintext      = b"ascon"
    ciphertext        = ascon_encrypt(key, nonce, associateddata, plaintext,
        variant)
    receivedplaintext = ascon_decrypt(key, nonce, associateddata, ciphertext,
        variant)

    #if receivedplaintext == None: print("verification failed!")

    demo_print([("key", key),
                ("nonce", nonce),
                ("plaintext", plaintext),
                ("ass.data", associateddata),
                ("ciphertext", ciphertext[:-16]),
                ("tag", ciphertext[-16:]),
                ("received", receivedplaintext),
               ])

def experiment(key_random,testlength,variant):
    assert testlength % 8 == 0, "testlength must be divisible by 8"

    nonce = b’\xff\x1b\xe0\xb1\xc1L\x03;NI,;^\\\x00\x17’ #get_random_bytes(16)
    associateddata = b’’ #get_random_bytes(16)
    plaintext      = b’\x96~\x16!\x8ey\x9e’ #get_random_bytes(7) #plaintext to
        be encrypted

    import binascii
    key_orginal=key_random
    key_random_hex = binascii.hexlify(key_random).decode(’utf-8’)#converte the
        key_random from ASCII to hex
    key_random_bin = bin(int(key_random_hex, 16))[2:].zfill(len(key_random) *
        8)#transforms the key_random_hex from hex to bin


    binary_numbers = [format(i, ’0{}b’.format(testlength)) for i in range(2**
        testlength)]#creates all binary numbers from 0 to 2**testlength
    ciphertext_list = []#list to store the ciphertexts
    for i in range(2**testlength):
        key_random_bin=binary_numbers[i]+key_random_bin[testlength:]#replaces
            the first testlength bits of the key_random_bin with the i’th
            binary number of length testlength
        key = b’’.join([int(key_random_bin[i:i+8], 2).to_bytes(1, ’big’) for i
            in range(0, len(key_random_bin), 8)])#transforms it to bytes to be
            used in ascon_encrypt

        ciphertext=ascon_encrypt_experiment(key, nonce, associateddata,
            plaintext,  variant)#encrypts the plaintext with the key
        ciphertext_list.append(ciphertext[:(testlength)//8])#appends the
            ciphertext to the list (only the first testlength bits are taken
            into account)



            #frequency of elements and percentage of elements hit
    ciphertext_list_int = [int.from_bytes(ciphertext, ’big’) for ciphertext in
        ciphertext_list]
    frequency = [0] * 2**testlength
    for num in ciphertext_list_int:
        frequency[num] += 1  # Accumulate the counts instead of replacing

    amount_hit = 0
    for i in range(2**testlength):
        if frequency[i] != 0:
            amount_hit += 1

    amount_hit_percentage=amount_hit/2**testlength


            #Printing the values
    print("Key: ",key_orginal)
    print("Ciphertext in bytes: ",ciphertext_list)
    print("Ciphertext in bytes sorted: ",sorted(ciphertext_list))
    print("Frequencies of elements sorted: ",frequency)
    print("Percentage of elements hit: ",amount_hit_percentage)
            #calculating and printing the entropy
    entropy(frequency,testlength)


def experiment_iterated(testlength,variant):
    for i in range(testlength):
        i=get_random_bytes(16)
        experiment(i,testlength,variant)


# demo_aead(’Ascon-128’)
experiment(b’\xe50\x00\xdd\x1ee\x15_K\x91<I\x0e\xa5\xa2&’,8,’Ascon-128’)
# experiment_iterated(8,’Ascon-128’)
```

This code outputs

```
Key:  b’\xe50\x00\xdd\x1ee\x15_K\x91<I\x0e\xa5\xa2&’
Ciphertext in bytes:  [b’\x1c’, b’W’, b’\xdf’, b’\x9a’, b"’", b’,’, b’c’, b’>’,
     b’\x8b’, b’N’, b’+’, b’\xcb’, b’\x0c’, b’n’, b’O’, b’)’, b’6’, b’\x14’, b’
    \xba’, b’\xd9’, b’\xd4’, b’\x8f’, b’K’, b’\xc8’, b’I’, b’d’, b’\x86’, b’C’,
     b’\x03’, b’\x07’, b’~’, b’J’, b’\xe2’, b’\xe4’, b’\xd5’, b’\x9b’, b’a’, b’
    \xc2’, b’@’, b’L’, b’\xc5’, b’\x9a’, b’c’, b’)’, b’\xb5’, b’p’, b’B’, b’\
    x1a’, b’\xa4’, b’R’, b’\xfa’, b’\x00’, b’\xde’, b’+’, b’\x9b’, b’\xa3’, b’q
    ’, b’\x18’, b’k’, b’\xc4’, b’\xd1’, b’\x1b’, b’W’, b’ ’, b’V’, b’\xc4’, b’\
    xfb’, b’\xe8’, b’\xe3’, b’l’, b’\xf6’, b’$’, b’?’, b’\xf1’, b’I’, b’\xe9’,
    b’e’, b’i’, b’\x13’, b’\xe5’, b’\x9f’, b’B’, b’8’, b’n’, b’\xdc’, b’\xa2’,
    b’>’, b’\xff’, b’\\’, b’7’, b’\x8b’, b’F’, b’\x1a’, b’W’, b’E’, b’\xd9’, b’
    \xca’, b’s’, b’\x00’, b’\xde’, b’\x1e’, b’\\’, b’\xd5’, b’J’, b’s’, b’\xf8’
    , b’\x01’, b’!’, b’\x98’, b’\xe1’, b’^’, b’\x8f’, b’\xbb’, b’\x89’, b’\xdd’
    , b’\xd9’, b’\x84’, b’\x96’, b’U’, b’\x01’, b’\x15’, b’4’, b’\x8e’, b’\x80’
    , b’m’, b’^’, b’\x90’, b’\x15’, b’(’, b’P’, b’\xeb’, b’\xdd’, b’\xfa’, b’\
    xbf’, b’\xbb’, b’(’, b’\xb3’, b’a’, b’\x04’, b’u’, b’_’, b’|’, b’\xf3’, b’\
    xdf’, b’,’, b’p’, b’\xd6’, b’\x15’, b’\x83’, b’c’, b’\x04’, b’\xc0’, b’\xee
    ’, b’f’, b’\xd8’, b’\xf9’, b’K’, b’\x95’, b’ ’, b’^’, b’\x8d’, b’\x15’, b"’
    ", b’\xf5’, b’?’, b’M’, b’C’, b’\x91’, b’\xcf’, b’\x07’, b’\xa8’, b’\x01’,
    b’\x0b’, b"’", b’c’, b’\xac’, b’\x1a’, b’\xd0’, b’5’, b’\x15’, b’\x87’, b’\
    x88’, b’\x13’, b’\x96’, b’\xf9’, b’_’, b’R’, b’\xa0’, b’T’, b’\xcf’, b’$’,
    b’T’, b’\xb9’, b’s’, b’\xc7’, b’\xe1’, b’\x8a’, b’\xe2’, b’\xd1’, b’\x0f’,
    b’\x11’, b’\x88’, b’|’, b’\xc3’, b’\xce’, b’\x08’, b’M’, b’\x16’, b’-’, b’q
    ’, b’\xed’, b’\x1d’, b’#’, b’\xd3’, b’\x13’, b’\xb8’, b’\r’, b’\xa4’, b’\
    x95’, b’W’, b’K’, b’\xb6’, b’(’, b’,’, b’\xcd’, b’\x10’, b’I’, b’\xde’, b’)
    ’, b’\xb8’, b’\x8f’, b’p’, b’}’, b’\x98’, b’\x1b’, b’\xed’, b’\xd8’, b’\x02
    ’, b’\xb7’, b’.’, b’\xe0’, b’\xca’, b’_’, b’\x03’, b’\xf6’, b’\xbe’, b’\x91
    ’, b’S’, b’\t’, b’>’, b’\xa1’, b’\xd6’, b’Q’, b’\x9b’, b’w’, b’v’]
Ciphertext in bytes sorted:  [b’\x00’, b’\x00’, b’\x01’, b’\x01’, b’\x01’, b’\
    x02’, b’\x03’, b’\x03’, b’\x04’, b’\x04’, b’\x07’, b’\x07’, b’\x08’, b’\t’,
     b’\x0b’, b’\x0c’, b’\r’, b’\x0f’, b’\x10’, b’\x11’, b’\x13’, b’\x13’, b’\
    x13’, b’\x14’, b’\x15’, b’\x15’, b’\x15’, b’\x15’, b’\x15’, b’\x16’, b’\x18
    ’, b’\x1a’, b’\x1a’, b’\x1a’, b’\x1b’, b’\x1b’, b’\x1c’, b’\x1d’, b’\x1e’,
    b’ ’, b’ ’, b’!’, b’#’, b’$’, b’$’, b"’", b"’", b"’", b’(’, b’(’, b’(’, b’)
    ’, b’)’, b’)’, b’+’, b’+’, b’,’, b’,’, b’,’, b’-’, b’.’, b’4’, b’5’, b’6’,
    b’7’, b’8’, b’>’, b’>’, b’>’, b’?’, b’?’, b’@’, b’B’, b’B’, b’C’, b’C’, b’E
    ’, b’F’, b’I’, b’I’, b’I’, b’J’, b’J’, b’K’, b’K’, b’K’, b’L’, b’M’, b’M’,
    b’N’, b’O’, b’P’, b’Q’, b’R’, b’R’, b’S’, b’T’, b’T’, b’U’, b’V’, b’W’, b’W
    ’, b’W’, b’W’, b’\\’, b’\\’, b’^’, b’^’, b’^’, b’_’, b’_’, b’_’, b’a’, b’a’
    , b’c’, b’c’, b’c’, b’c’, b’d’, b’e’, b’f’, b’i’, b’k’, b’l’, b’m’, b’n’, b
    ’n’, b’p’, b’p’, b’p’, b’q’, b’q’, b’s’, b’s’, b’s’, b’u’, b’v’, b’w’, b’|’
    , b’|’, b’}’, b’~’, b’\x80’, b’\x83’, b’\x84’, b’\x86’, b’\x87’, b’\x88’, b
    ’\x88’, b’\x89’, b’\x8a’, b’\x8b’, b’\x8b’, b’\x8d’, b’\x8e’, b’\x8f’, b’\
    x8f’, b’\x8f’, b’\x90’, b’\x91’, b’\x91’, b’\x95’, b’\x95’, b’\x96’, b’\x96
    ’, b’\x98’, b’\x98’, b’\x9a’, b’\x9a’, b’\x9b’, b’\x9b’, b’\x9b’, b’\x9f’,
    b’\xa0’, b’\xa1’, b’\xa2’, b’\xa3’, b’\xa4’, b’\xa4’, b’\xa8’, b’\xac’, b’\
    xb3’, b’\xb5’, b’\xb6’, b’\xb7’, b’\xb8’, b’\xb8’, b’\xb9’, b’\xba’, b’\xbb
    ’, b’\xbb’, b’\xbe’, b’\xbf’, b’\xc0’, b’\xc2’, b’\xc3’, b’\xc4’, b’\xc4’,
    b’\xc5’, b’\xc7’, b’\xc8’, b’\xca’, b’\xca’, b’\xcb’, b’\xcd’, b’\xce’, b’\
    xcf’, b’\xcf’, b’\xd0’, b’\xd1’, b’\xd1’, b’\xd3’, b’\xd4’, b’\xd5’, b’\xd5
    ’, b’\xd6’, b’\xd6’, b’\xd8’, b’\xd8’, b’\xd9’, b’\xd9’, b’\xd9’, b’\xdc’,
    b’\xdd’, b’\xdd’, b’\xde’, b’\xde’, b’\xde’, b’\xdf’, b’\xdf’, b’\xe0’, b’\
    xe1’, b’\xe1’, b’\xe2’, b’\xe2’, b’\xe3’, b’\xe4’, b’\xe5’, b’\xe8’, b’\xe9
    ’, b’\xeb’, b’\xed’, b’\xed’, b’\xee’, b’\xf1’, b’\xf3’, b’\xf5’, b’\xf6’,
    b’\xf6’, b’\xf8’, b’\xf9’, b’\xf9’, b’\xfa’, b’\xfa’, b’\xfb’, b’\xff’]
Frequencies of elements sorted:  [2, 3, 1, 2, 2, 0, 0, 2, 1, 1, 0, 1, 1, 1, 0,
    1, 1, 1, 0, 3, 1, 5, 1, 0, 1, 0, 3, 2, 1, 1, 1, 0, 2, 1, 0, 1, 2, 0, 0, 3,
    3, 3, 0, 2, 3, 1, 1, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 3, 2, 1,
    0, 2, 2, 0, 1, 1, 0, 0, 3, 2, 3, 1, 2, 1, 1, 1, 1, 2, 1, 2, 1, 1, 4, 0, 0,
    0, 0, 2, 0, 3, 3, 0, 2, 0, 4, 1, 1, 1, 0, 0, 1, 0, 1, 1, 1, 2, 0, 3, 2, 0,
    3, 0, 1, 1, 1, 0, 0, 0, 0, 2, 1, 1, 0, 1, 0, 0, 1, 1, 0, 1, 1, 2, 1, 1, 2,
    0, 1, 1, 3, 1, 2, 0, 0, 0, 2, 2, 0, 2, 0, 2, 3, 0, 0, 0, 1, 1, 1, 1, 1, 2,
    0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 1, 1, 1, 2, 1, 1, 2, 0, 0,
    1, 1, 1, 0, 1, 1, 2, 1, 0, 1, 1, 0, 2, 1, 0, 1, 1, 2, 1, 2, 0, 1, 1, 2, 2,
    0, 2, 3, 0, 0, 1, 2, 3, 2, 1, 2, 2, 1, 1, 1, 0, 0, 1, 1, 0, 1, 0, 2, 1, 0,
    0, 1, 0, 1, 0, 1, 2, 0, 1, 2, 2, 1, 0, 0, 0, 1]
Percentage of elements hit:  0.64453125
Entropy:  7.206259314400863
```

Bibliography

[1] M. D. Aagaard and N. Zidaric. Asic benchmarking of round 2 candidates in the nist lightweight cryptography standardization process. Cryptology ePrint Archive, Paper 2021/049, 2021. URL https://eprint.iacr.org/2021/049.

[2] A. Bannier, N. Bodin, and E. Filiol. Automatic search for a maximum probability differential characteristic in a substitution-permutation network. In *2015 48th Hawaii International Conference on System Sciences*, pages 5165–5174, 2015. doi: 10.1109/HICSS.2015.610.

[3] A. Bar-On, O. Dunkelman, N. Keller, and A. Weizman. Dlct: a new tool for differential-linear cryptanalysis. In *Advances in Cryptology–EUROCRYPT 2019: 38th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Darmstadt, Germany, May 19–23, 2019, Proceedings, Part I 38*, pages 313–342. Springer, 2019.

[4] G. Bertoni, J. Daemen, M. Peeters, and G. V. Assche. Permutation-based encryption, authentication and authenticated encryption. *DIAC - Directions in Authenticated Ciphers*, July 5-6, 2012.

[5] C. Blondeau, G. Leander, and K. Nyberg. Differential-linear cryptanalysis revisited. *Journal of Cryptology*, 30:859–888, 2017.

[6] C. S. Division and I. T. Laboratory. Announcing lightweight cryptography selection, Feb 2023. URL https://csrc.nist.gov/News/2023/lightweight-cryptography-nist-selects-ascon.

[7] C. Dobraunig, M. Eichlseder, F. Mendel, and M. Schläffer. Cryptanalysis of ascon. pages 371–387, 2015.

[8] C. Dobraunig, M. Eichlseder, F. Mendel, and M. Schläffer. Ascon v1.2 (authenticated encryption and hash). *Submission to NIST*, May 31, 2021.

[9] M. Eichlseder, V. Vissoultchev, and A. Faz. pyascon. https://github.com/meichlseder/pyascon/blob/master/ascon.py, 2023.

[10] O. Goldreich. *Foundations of Cryptography: Volume 2, Basic Applications*. Cambridge University Press, USA, 1st edition, 2009. ISBN 052111991X.

[11] H. Heys. A tutorial on linear and differential cryptanalysis. *Cryptologia*, 26(3):189–221, 2002.

[12] S. Hoffert. Domain separation, Dec 2016. URL https://seth.rocks/articles/domainseparation/.

[13] JacksonInfoSec. The linear approximation table (lat) of a s-box. https://www.youtube.com/watch?v=hHG_Ife-of0&ab_channel=JacksonInfoSec, October 2022.

[14] S. K.Langford and M. E. Hellman. Differential-linear cryptanalysis. In Y. G. Desmedt, editor, *Advances in Cryptology — CRYPTO '94*, pages 17–25, Berlin, Heidelberg, 1994. Springer Berlin Heidelberg.

[15] M. Matsui. Linear cryptanalysis method for des cipher. In *Workshop on the Theory and Application of of Cryptographic Techniques*, pages 386–397. Springer, 1993.

[16] K. McKay, L. Bassham, M. S. Turan, and N. Mouha. Report on lightweight cryptography, 2017-03-28 00:03:00 2017. URL https://tsapps.nist.gov/publication/get_pdf.cfm?pub_id=922743.

[17] F. Mendel, M. Schläffer, H. Luo, and R. Taori. Ascon: The lightweight cryptography standard for iot, March 2024.

[18] B. Mennink and C. Lefevre. Generic security of the ascon mode: On the power of key blinding. *IACR*, 2023. URL https://eprint.iacr.org/2023/796.

[19] N/A. Cryptographic competitions, Feb 2019. URL https://competitions.cr.yp.to/caesar-submissions.html.

[20] NIST. Submission requirements and evaluation criteria for the lightweight cryptography standardization process, August 2018.

[21] Wikipedia. Galois/counter mode, April 2024. URL https://en.wikipedia.org/wiki/Galois/Counter_Mode.