Analysis of Symmetric-Key Cryptosystems and Subset-Sum Problem

A thesis submitted

in Partial Fulfillment of the Requirements

for the Degree of

Doctor of Philosophy

by

Mahesh Sreekumar Rajasree

17111273

to the

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

INDIAN INSTITUTE OF TECHNOLOGY KANPUR

Dec, 2022

CERTIFICATE

It is certified that the work contained in the thesis titled **Analysis of Symmetric-Key Cryptosystems and Subset-Sum Problem**, by **Mahesh Sreekumar Rajasree**, has been carried out under my supervision and that this work has not been submitted elsewhere for a degree.

Prof. Manindra Agrawal
Department of Computer Science & Engineering
IIT Kanpur

Dec, 2022

DECLARATION

This is to certify that the thesis titled **Analysis of Symmetric-Key Cryptosystems and Subset-Sum Problem** has been authored by me. It presents the research conducted by me under the supervision of **Prof. Manindra Agrawal**. To the best of my knowledge, it is an original work, both in terms of research content and narrative, and has not been submitted elsewhere, in part or in full, for a degree. Further, due credit has been attributed to the relevant state-of-the-art and collaborations (if any) with appropriate citations and acknowledgements, in line with established norms and practices.

Name: Mahesh Sreekumar Rajasree
Programme: Doctor of Philosophy
Department: Computer Science & Engineering
Indian Institute of Technology Kanpur
Kanpur 208016

Dec, 2022

ABSTRACT

Name of student: **Mahesh Sreekumar Rajasree** Roll no: **17111273**

Degree for which submitted: **Doctor of Philosophy**

Department: **Computer Science & Engineering**

Thesis title: **Analysis of Symmetric-Key Cryptosystems and Subset-Sum Problem**

Name of Thesis Supervisor: **Prof. Manindra Agrawal**

Month and year of thesis submission: **Dec, 2022**

Cryptanalysis deals with the construction of tools and analysis of cryptosystems for subtle details and possible weaknesses that can be exploited to attack the security promised by these systems. Several cryptosystems have been proposed but later proved to be ineffective after rigorous examination. The Nazi used the Enigma machine, which was believed to be unbreakable, during World War II. Alan Turing and other mathematicians eventually succeeded in cracking the cryptosystem. More recently, a candidate for post-quantum public key encryption called SIKE was shown to be completely broken within an hour using a single core computer.

In this thesis, we present various attacks against practical symmetric key cryptosystems and algorithms for the subset-sum problem. First, we demonstrate new preimage attacks against round-reduced Keccak. Keccak was chosen as the new Standard Hashing Algorithm (SHA3) in 2012 superceding the previous standards. The attacks are based on constructing non-linear structures such that the quadratic terms are not spread across the whole state. We then present distinguishers in the weak key setting and key-recovery attacks against round-reduced TinyJAMBU and distinguishers for round-reduced Ascon. TinyJAMBU and Ascon are among the finalists for the NIST lightweight cryptography standardization competition. To identify distinguishers in the weak key setting for TinyJAMBU, we found a class of good cubes, whereas for the key-recovery attacks, we applied the monomial trail concept and MILP tool to find superpolies consisting of key variables. We exploit superpolies and monomial prediction to give better cube attacks for Ascon.

Finally, we present novel reductions and algorithms for variants of the subset sum problem. These variants include unique subset-sum, simultaneous subset-sum, unbounded subset-sum and subset-product problems, which are NP-hard. Some of these variants are used in building cryptographic schemes. Our algorithms use multivariate FFT, power series and number-theoretic techniques.

To my parents.

Acknowledgements

First and foremost, I would like to thank my advisor, Prof. Manindra Agrawal, for introducing me to the area of cryptography. He was always able to instil confidence in me so that I could work in the area of theoretical computer science.

I want to express my deepest gratitude to Prof. Shashank K. Mehta for his constant guidance and support. His enthusiasm and perseverance in solving problems motivated me to become a better researcher. Prof. Mehta is not only an excellent teacher but also a wonderful person who has always cared about my well-being throughout my research career.

I thank Prof. Hoda AlKhzaimi for giving me an opportunity to visit NYU Abu Dhabi. The research on Keccak at NYU with Prof. Hoda and Rajendra Kumar inspired me to continue my research in this area, which resulted in a single author paper. I appreciate Prof. Satya Lokam's support in paying for my numerous trips to Microsoft Research, Bengaluru. I was able to learn the fundamentals of public key cryptography thanks to these visits.

I owe Pranjal a great deal for his assistance during the COVID period. His positive and energetic approach to research inspired me to engage in our lengthy discussions with great enthusiasm and confidence. His care for my research is immeasurable, and it inspired us to write several student paper. I appreciate him putting me in touch with Prof. Santanu Sarkar, who inspired us to work on symmetric key cryptanalysis and provided us with a lot of insights in this field.

I thank Prof. Venkata Koppula for introducing me to public key cryptography. I was able to gain knowledge of subset-sum cryptography from the discussions with him.

I thank my friends Sumanta, Amit, Ramaiah, Prateek, Priyanka, Bhargav, Shanmugapriya, Muzafar, Krishnaprasad, Parvathy, Telma, Smriti and Aditi for the amazing discussions and wonderful friendships which made my life in IITK peaceful. Last but not least, I am extremely thankful to my parents and brother for bearing with me, helping me during my troubled times and having faith in me.

Contents

List of Tables

List of Figures

List of Publications

[KRA18] Rajendra Kumar, Mahesh Sreekumar Rajasree, and Hoda AlKhzaimi. "Cryptanalysis of 1-round KECCAK". In: *International Conference on Cryptology in Africa*. Springer. 2018, pp. 124–137.

[Raj19] Mahesh Sreekumar Rajasree. "Cryptanalysis of round-reduced Keccak using non-linear structures". In: *International Conference on Cryptology in India*. Springer. 2019, pp. 175–192.

[DR21] Pranjal Dutta and Mahesh Sreekumar Rajasree. "Efficient reductions and algorithms for variants of Subset Sum". In: *arXiv preprint arXiv:2112.11020* (2021).

[DR22] Pranjal Dutta and Mahesh Sreekumar Rajasree. "Algebraic Algorithms for Variants of Subset Sum". In: *Conference on Algorithms and Discrete Applied Mathematics*. Springer. 2022, pp. 237–251.

[DRS22a] Pranjal Dutta, Mahesh Sreekumar Rajasree, and Santanu Sarkar. "On the hardness of monomial prediction and zero-sum distinguishers for Ascon". In: *12th International Workshop on Coding and Cryptography (WCC 2022)*. 2022.

[DRS22b] Pranjal Dutta, Mahesh Sreekumar Rajasree, and Santanu Sarkar. "Weak-keys and key-recovery attack for TinyJAMBU". In: *Scientific Reports* 12.1 (2022), p. 16313.

[MR22] Shashank Mehta and Mahesh Rajasree. *On the Bases of Z^ n Lattice*. Tech. rep. EasyChair, 2022.

[DR23] Pranjal Dutta and Mahesh Sreekumar Rajasree. "Efficient Reductions and Algorithms for Subset Product". In: *Algorithms and Discrete Applied Mathematics: 9th International Conference, CALDAM 2023, Gandhinagar, India, February 9–11, 2023, Proceedings*. Springer. 2023, pp. 3–14.

Chapter 1

Introduction

A significant challenge while constructing a cryptosystem is ensuring it provides the desired level of security. Cryptanalysis deals with the construction of tools and analysis of cryptosystems for subtle details and possible weaknesses that can be exploited to attack the security promised by these systems. It is an essential area of study because it helps us understand the limitations of cryptographic systems and improve their security. Several cryptosystems have been proposed but later proved to be ineffective after rigorous examination.

One of the most famous events in the history of cryptanalysis was the breaking of the Enigma machine during World War II. Nazi Germany used the Enigma machine, which was believed to be unbreakable, during World War II for secure communication. The Allies, led by Alan Turing and other mathematicians, eventually succeeded in cracking the Enigma machine, which played a significant role in the outcome of the war.

NIST (National Institute of Standards and Technology) started the competition for SHA3 (Secure Hash Algorithm 3) due to concerns over the security of SHA-2, the previous standard for secure hash algorithms. Many attacks [RO05; WYY05; Coc07] against SHA-1 (a predecessor to SHA-2) were discovered, which raised concerns about the security of the entire SHA family of hash algorithms. Although SHA-2 was not directly affected by this vulnerability, it was clear that the security of the SHA family needed to be reevaluated. In February 2017, a team of researchers announced that they had successfully generated a collision for the SHA-1 hash function [Ste+17].

In more recent times, the development of quantum computers has raised concerns about the security of current cryptographic systems. Many cryptosystems that are provably secure in the classical setting have been shown to be vulnerable against a quantum adversary. For example, the 3-round Luby-Rackoff construction for pseudorandom functions is proven to be secure in the classical setting, whereas it was shown broken by a quantum attack [KM10].

Post-quantum cryptography, which aims to develop cryptographic systems that are secure against attacks by quantum computers, is an active area of research. One example of post-quantum cryptography is the SIKE protocol [Jao+17], which is a candidate for post-quantum public key encryption. However, recent research [CD22] has shown that SIKE can be completely broken within an hour using a single-core computer.

From the above examples, we can agree that cryptanalysis is an important area of study for building secure cryptosystems. The field of cryptography can be broadly classified into symmetric (secret) and asymmetric (public) key cryptography. In the public key setting, there are two types of keys - secret and public. All parties, including the adversaries, know the public key, whereas only a subset of the (honest) parties have access to the secret key. In the symmetric key setting, all the (honest) parties possess a secret key hidden from the adversaries.

Public key cryptosystems are built on the assumption that specific problems are computationally hard to solve. To be precise, these problems are believed to be in NP but not in P. For example, the first public key encryption schemes were based on *variants of* hard problems such as factoring, discrete logarithm, Subset Sum problem (SSUM), etc.

Cryptanalysis of public key cryptosystems mainly involves studying the hardness of such variants. For example, SSUM is a well-known NP-complete problem [GJ79], where given $(a_1, \ldots, a_n, t) \in \mathbb{Z}_{\geq 0}^{n+1}$, the problem is to decide whether there exists

$S \subseteq [n]$ such that $\sum_{i \in S} a_i = t$. One of the first public-key cryptosystems is the Merkle-Hellman cryptosystem [MH78] which is based on a variant of SSUM problem where the set $(a_1, \ldots, a_n)$ is a superincreasing sequence. This cryptosystem was shown to be broken by Shamir [Sha82]. Similarly, many SSUM based cryptosystems were found to be vulnerable to different attacks [Adl83; Odl84; LO85].

In recent years, provable-secure cryptosystems based on SSUM such as private-key encryption schemes [LPS10], tag-based encryption schemes [FMV16], etc., have been proposed. There are numerous improvements made in the algorithms that solve the SSUM problem in both the classical [Bri17; JW18; EM20; JVW21; BW21] and the quantum world [Ber+13a; HM18; LL19]. One of the first algorithms was due to Bellman [Bel57], who gave a $O(nt)$ time (*pseudo-polynomial* time) algorithm, which requires $\Omega(t)$ space.

In the secret key setting, the cryptosystems/primitives are based on problems related to multivariate polynomials. These systems consider the input (the secret key and the data) as binary strings and repeatedly apply a sequence of operations (addition, multiplication and shifting) to produce the output. Each bit in the output can be viewed as a high-degree multivariate polynomial with variables as the input bits.

Dinur and Shamir [DS09] proposed the *cube attack* against symmetric-key primitives with a secret key and a public input. It has since evolved into a universal tool for assessing the security of cryptographic primitives, and it has been successfully applied to a variety of symmetric primitives. Let the output bit of a cipher be an *unknown* Boolean polynomial $f(\boldsymbol{x}, \mathbf{k})$, over $\mathbb{F}_2$, where $\boldsymbol{x} = (x_1, \ldots, x_n)$ is a vector of public input variables, and $\mathbf{k} = (k_1, \ldots, k_m)$ is a vector of secret input variables, and $\mathbb{F}_2 = \{0, 1\}$ is the field containing 2 elements. Given a Boolean function $f(\boldsymbol{x}, \mathbf{k}) \in \mathbb{F}_2[\boldsymbol{x}, \mathbf{k}]$, let $\mathbf{I} = (i_1, i_2, \ldots, i_c) \subseteq [n]$ be the index subset such that $t = \prod_{i \in \mathbf{I}} x_i$. One can express $f(\boldsymbol{x}, \mathbf{k})$ as

$$f(\boldsymbol{x}, \mathbf{k}) \;=\; t \cdot p_{\mathbf{I}}(\boldsymbol{x}, \mathbf{k}) \;+\; q_{\mathbf{I}}(\boldsymbol{x}, \mathbf{k}) \;,$$

such that none of the monomials in $q_{\mathbf{I}}(\boldsymbol{x}, \mathbf{k})$ is divisible by t . The function $p_{\mathbf{I}}$ is called the *superpoly* of t in f. It can be easily verified that for any constants $b_j, \forall j \notin \mathbf{I}$,

$$\sum_{\substack{(x_{i_1}, x_{i_2}, \ldots, x_{i_c}) \in \mathbb{F}_2^c \\ x_j = b_j, \forall j \notin \mathbf{I}}} f(\boldsymbol{x}, \mathbf{k}) = p_{\mathbf{I}}(\boldsymbol{b}, \mathbf{k}) \ .$$

A *cube tester* basically computes the above summation (called *cube sum*) for a carefully chosen monomial t such that its superpoly $p_{\mathbf{I}}$ is equal to the constant zero. This serves as a distinguishing attack between f and a random polynomial.

To broaden the integral and higher-order differential distinguishers, Todo [Tod15] introduced the *division property*. Soon, division property based cube attacks became a prominent topic in the community. In [Hu+20], a new technique termed *monomial prediction* was proposed, which captures the algebraic basics of various attempts to improve the detection of division property. The goal was simple: detect a monomial $\boldsymbol{x}^{\mathbf{u}_1}$ in the product $\boldsymbol{y}^{\mathbf{u}_2}$ of any output bits of a vectorial Boolean function $\boldsymbol{y} = f(\boldsymbol{x})$. The monomial prediction approach was shown to be equivalent to the proposed three-subset division property without unknown subset [Hao+21].

Apart from these, there are numerous attacks developed against symmetric key primitives such as differential attacks [Knu94; MSK98], linear attacks [Mat93], algebraic attacks [Cou03; CM03], integral attacks [KW02], etc.

1.1 Our Contributions

In this thesis, we present a hardness result for Problem 1.1, new preimage attacks against Keccak, distinguishers in the weak key setting and key-recovery attacks against TinyJAMBU, distinguishers for Ascon and algorithms for variants of subset sum problem. In this section, we will briefly summarise these results, which originally appeared in a single author paper [Raj19]; and joint works with Dutta [DR22; DR23]; Dutta and Sarkar [DRS22a; DRS22b].

1.1.1 On the Hardness of Monomial Prediction

In Chapter 3, we study the hardness of the following problem for specific parameters.

Problem 1.1. *Given a composition $f := (f_1, \dots, f_n) := g_r \circ g_{r-1} \circ \dots g_0$, and a monomial m, where each $g_i : \mathbb{F}_2^n \longrightarrow \mathbb{F}_2^n$, is a quadratic function, decide whether m is a monomial in f_1.*

In [Kay10], Kayal considered this computational problem, known as 'monomial prediction', from a complexity-theoretic (hardness) point of view, which can be broadly (re)stated as follows: Given a blackbox access to an n-variate degree-d polynomial $f(\boldsymbol{x})$, over a finite field $\mathbb{F}$ and a monomial $\boldsymbol{x}^{\boldsymbol{e}} = x_1^{e_1} \cdots x_n^{e_n}$, determine the coefficient of $\boldsymbol{x}^{\boldsymbol{e}}$ in $f(\boldsymbol{x})$. Before Kayal, a similar search problem was also studied by Malod [Mal03] in his PhD thesis, from an algebraic complexity theoretic lens.

Kayal termed this problem as $\mathsf{CoeffSLP}$ and showed that it is $\#\mathsf{P}$-complete over integers (for a self-contained proof, see [Kay10, Appendix A]). We recall that $\#\mathsf{P}$ essentially captures the number of solutions of a given instance, and thus obviously a $\#\mathsf{P}$ problem must be *at least as hard* as the corresponding NP problem. However, for most of the stream ciphers, f can be thought as a composition of a bunch of *linear* and *quadratic* Boolean functions, and not *arbitrary* compositions. This makes the monomial prediction paradigm both theoretically and practically interesting! So, we restrict ourselves to this particular case and ask the complexity of the following problem, defined below.

We show that Problem 1.1 is $\oplus\mathsf{P}$-complete. Though it may sound obvious being 'theoretically' hard, the hardness proof is far from the obvious. To show the hardness formally, we define the following language.

$$\begin{aligned} L := \{(f, m) \mid \operatorname{coef}_m(f_1) = 1\,, &\text{ where } f = (f_1, \dots, f_{n_{r+1}}) = g_r \circ g_{r-1} \circ \dots g_0\,, \\ &\text{and } g_i : \mathbb{F}_2^{n_i} \longrightarrow \mathbb{F}_2^{n_{i+1}}\,, n_i \in \mathbb{N}\ \forall\ i \in [r+1], \text{with } n_0 = n, \\ &\text{monomial } m \in \mathbb{F}_2[x_1, \dots, x_n], \text{and } \deg((g_i)_j) \le 2\,\}\,. \end{aligned}$$

where $\text{coef}_m(f_1)$ denotes the coefficient of m in the polynomial f_1. In general, we will be working with $r, n_i = \mathsf{poly}(n)$ (so that one can think of the input complexity with respect to parameter n). Given (f, m) as an input where f is described by providing a compact representation of g_i's, we want to decide whether $(f, m) \in L$. Note that this may be 'easier' than solving a large system of multivariate polynomial equations over $\mathbb{F}_2$, and thus one cannot rigorously argue the hardness. However, we show that deciding $(f, m) \in L$ is $\oplus\mathsf{P}$-complete.

Theorem 1.2 ($\oplus\mathsf{P}$-completeness)**.** *Given a composition of quadratic functions f and a monomial m, deciding whether $(f, m) \in L$ is $\oplus\mathsf{P}$-complete.*

Remark 1.3. *The hardness proof can be adapted to work over a finite field $\mathbb{F}$ or integer ring $\mathbb{Z}$. In fact, over integers, one can show NP-hardness of the above problem by adapting the proof strategy of [Kay10]. Of course, a hardness proof over $\mathbb{Z}$ usually* does not *translate to very small characteristic fields like $\mathbb{F}_2$.*

1.1.2 Preimage Attacks on Round Reduced Keccak using Non-Linear Structures

In Chapter 4, we present new theoretical preimage attacks for Keccak-384 for 2,3,4 rounds and Keccak-512 for 2,3 rounds. Keccak designed by Guido Bertoni, Joan Daemen, Michaël Peeters, and Gilles Van Assche [Ber+09] became one of the candidates for SHA-3. It won the competition in October 2012 and was standardised as a "Secure Hash Algorithm 3" [Dwo15].

The Keccak hash family is based on the sponge construction [Ber+11b]. Its design was made public in 2008 and since then, it has received intense security analysis. In 2016, Guo et al. [GLS16] formalised the idea of linear structures and gave practical preimage attacks for 2 rounds Keccak-224/256. They also gave better preimage attacks for Keccak-384/512, all variants of 3-rounds Keccak as well as preimage attacks for 4-rounds Keccak-224/256. Li et al. [Li+17] improved the complexity of preimage attack for 3-rounds Keccak-256 by using a new type of structure called cross-linear structure. The best-known attacks for 3 and 4 rounds KECCAK-224/256 prior to our work are given by Li et al. [LS19] using a new technique called allocating approach, which consists of two phases - Precomputation phase and Online phase. They gave the first practical preimage attack for 3-rounds KECCAK-224. Theoretical preimage attacks for higher rounds on KECCAK are considered in [Ber10; Cha+14; MPS13]. Apart from the attacks mentioned above, there are several other attacks against KECCAK such as preimage attacks in [NRM11; MS13; KMS18; KRA18], collision attacks in [DDS12; DDS13; Köl+13; DDS14; SLG17] and distinguishers in [AM09; BCD11; Duc+12; JN15; GLS16].

The attacks in Chapter 4 are achieved by carefully constructing *non-linear structures* such that the quadratic terms are not spread throughout the whole state and the number of free variables in the system of equations is more. Table 1.1 summarises our contributions and the best theoretical preimage attacks up to four rounds. The space complexity in most of the attacks is constant unless it is explicitly mentioned.

1.1.3 Weak-Keys and Key-Recovery Attacks for TinyJAMBU

In Chapter 5, we give new insights into the structure of TinyJAMBU and present new attacks against TinyJAMBU. NIST [NIS18] has launched a process for soliciting, evaluating, and standardising lightweight cryptographic algorithms suited for use in limited contexts when the performance of current NIST cryptographic standards is unacceptably low. In August 2018, NIST issued a call for algorithms to be considered for lightweight cryptography standards. There were initially 57 submissions. NIST picked 32 candidates in the second round of trimming on August 30 2019. On March 29, 2021, NIST released ten candidates following the third round of pruning. TinyJAMBU is one of these candidates. TinyJAMBU employs a keyed-permutation based on an NLFSR that computes only a single NAND gate as a non-linear component per round. It is a small variant of the JAMBU mode, which is the smallest block cipher authenticated encryption mode in the CAESAR competition [CAE], and it was selected for the third round of the competition.

Rounds	**Instances**	**Complexity**	**References**
1	224 256 384 512	Practical	[KRA18]
2	224 256 384 512	Practical Practical 2^{129} 2^{384}	[GLS16]
2	384	Time 2^{89} Space 2^{87}	[KMS18]
2	384 512	$\mathbf{2^{113}}$ $\mathbf{2^{321}}$	Section 4.3 Section 4.2
3	224 256	2^{38} 2^{81}	[LS19]
3	384 512	2^{322} 2^{482}	[GLS16]
3	384 512	$\mathbf{2^{321}}$ $\mathbf{2^{475}}$	Section 4.5 Section 4.6
4	224 256	2^{207} 2^{239}	[LS19]
4	384 512	2^{378} 2^{506}	[MPS13]
4	384	$\mathbf{2^{371}}$	Section 4.7

Table 1.1: Summary of preimage attacks for KECCAK

In [WH19; WH21], the designers of TinyJAMBU analysed the system against various attacks. They analysed the differential properties of keyed permutation $\mathcal{P}_n$, which is the core component of TinyJAMBU. They demonstrated that the differential probability for $\mathcal{P}_{640}$ rounds is *insignificant*. They also studied the linear properties of $\mathcal{P}_n$ claiming that 32 sized cube attacks are ineffective against $\mathcal{P}_n$, when

$n \geq 512$.

Using Mixed Integer Linear Programming (MILP), the designers [WH19] count the least number of active AND gates to find differential and linear trails. This counting technique, however, disregards the inter-dependency between several AND gates. This flaw was identified by Saha et al. [Sah+20]. While the designers suggested the 384-round differential trail with probability 2^{-80} by regarding each AND gate independently, [Sah+20] confirmed that there is no such trail by taking into account the dependency. Further, they proposed a forgery attack with complexity $2^{62.68}$ on 338 rounds and a differential trail with probability $2^{-70.68}$ for 384 rounds using their refined model.

Recently, Teng et al. [Ten+21] looked into the TinyJAMBU cipher's resistance to cube attacks. They showed key-recovery attack for 428 rounds and distinguishing attack for 438 rounds using small size cubes.

In Chapter 5, we study TinyJAMBU from three important and different contexts – (i) the *weak-key* setting, (ii) understanding the *exact* degree of the feedback polynomial in the nonce variables (iii) the *key-recovery* attacks. Most significantly, this is the *first* time that the concept of *monomial trail*, introduced in [Hu+20], and the MILP tool have been employed together in analyzing TinyJAMBU.

1. We begin by studying TinyJAMBU in the *weak-key* setting. Since the core component of TinyJAMBU is the keyed permutation $\mathcal{P}_n$, we analyse the structure of $\mathcal{P}_n$ for any weakness. Using this, we present a class of *good* cubes which will help us in our attack. We show that there are at least 2^{108} keys for which TinyJAMBU can be distinguished from a random source for up to 476 rounds.

2. Next, we focus on determining the *exact* degree in the nonce variables of the feedback polynomial. Using the monomial trail concepts [Hu+20] and MILP tool, we demonstrate that after 381 rounds, the degree equals 32. This is the *first* time one has studied the exact degree of the feedback polynomial of TinyJAMBU. Moreover, the exact degree from our experiments implies that TinyJAMBU is secure against cube attacks with 32-dimension cubes after 445

rounds.

3. Finally, we present key-recovery attacks for reduced round TinyJAMBU. We again use the monomial trail concept and MILP tool to find superpolies consisting of key variables only for 440 rounds. This leads to an *improved* key-recovery attack, which is better than the 428 round key-recovery attack presented by Teng et al. [Ten+21], in their recent result published in Scientific Reports - Nature.

1.1.4 New Zero-Sum Distinguishers for Ascon

Ascon [Dob+20] is one of the elegant designs of authenticated encryption with associated data (AEAD) that was selected as the first choice for lightweight applications in the CAESAR competition, which also has been submitted to NIST lightweight cryptography standardization [NIS18]. On February 7, 2023, NIST announced the selection of the Ascon family for lightweight cryptography standardisation. It has been in the literature for a while, however, there has been no successful AEAD that is secure and at the same time lighter than Ascon.

In Chapter 6, we exploit superpolies and monomial prediction to give better cube attacks for Ascon. Most importantly, the attacks are *not restricted* to Ascon, and can be used in other cryptosystems as well since most cryptosystems, e.g., SHA3 [Dwo15], TinyJambu [WH21], etc. can be thought as a composition of quadratic functions. In Section 6.2, we present a new zero-sum distinguisher for 5-round Ascon with complexity 2^{14} which *improves* the best-known cube distinguishers [Roh+21] by a factor of 2^2.

1.1.5 Efficient Reductions and Algorithms for Variants of Subset Sum

In Chapter 7, we give efficient reductions and (time, space) algorithms for many variants of subset sum which we define below.

Problem 1.4 (Subset Sum problem (SSUM)). *Given $(a_1, \ldots, a_n, t) \in \mathbb{Z}_{\geq 0}^{n+1}$, the subset sum problem is to decide whether t is a realisable target with respect to $(a_1, \ldots, a_n)$, i.e., there exists $S \subseteq [n]$ such that $\sum_{i \in S} a_i = t$. Here, n is called the* size, *t is the* target *and any $S \subseteq [n]$ such that $\sum_{i \in S} a_i = t$ is a* realisable set *of the subset sum instance.*

Assumptions. Throughout the paper, we assume that $t \geq \max a_i$ for simplicity.

Problem 1.5 ($k - \mathsf{SSSUM}$). *Given $(a_1, \ldots, a_n, t) \in \mathbb{Z}_{\geq 0}^{n+1}$, the k-solution* SSUM *($k - \mathsf{SSSUM}$) problem asks to output all $S \subseteq [n]$ such that $\sum_{i \in S} a_i = t$ provided with the guarantee that the number of such subsets is at most k.*

We denote $1 - \mathsf{SSSUM}$ as unique Subset Sum problem (uSSSUM). In stackexchange, a more restricted version was asked where it was assumed that $k = 1$, for *any* realisable t. Here we just want $k = 1$ for some fixed target value t and we do not assume anything for any other value t'.

Problem 1.6 ($\mathsf{Hamming} - k - \mathsf{SSSUM}$). *Given an instance of the $k - \mathsf{SSSUM}$, say $(a_1, \ldots, a_n, t) \in \mathbb{Z}_{\geq 0}^{n+1}$, with the promise that there are at most k-many $S \subseteq [n]$ such that $\sum_{i \in S} a_i = t$,* $\mathsf{Hamming} - k - \mathsf{SSSUM}$ *asks to output all the hamming weights (i.e., $|S|$) of the solutions.*

Problem 1.7 ($\mathsf{Subset\ Product}$). *Given $(a_1, \ldots, a_n, t) \in \mathbb{Z}_{\geq 1}^{n+1}$, the* $\mathsf{Subset\ Product}$ *problem asks to decide whether there exists an $S \subseteq [n]$ such that $\prod_{i \in S} a_i = t$.*

Problem 1.8 ($\mathsf{SimulSubsetSum}$). *Given subset sum instances $(a_{1j}, \ldots, a_{nj}, t_j) \in \mathbb{Z}_{\geq 0}^{n+1}$, for $j \in [k]$, where k is some parameter, the Simultaneous Subset Sum problem (in short,* $\mathsf{SimulSubsetSum}$*) asks to decide whether there exists an $S \subseteq [n]$ such that $\sum_{i \in S} a_{ij} = t_j, \forall j \in [k]$.*

Time-efficient algorithms for variants of Subset Sum

Our first theorem gives an efficient pseudo-linear $\tilde{O}(n + t)$ time *deterministic* algorithm for Problem 1.6, for constant k.

Theorem 1.9 (Algorithm for hamming weight)**.** *There is a $\tilde{O}(k(n+t))$-time deterministic algorithm for* $\mathsf{Hamming} - k - \mathsf{SSSUM}$.

▶ Remark (Optimality). We emphasize the fact that Theorem 1.9 is likely to be *near*-optimal for bounded k, due to the following argument. An $O(t^{1-\epsilon})$ time algorithm for $\mathsf{Hamming} - 1 - \mathsf{SSSUM}$ can be directly used to solve $1 - \mathsf{SSSUM}$, as discussed above. By using the *randomized* reduction (Theorem 7.1), this would give us a randomized $n^{O(1)}t^{1-\epsilon}$-time algorithm for SSUM. But, in [Abb+19] the authors showed that SSUM does not have $n^{O(1)}t^{1-\epsilon}$ time algorithm unless the Strong Exponential Time Hypothesis (SETH) is false.

Theorem 1.9 is better than the trivial. Consider the usual 'search-to-decision' reduction for subset sum: First try to include a_1 in the subset, and if it is feasible then we subtract t by a_1 and add a_1 into the solution, and then continue with a_2, and so on. This procedure finds a single solution, but if we implement it in a recursive way then it can find all the k solutions in $k \cdot n \cdot$(time complexity for decision version) time; we can think about an n-level binary recursion tree where all the infeasible subtrees are pruned. Since the number of solutions is bounded by k, choosing a prime $p > n + t + k$ suffices in [JW18], to make the algorithm deterministic. Thus, the time complexity of the decision version is $\tilde{O}((n+t)\log k)$. Hence, from the above, the search complexity is $\tilde{O}(kn(n+t))$ which is *worse* than Theorem 1.9.

Theorem 1.10 (Time-efficient algorithm for Subset Product)**.** *There exists a randomized algorithm that solves* Subset Product *in $\tilde{O}(n + t^{o(1)})$ expected-time.*

Remarks. 1. The result in the first part of the above theorem is reminiscent of the $\tilde{\mathrm{O}}(n+t)$ time randomised algorithms for the subset sum problem [Bri17; JW18], although the time complexity in our case is the expected time, and ours is better.

2. The expected time is because to factor an integer t takes expected $\exp(O(\sqrt{\log(t)\log\log(t)}))$ time [LP92]. If one wants to remove expected time analysis (and do the worst case analysis), the same problem can be solved in $\tilde{O}(n^2 + t^{o(1)})$ randomised time. For details, see the end of Section 7.2.2.

3. While it is true that Bellman's algorithm gives $O(nt)$ time algorithm, the state-space of this algorithm can be improved to (expected) $nt^{o(1)}$-time for the Subset Product, using a similar dynamic algorithm with careful analysis. For details, see Appendix A.2.3.

Space-efficient algorithms for variants of Subset Sum

Theorem 1.11 (Algorithms for finding solutions in low space)**.** *There is a* poly(knt)-*time and* $O(\log(knt))$*-space deterministic algorithm which solves* $k-\mathsf{SSSUM}$*.*

▶ Remark. When considering low space algorithms outputting multiple values, the standard assumption is that the output is written onto a one-way tape which *does not* count into the space complexity; so an algorithm outputting $kn\log n$ bits (like in the above case) could use much less working memory than $kn\log n$; for a reference see McKay and Williams [MW18].

Theorem 1.11 is better than the trivial. Let us again compare with the trivial search-to-decision reduction time algorithm, as mentioned in Section 1.1.5. For solving the decision problem in low space, we simply use Kane's $O(\log(nt))$-space poly(nt)-time algorithm [Kan10]. As explained (and improved) in [JVW21], the time complexity is actually $O(n^3t)$ and the extra space usage is $\tilde{O}(n)$ for remembering the recursion stack. Thus the total time complexity is $O(kn^4t)$ and it takes $\tilde{O}(n)+O(\log t)$ space. While Theorem 1.11 takes $O(\log(knt))$ space and poly(knt) time. Although our time complexity is worse[1], when $k \le 2^{O((n\log t)^{1-\epsilon})}$, for $\epsilon > 0$, our space complexity is *better*.

Theorem 1.12. *(Algorithm for* Subset Product*)* Subset Product *can be solved deterministically in* $O(\log^2(nt))$ *space and* poly(nt)*-time.*

▶ Remark. We *cannot* directly invoke the theorem in [Kan10, Section 3.3] to conclude, since the reduction from Subset Product to SimulSubsetSum requires $O(n\log(nt))$

[1] Theorem 1.11 is *not about* time complexity; as long as it is pseudo-polynomial time it's ok.

$$\begin{tikzcd}
& \mathsf{HWSSUM} \arrow[d, shift right] & \\
\mathsf{SimulSubsetSum} \arrow[r, shift right] & \mathsf{SSUM} \arrow[l, shift right] \arrow[u, shift right] \arrow[r, shift left] \arrow[d, shift right] & \mathsf{SSSUM} \arrow[l, shift left] \\
& \mathsf{UBSSUM} \arrow[u, shift right] &
\end{tikzcd}$$

Figure 1.1: Reductions among variants of the Subset Sum problem

space. Essentially, we use the same identity lemma as [Kan10] and carefully use the space; for details see Section 7.3.2.

Reductions among variants of Subset Sum

Using a pseudo-prime-factorisation decomposition, we show that given a target t in Subset Product, it suffices to solve SimulSubsetSum with at most $\log t$ many instances, where each of the targets are also 'small', at most $O(\log\log t)$ bits.

Theorem 1.13 (Reducing Subset Product to SimulSubsetSum)**.** *There is a deterministic polynomial time reduction from* Subset Product *to* SimulSubsetSum.

▶ Remark. The reduction uses $\tilde{O}(n\log t)$ space as opposed to the following chain of reductions: Subset Product $\leq_{\mathsf{P}}$ SSUM $\leq_{\mathsf{P}}$ SimulSubsetSum. The first reduction is a *natural* reduction, from an input $(a_1,\ldots,a_n,t)$, which takes log both sides and adjust (multiply) a 'large' M (it could be $O(n\log t)$ bit [KP10; PST21]) with $\log a_i$, to reduce this to a SSUM instance with $b_i := \lfloor M\log a_i\rfloor$. Therefore, the total space required could be as large as $\tilde{O}(n^2\log t)$. The second reduction follows from Theorem 7.2. Therefore, ours is more space efficient. Motivated thus, we give an efficient randomised algorithm for SimulSubsetSum.

In the latter part, we present a few reductions among SSUM, $k-$ SSSUM and SimulSubsetSum problems. We also extend Problem 1.5-1.6 to the *unbounded version* of the Subset Sum problem (UBSSUM) and show similar theorems as above. For details, see Section 7.4-7.5.

Chapter 2

Preliminaries and Notations

2.1 Notations

In this thesis, $\mathbb{N}, \mathbb{Z}$, $\mathbb{Q}$ and $\mathbb{R}$ will denote the sets of all natural numbers, integers, rationals and reals, respectively. Let a, b be two m-bit integers.Then, $a//b$ denotes a/b^e where e is the largest non-negative integer such that $b^e \,|\, a$. Observe that $a//b$ is not divisible by b and the time to compute $a//b$ is $O(m\log(m) \cdot \log(e))$. Also, $\tilde{O}(N)$ denotes $N \cdot \mathsf{poly}(\log N)$.

For any positive integer $n > 0$, $[n]$ denotes the set $\{1, 2, \ldots, n\}$ while $[a, b]$ denotes the set of integers i such that $a \leq i \leq b$. Also, $2^{[n]}$ denotes the set of all subsets of $[n]$, while log denotes $\log_2$. A weight function $w : [n] \longrightarrow [m]$, can be naturally extended to a set $S \in 2^{[n]}$, by defining $w(S) := \sum\limits_{i \in S} w(i)$. We also denote $\tilde{O}(g)$ to be $g \cdot \mathsf{poly}(\log g)$.

Vectors will be denoted by small cases, and matrices and basis sets will be denoted in capital letters. Let $B = \{\vec{b_1}, \ldots, \vec{b_k}\}$ be a set of vectors in $\mathbb{R}^n$. The subspace of $\mathbb{R}^n$ spanned by B will be denoted by $\mathsf{span}(B)$. The norm of a vector $\vec{v} = [v_1, \ldots, v_n]$ is the normal Euclidean norm, i.e, $||\vec{v}|| = \sqrt{\sum_i v_i^2}$. The norm of B is defined as $||B|| = \max\limits_{i \in [n]} ||\vec{b_i}||$. For any two sets of vectors U and V, $U + V$ will denote the set $\{\vec{u} + \vec{v} \mid \vec{u} \in U,\ \vec{v} \in V\}$.

$\mathbb{F}[x_1, \ldots, x_k]$ denotes the ring of k-variate polynomials over field $\mathbb{F}$. $\mathbb{F}[[x_1, \ldots, x_k]]$ denotes the ring of power series in k-variables over $\mathbb{F}$. We will use the short-hand notation $\boldsymbol{x}$ to denote the collection of variables $(x_1, \ldots, x_k)$ for some k. For any non-negative integer vector $\overline{\boldsymbol{e}} \in \mathbb{Z}^k$, $\boldsymbol{x}^{\overline{\boldsymbol{e}}}$ (and $\pi_{\overline{\boldsymbol{e}}}(\boldsymbol{x})$) denotes $\prod_{i=1}^{k} x_i^{e_i}$. Using these notations, we can will write any polynomial $f(\boldsymbol{x}) \in \mathbb{Z}[\boldsymbol{x}]$ as $f(\boldsymbol{x}) = \sum_{\overline{\boldsymbol{e}} \in S} f_e \cdot \boldsymbol{x}^{\overline{\boldsymbol{e}}}$ for some suitable set S. We will use the notation $\pi_{\overline{\boldsymbol{e}}}(\boldsymbol{x}) \to f$ to denote that the coefficient of $\pi_{\overline{\boldsymbol{e}}}(\boldsymbol{x})$ in $f(\boldsymbol{x})$ is non-zero. If the coefficient is 0, it will be denoted using the notation $\pi_{\overline{\boldsymbol{e}}}(\boldsymbol{x}) \not\to f$.

We denote $\text{coef}_{\boldsymbol{x}^{\boldsymbol{e}}}(f)$, as the coefficient of $\boldsymbol{x}^{\boldsymbol{e}}$ in the polynomial $f(\boldsymbol{x})$ and $\deg_{x_i}(f)$ as the highest degree of x_i in $f(\boldsymbol{x})$. Sparsity of a polynomial $f(x_1, \ldots, x_k) \in \mathbb{F}[x_1, \ldots, x_k]$ over a field $\mathbb{F}$, denotes the number of nonzero terms in f.

2.2 Sets, Polynomials and Probability

Lemma 2.1 ([MVV87, Isolation Lemma]). *Let n and N be positive integers, and let $\mathcal{F}$ be an arbitrary family of subsets of $[n]$. Suppose $w(x)$ is an integer weight given to each element $x \in [n]$ uniformly and independently at random from $[N]$. The weight of $S \in \mathcal{F}$ is defined as $w(S) = \sum_{x \in S} w(x)$. Then, with probability at least $1 - n/N$, there is a unique set $S' \in \mathcal{F}$ that has the minimum weight among all sets of $\mathcal{F}$.*

Lemma 2.2 (Kane's Identity [Kan10]). *Let $f(x) = \sum_{i=0}^{d} c_i x^i$ be a polynomial of degree at most d with coefficients c_i being integers. Let $\mathbb{F}_q$ be the finite field of order $q = p^k > d + 2$. For $0 \leq t \leq d$, define*

$$r_t = \sum_{x \in \mathbb{F}_q^*} x^{q-1-t} f(x) = -c_t \in \mathbb{F}_q$$

Then, $r_t = 0 \iff c_t$ is divisible by p.

Lemma 2.3 (Newton's Identities). *Let $X_1, \ldots, X_n$ be $n \geq 1$ variables. Let $P_m(X_1, \ldots, X_n) = \sum_{i=1}^{n} X_i^m$, be the m-th power sum and $E_m(X_1, \ldots, X_n)$ be the m^{th} elementary symmetric polynomials, i.e., $E_m(x_1, \ldots, x_n) = \sum_{1 \leq j_1 \leq \ldots \leq j_m \leq n} X_{j_1} \cdots X_{j_m}$,* *then*

$$m \cdot E_m(X_1, \ldots, X_n) \;=\; \sum_{i=1}^{m} (-1)^{i-1} E_{m-i}(X_1, \ldots, X_n) \cdot P_i(X_1, \ldots, X_n)\ .$$

Remark 2.4. $E_m(X_1, \ldots, X_n) = 0$ *when* $m > n$*.*

Lemma 2.5 (Vieta's formulas)**.** *Let* $f(x) = \prod_{i=1}^{n}(x - a_i)$ *be a monic polynomial of degree* n*. Then,* $f(x) = \sum_{i=0}^{n} c_i x^i$ *where* $c_{n-i} = (-1)^i E_i(a_1, \ldots, a_n), \forall 1 \leq i \leq n$ *and* $c_n = 1$*.*

Lemma 2.6 (Polynomial division with remainder [VG13, Theorem 9.6])**.** *Given a* d*-degree polynomial* f *and a linear polynomial* g *over a finite field* $\mathbb{F}_p$*, there exists a deterministic algorithm that finds the quotient and remainder of* f *divided by* g *in* $\tilde{O}(d \log p)$*-time.*

Definition 2.7 (Order of a number mod p)**.** *The order of* a $\pmod{p}$*, denoted as* $ord_p(a)$ *is defined to be the* smallest *positive integer* m *such that* $a^m \equiv 1 \mod p$*.*

Theorem 2.8 ([Shp96])**.** *There exists a* $\tilde{O}(p^{1/4+\epsilon})$ *time algorithm to deterministically find a primitive root over* $\mathbb{F}_p$*.*

Theorem 2.9 ([Nag52])**.** *For* $n \geq 25$*, there is a prime in the interval* $[n, \frac{6}{5} \cdot n]$*.*

The following is a naive bound, but it is sufficient for our purpose.

Lemma 2.10. *For integers* $a \geq b \geq 1$*, we have* $(a/b)^b \leq 2^{2\sqrt{ab}}$*.*

Proof. Let $x = \sqrt{a/b}$. We need to show that $x^{2b} \leq 2^{2bx}$, which is trivially true since $x \leq 2^x$, for $x \geq 1$. $\square$

2.3 Attacks against Symmetric Cryptosystems

2.3.1 Cube Attacks

Here we provide a high-level overview of the cube attack model. To initialise its state, a stream cipher typically uses one secret key, **k**, and a set of public variables

$\boldsymbol{x}$. The secret key remains secret, and it is known only to the encryptor and the decryptor.

However, the $\boldsymbol{x}$ (or nonce) is considered a public variable. Keystream bits are typically generated by the ciphers after some initialisation rounds. By interpreting the keystream bit z as a Boolean polynomial over the secret key $\mathbf{k}$ and $\boldsymbol{x}$, it can be expressed as $z = f(\boldsymbol{x}, \mathbf{k})$. Let secret key variable be denoted by $\mathbf{k} = (k_1, \ldots, k_m)$ and public variable by $\boldsymbol{x} = (x_1, \ldots, x_n)$. The cube attack is based on the principle of simplifying the polynomial $z = f(\boldsymbol{x}, \mathbf{k})$ in order to obtain the value of $\mathbf{k}$. Let us set the cube indices $\mathbf{I} = \{i_1, \ldots, i_c\} \subseteq [n]$ in the preprocessing phase. Then we can express $f(\boldsymbol{x}, \mathbf{k})$ as

$$f(\boldsymbol{x}, \mathbf{k}) = \left(\prod_{i \in \mathbf{I}} x_i \cdot p_{\mathbf{I}}(\boldsymbol{x}, \mathbf{k})\right) + q_{\mathbf{I}}(\boldsymbol{x}, \mathbf{k}),$$

where each monomial of the function $q_{\mathbf{I}}$ misses at least one variable from the set $\{x_{i_1}, x_{i_2}, \cdots, x_{i_c}\}$.

In order to use the attack during the online phase, we only need to know its superpoly $p_{\mathbf{I}}$, and we can set the values of non-cube variables x_i to 0 (or to 1) to simplify the long superpoly expression. Consider the $C_{\mathbf{I}} = \{(x_{i_1}, \ldots, x_{i_c}) : x_j \in \{0, 1\} \text{ for } j \in \mathbf{I}\}$. The superpoly can then be recovered because $\sum_{(x_{i_1}, \ldots, x_{i_c}) \in C_{\mathbf{I}}} f(\boldsymbol{x}, \mathbf{k}) = p_{\mathbf{I}}(\boldsymbol{x}, \mathbf{k})$.

Thus, the attacker first finds some cube variables and then computes the sum of the output bits for all possible values of the cube variables to determine the value of the superpoly. The attacker's objective is to select cube variables and fix the remaining public variables in such a way that the superpoly is reduced to a linear function in secret variables.

The main point of concern is that after a large number of initialisation rounds, the expression of the output bit always becomes very much complicated.

In reality, after a few rounds of initialisation, it is not possible to compute the algebraic expression of the output bit in the latest stream ciphers because of the way they are designed. To tackle this problem, the attacker uses the cipher as a blackbox. He randomly chooses the cube variables, and the blackbox provides the output bits corresponding to all possible values of the cube variables. The value of the superpoly can be recovered by the attacker by performing the sum on the values that have been retrieved. The BLR linearity test [BLR90] can be used by the attacker to determine whether or not the superpoly is linear. A more in-depth description of the cube attack may be found in [DS09].

In 2009, Aumasson et al. [Aum+09] proposed cube tester to assess the non-randomness of a Boolean function. The presence of a monomial, balancedness, constantness, presence of linear variables, and the presence of neutral variables can be determined by using cube tester. A Boolean function is said to be vulnerable if it can be distinguished by some property such as the ones mentioned above.

Upper bound of degree

Given a Boolean polynomial f in n variable, we want to find its algebraic degree after fixing some variables. Let $x_1, \ldots, x_n$ be the variables of the function f.

Assume that $x_1, \ldots, x_k$ are initially fixed to 0. Now we are interested in checking whether the reduced polynomial (after setting $x_1, \ldots, x_k$ to 0) has degree $n-k$ or not. For this purpose, we do the following:

1. Consider $x_{k+1}, x_{k+2}, \ldots, x_n$ as cube variables.

2. Calculate the cube sum on the reduced polynomial over the prescribed cube variables.

3. If the cube sum is zero, then we conclude that the degree of the reduced polynomial is strictly less than $n-k$.

2.3.2 Monomial Trails

Definition 2.11 (Monomial trail)**.** *Let $\boldsymbol{x}^{(i+1)} = f^{(i)}(\boldsymbol{x}^{i})$ for $0 \leq i < r$. We say a monomial trail exists from $\pi_{\mathbf{u}^{(0)}}(\boldsymbol{x}^{(0)})$ to $\pi_{\mathbf{u}^{(r)}}(\boldsymbol{x}^{(r)}) = \pi_{\mathbf{u}^{(r)}}(f^{(r-1)}(\boldsymbol{x}^{(r-1)}))$ (denoted* *as $\pi_{\mathbf{u}^{(0)}}(\boldsymbol{x}^{(0)}) \rightsquigarrow \pi_{\mathbf{u}^{(r)}}(\boldsymbol{x}^{(r)})$) if there exists a sequence of monomials*

$$(\pi_{\mathbf{u}^{(0)}}(\boldsymbol{x}^{(0)}), \pi_{\mathbf{u}^{(1)}}(\boldsymbol{x}^{(1)}), \ldots, \pi_{\mathbf{u}^{(r)}}(\boldsymbol{x}^{(r)}))$$

such that $\pi_{\mathbf{u}^{(i)}}(\boldsymbol{x}^{(i)}) \rightarrow \pi_{\mathbf{u}^{(i+1)}}(\boldsymbol{x}^{(i+1)})$ for all $0 \le i < r$.

Example 2.12. *Let $\boldsymbol{y} = (y_0, y_1) = f^{(0)}(x_0, x_1) = (x_0 \oplus x_1, x_0x_1 \oplus x_1)$ and $\boldsymbol{z} = (z_0) = f^{(1)}(y_0, y_1) = (y_0 \oplus y_1 \oplus y_0y_1)$. Consider the monomial $\boldsymbol{x}^{(0,1)} = x_1$. We have*

$$\boldsymbol{y}^{(0,0)} = 1, \quad \boldsymbol{y}^{(0,1)} = y_1 = x_0x_1 \oplus x_1, \quad \boldsymbol{y}^{(1,0)} = y_0 = x_0 \oplus x_1$$

$$\boldsymbol{y}^{(1,1)} = y_0y_1 = (x_0 \oplus x_1) \cdot (x_0x_1 \oplus x_1) = x_0x_1 \oplus x_1$$

which implies $x_1 \rightarrow y_0, x_1 \rightarrow y_1, x_1 \rightarrow y_0y_1$. Also, we have

$$\boldsymbol{z}^{(1)} = z_0 = y_0 \oplus y_1 \oplus y_0y_1 = x_1 \oplus x_0$$

This implies that $y_0 \rightarrow z_0, y_1 \rightarrow z_0, y_0y_1 \rightarrow z_0$. Combining all the above, we get

$$x_1 \rightarrow y_0 \rightarrow z_0, \quad x_1 \rightarrow y_1 \rightarrow z_0, \quad x_1 \rightarrow y_0y_1 \rightarrow z_0$$

Hence, we have $x_1 \rightsquigarrow z_0$ and there are three different monomial trails from x_1 to z_0.

As seen in Example 2.12, there can be multiple monomial trails from $\pi_{\mathbf{u}^{(0)}}(\boldsymbol{x}^{(0)})$ to $\pi_{\mathbf{u}^{(r)}}(\boldsymbol{x}^{(r)})$. Observe that the existence of a monomial trail from $\pi_{\mathbf{u}^{(0)}}(\boldsymbol{x}^{(0)})$ to $\pi_{\mathbf{u}^{(r)}}(\boldsymbol{x}^{(r)})$ does not necessarily imply that $\pi_{\mathbf{u}^{(0)}}(\boldsymbol{x}^{(0)})$ is a monomial in $\pi_{\mathbf{u}^{(r)}}(\boldsymbol{x}^{(r)})$. Considering Example 2.12, we have $x_0x_1 \rightarrow y_0 \rightarrow z_0$, i.e., $x_0x_1 \rightsquigarrow z_0$ but $x_0x_1 \not\rightarrow z_0$.

We can only guarantee the converse, i.e.,

Claim 2.13. *If $\pi_{\mathbf{u}^{(0)}}(\boldsymbol{x}^{(0)}) \not\rightsquigarrow \pi_{\mathbf{u}^{(r)}}(\boldsymbol{x}^{(r)})$ then $\pi_{\mathbf{u}^{(0)}}(\boldsymbol{x}^{(0)}) \not\rightarrow \pi_{\mathbf{u}^{(r)}}(\boldsymbol{x}^{(r)})$.*

Let us denote $|\pi_{\mathbf{u}^{(0)}}(\boldsymbol{x}^{(0)}) \rightsquigarrow \pi_{\mathbf{u}^{(r)}}(\boldsymbol{x}^{(r)})|$ the number of distinct monomial trails from $\pi_{\mathbf{u}^{(0)}}(\boldsymbol{x}^{(0)})$ to $\pi_{\mathbf{u}^{(r)}}(\boldsymbol{x}^{(r)})$. Then, the monomial $\pi_{\mathbf{u}^{(0)}}(\boldsymbol{x}^{(0)})$ exists in $\pi_{\mathbf{u}^{(r)}}(\boldsymbol{x}^{(r)})$ if the number is odd.

Claim 2.14. *If* $|\pi_{\mathbf{u}^{(0)}}(\boldsymbol{x}^{(0)}) \rightsquigarrow \pi_{\mathbf{u}^{(r)}}(\boldsymbol{x}^{(r)})|$ *is odd, then* $\pi_{\mathbf{u}^{(0)}}(\boldsymbol{x}^{(0)}) \rightarrow \pi_{\mathbf{u}^{(r)}}(\boldsymbol{x}^{(r)})$.

We can see that in Example 2.12, $|x_1 \rightsquigarrow z_0| = 3$ which is odd, therefore, $x_1 \rightarrow z_0$. But, $|x_0x_1 \rightsquigarrow z_0| = 2$, hence $x_0x_1 \not\rightarrow z_0$.

Therefore, Claim 2.14 gives us a procedure to find whether a monomial exists in a polynomial or not. For more details on monomial trails, we refer to [Hu+20].

Chapter 3

On the Hardness of Monomial Prediction

3.1 Preliminaries

Definition 3.1 (The class $\oplus\mathsf{P}$)**.** *In computational complexity theory, the complexity class $\oplus\mathsf{P}$ (pronounced 'parity P') is the class of decision problems solvable by a nondeterministic Turing machine in polynomial time, where the acceptance condition is that the number of accepting computation paths is* odd*.*

Definition 3.2 ($\oplus\mathsf{P}$-complete)**.** *A problem L is said to be $\oplus\mathsf{P}$-hard if every problem in $\oplus\mathsf{P}$ can be reduced to L in polynomial time. It is said to be $\oplus\mathsf{P}$-complete if it is in $\oplus\mathsf{P}$ and also $\oplus\mathsf{P}$-hard.*

There are interesting problems known to be $\oplus\mathsf{P}$-complete [Val05]. We will also use one of them in our hardness result; for details see Section 3.2.

How hard are $\oplus\mathsf{P}$-complete problems? It is not hard to show that $\oplus\mathsf{P}$ contains the graph isomorphism problem. On the other hand, $\mathsf{P}^{\oplus\mathsf{P}}$ (oracle power to a machine computing $\oplus\mathsf{P}$ function) is *not known* to contain NP. However, it is *not known* (or believed) to be as strong as $\#\mathsf{P}$. This distinction is important in our context since monomial prediction in the general setting over $\mathbb{Z}$ is known to be

#P-complete [Kay10], while the scenario changes when one works over the field $\mathbb{F}_2$.

3.2 Hardness Result

We recall the definition of the language and the hardness theorem mentioned in Section 1.1.1.

$$\begin{aligned} L \;:=\; \{(f,m) \mid \mathrm{coef}_m(f_1) \;=\; 1\,,\ \text{where } (f_1,\ldots,f_{n_{r+1}}) \;=\; g_r \circ g_{r-1} \circ \ldots g_0\,, \\ \text{and } g_i : \mathbb{F}_2^{n_i} \longrightarrow \mathbb{F}_2^{n_{i+1}}\,, n_i \in \mathbb{N}\ \forall\ i \;\in\; [r+1], \text{with } n_0 = n, \\ \text{monomial } m \;\in\; \mathbb{F}_2[x_1,\ldots,x_n], \text{and } \deg((g_i)_j) \le 2\,\}\,. \end{aligned}$$

Theorem 3.3 (⊕P-completeness)**.** *Given a composition of quadratic functions f and a monomial m, deciding whether $(f,m) \in L$ is ⊕P-complete.*

Proof. The proof is motivated from algebraic complexity theory and uses the *Hamiltonian Cycle polynomial*, HC_n, defined below, which is a well-known VNP-complete[1] polynomial over $\mathbb{F}_2$ [Bür00; Mal03]. Remarkably the motivation for studying the hardness of HC_n is quite different from ours and concerns arithmetic circuit complexity, while in this paper, we are interested in *Boolean hardness* results!

Recall the definition of *Hamiltonian cycle*: it is a closed loop on a graph where every node (vertex) is visited *exactly* once. It is known that the problem `Odd Hamiltonian Cycle` – deciding whether a given graph $G = (V,E)$ has an odd number of Hamiltonian cycles, is ⊕P-complete [Val05], i.e., it is in ⊕P and also is ⊕P-hard. We will use `Odd Hamiltonian Cycle` to show the completeness of L. In particular, we will show the proof in two parts –

1. Part A - a reduction from `Odd Hamiltonian cycle` $\le_{\mathsf{P}}$ L this implying that our problem is ⊕P-hard, and
2. Part B - L is in ⊕P.

[1]The class VNP, Valiant's NP, is known as the *algebraic* NP class in the algebraic complexity theory.

Part A: Proof of $\oplus\mathsf{P}$-hardness

Define the Hamiltonian Cycle polynomial (HC_n) for a graph with n nodes, with the adjacency matrix $(x_{i,j})_{1\le i,j\le n}$ (they are just elements from $\{0,1\}$), as follows:

$$\mathsf{HC}_n\left(x_{1,1},\ldots,x_{n,n}\right) \;=\; \sum_{\sigma\in S_n}\prod_{i=1}^{n} x_{i,\sigma(i)}\ ,$$

where S_n is the symmetric group on a set of size n and the sum is taken over all n-cycles of S_n (i.e. , every monomial in HC_n corresponds to a Hamiltonian cycle in the complete directed graph on n vertices). Here is the crucial lemma.

Lemma 3.4 (Composition lemma)**.** *Let $G=(V,E)$ be a given graph with the adjacency matrix $\boldsymbol{x}=(x_{i,j})_{i,j\in[n]}$. Let $\boldsymbol{y}=(y_1,\ldots,y_n)$ and $\boldsymbol{z}=(z_1,\ldots,z_n)$ be $2n$ variables. Then, there exist $g_0,\ldots,g_n$, polynomial maps such that*

(i) $g_0:\mathbb{F}_2^{n^2+2n}\longrightarrow\mathbb{F}_2^{2n^2}$, *and* $g_i:\mathbb{F}_2^{2n^2}\longrightarrow\mathbb{F}_2^{2n^2}$, *for* $i\in[n]$, *with* $\deg((g_i)_j)\le 2$, *and*

(ii) $\mathrm{coef}_{y_1\cdots y_n\cdot z_1\cdots z_n}(f_1(\boldsymbol{x},\boldsymbol{y},\boldsymbol{z}))=\mathsf{HC}_n(\boldsymbol{x})$, *where* $(f_1,\ldots,f_{2n^2})=g_n\circ\ldots\circ g_0$.

The above lemma directly implies that for a given graph $G=(V,E)$ with adjacency matrix $(x_{i,j})_{i,j}$, $(f:=g_n\circ\ldots\circ g_0, m:=y_1\cdots y_nz_1\cdots z_n)\in L \iff G$ has an odd number of Hamiltonian cycles, which would finish the proof.

Proof of Lemma 3.4. In the proof, we will often interchange k-th coordinate with (i,j)-th position, for $k\in[n^2]$, where $k-1=(i-1)+n(j-1)$, and $i,j\in[n]$. Since, $k-1\in[0,n^2-1]$ can be *uniquely* written as $(i-1)+n(j-1)$, for some $i,j\in[n]$, there is a *one-to-one* correspondence. We divide the proof into two:

Part 1 : Construction of g_i's. Define the polynomial map $g_0:\mathbb{F}_2^{n^2+2n}\longrightarrow\mathbb{F}_2^{2n^2}$, by defining each coordinate of g_0, namely $(g_0(\boldsymbol{x},\boldsymbol{y},\boldsymbol{z}))_k$, for $k\in[2n^2]$ by:

$$(g_0(\boldsymbol{x},\boldsymbol{y},\boldsymbol{z}))_k := \begin{cases} x_{i,j}, & \text{when } k\le n^2, \text{where } k-1=(i-1)+n(j-1),\\ y_i\cdot z_j, & \text{when } n^2<k\le 2n^2, \text{where } k-1-n^2=(i-1)+n(j-1).\end{cases}$$

In the above, we used the fact that $k-1-n^2\in[0,n^2-1]$ and hence the one-to-one correspondence exists. Trivially any coordinate $(g_0(\boldsymbol{x},\boldsymbol{y},\boldsymbol{z}))_k$ is at most a quadratic polynomial.

Now define $g_1 : \mathbb{F}_2^{2n^2} \longrightarrow \mathbb{F}_2^{2n^2}$, on $2n^2$ variables $\boldsymbol{w} := (w_{i,j})_{i,j\in[n]}$ and $\mathbf{s} := (s_{i,j})_{i,j\in[n]}$, as follows:

$$(g_1(\boldsymbol{w},\mathbf{s}))_k := \begin{cases} w_{i,j}\cdot s_{i,j}, & \text{when } k \le n^2, \text{where } k-1=(i-1)+n(j-1), \\ (g_1(\boldsymbol{w},\mathbf{s}))_{k-n^2}\,, & \text{when } n^2 < k \le 2n^2. \end{cases}$$

Basically, g_1 repeats the first n^2 coordinates. Again, by definition, each ordinate is a quadratic polynomial. Now, we can define $g_\ell : \mathbb{F}_2^{2n^2} \longrightarrow \mathbb{F}_2^{2n^2}$, again on $2n^2$ variables $(\boldsymbol{w},\mathbf{s})$, for $\ell > 1$, as follows:

$$(g_\ell(\boldsymbol{w},\mathbf{s}))_k := \begin{cases} \sum_{r=1}^{n} w_{i,r}\cdot s_{r,j}, & \text{when } k \le n^2, \text{where } k-1=(i-1)+n(j-1), \\ s_{i,j}, & \text{when } n^2 < k \le 2n^2, \text{where } k-1-n^2=(i-1)+n(j-1). \end{cases}$$

It is easy to see that, by definition, g_ℓ, restricted to the last n^2 coordinates, is an *identity* map. Also, trivially, each coordinate is a quadratic polynomial.

Part 2 : Getting HC_n as a coefficient of $g_n\circ\ldots\circ g_0$. We will prove two claims about the structure of the compositions. Here is the first claim.

Claim 3.5. *For any $\ell \ge 1$, we have $(g_\ell(\ldots(g_0(\boldsymbol{x},\boldsymbol{y},\boldsymbol{z})\ldots)_k = x_{i,j}\cdot y_i\cdot z_j$, for $k\in[n^2+1,2n^2]$, where $k-1-n^2=(i-1)+n(j-1)$.*

Proof. First, let us prove this for $\ell = 1$. Since, there is a one-to-one correspondence between the k-th coordinate and the pair (i,j), by definition,

$$g_1(g_0(\boldsymbol{x},\boldsymbol{y},\boldsymbol{z}))_k \;=\; g_1(g_0(\boldsymbol{x},\boldsymbol{y},\boldsymbol{z}))_{k-n^2} \;=\; x_{i,j}\cdot y_i\cdot z_j\;.$$

Since g_ℓ is an identity map in the last n^2 coordinates, for $\ell > 1$, the conclusion follows immediately. □

We remark that, in fact, in the above, it can be easily seen that $g_1(g_0(\boldsymbol{x},\boldsymbol{y},\boldsymbol{z}))_k =$

$x_{i,j} \cdot y_i \cdot z_j$, for $k \in [n^2]$, where $k-1 = (i-1) + n(j-1)$. However, since ℓ grows, $g_\ell \circ \ldots g_0$ looks *complicated.* Here is the main claim about the structure of the composition for the first n^2 coordinates.

Claim 3.6 (Main claim)**.** *For any $\ell \geq 2$, and $k \in [n^2]$, such that $(k-1) = (i-1) + n(j-1)$, the following holds:*

$$\begin{aligned}
&(g_\ell(\ldots(g_0(\boldsymbol{x},\boldsymbol{y},\boldsymbol{z})\ldots))_k \\
&= y_i z_j \cdot \sum_{1\leq m_1,\ldots,m_{\ell-1}\leq n} x_{i,m_1} x_{m_1,m_2} \cdots x_{m_{\ell-2},m_{\ell-1}} x_{m_{\ell-1},j} \cdot \left(\prod_{s=1}^{\ell-1} y_{m_s} z_{m_s}\right) .
\end{aligned}$$

Proof of the Claim. We will prove this by induction on ℓ.

Base case: $\ell = 2$**.** For $\ell = 2$, by definition, we have

$$\begin{aligned}
(g_2(g_1(g_0(\boldsymbol{x},\boldsymbol{y},\boldsymbol{z}))_k &= \sum_{r=1}^{n} (x_{i,r} y_i z_r) \cdot (x_{r,j} y_r z_j) \\
&= y_i z_j \cdot \sum_{1\leq r\leq n} x_{i,r} x_{r,j} y_r z_r \ ,
\end{aligned}$$

as desired. In the above, we implicitly used the (i,r)-th coordinate, by which we mean the k'-th coordinate such that $k'-1 = (i-1) + n(r-1)$, the similar correspondence as we mentioned at the beginning of the proof of Lemma 3.4. Thus, the base case is true.

Inductive step: $(\ell+1)$**-th step.** Let us assume that it is true for some ℓ. To show this for $\ell+1$, again, by definition (and the one-to-one correspondence between k

and (i, j)), we have

$$
\begin{aligned}
&(g_{\ell+1}(\dots(g_0(\boldsymbol{x},\boldsymbol{y},\boldsymbol{z})\dots)_k \\
&= \sum_{r=1}^{n} (g_\ell(\dots(g_0(\boldsymbol{x},\boldsymbol{y},\boldsymbol{z})\dots)_{i,r} \cdot (x_{r,j}y_rz_j) \\
&= \sum_{1\le r\le n} \left(\sum_{1\le m_1,\dots,m_{\ell-1}\le n} x_{i,m_1}x_{m_1,m_2}\cdots x_{m_{\ell-2},m_{\ell-1}}x_{m_{\ell-1},r} \cdot \left(\prod_{s=1}^{\ell-1} y_{m_s}z_{m_s}\right) \cdot y_iz_r \right) \cdot (x_{r,j}y_rz_j) \\
&= y_iz_j \cdot \sum_{1\le m_1,\dots,m_{\ell-1},m_\ell\le n} x_{i,m_1}x_{m_1,m_2}\cdots x_{m_{\ell-1},m_\ell}x_{m_\ell,j} \cdot \left(\prod_{s=1}^{\ell} y_{m_s}z_{m_s}\right) .
\end{aligned}
$$

The second last equality is by induction hypothesis, while in the last equality, we renamed r by m_ℓ. In the above, by (i,r)-th coordinate, again, we mean k'-th coordinate such that $k'-1 = (i-1)+n(r-1)$. This finishes the induction and the conclusion as well. □

Claim 3.6 with $k = 1$ (i.e. $i = j = 1$) and $\ell = n$, gives the following identity:

$$
\begin{aligned}
&(g_n(\dots(g_0(\boldsymbol{x},\boldsymbol{y},\boldsymbol{z})\dots)_1 \\
&= y_1z_1 \cdot \sum_{1\le m_1,\dots,m_{n-1}\le n} x_{1,m_1}x_{m_1,m_2}\cdots x_{m_{n-2},m_{n-1}}x_{m_{n-1},1} \cdot \left(\prod_{s=1}^{n-1} y_{m_s}z_{m_s}\right) .
\end{aligned}
$$

The coefficient of the monomial $y_1z_1\cdots y_nz_n$ is the Hamiltonian Cycle polynomial $\mathsf{HC}_n(\boldsymbol{x})$, because for any Hamiltonian cycle of length n, we must choose $m_1,\dots,m_{n-1}$, each between 2 and n, so that such a choice generates the monomial $x_{1,m_1}x_{m_1,m_2}\cdots x_{m_{n-2},m_{n-1}}x_{m_{n-1},1}$. Since, it visits each node *exactly once*, $y_1\cdots y_nz_1\cdots z_n$ is also generated with the $\boldsymbol{x}$-monomial. This finishes the proof of part 2. □

Since, Lemma 3.4 is now proved, the $\oplus$P-hardness follows, as well. □

Part B: Proof of $L \in \oplus\mathsf{P}$

We now show that deciding whether $(f, m) \in L$ is $\oplus\mathsf{P}$. To do this, we need to construct an NP machine $\mathcal{M}$ such that the number of accepting paths is *odd* due to Definition 3.1. The input to the machine $\mathcal{M}$ is (f, m), where $m = \prod_{i\in\mathbf{I}} x_i, \mathbf{I} \subseteq [n]$. The output of $\mathcal{M}$ is the evaluation of f_1 by setting $x_i = 0, \forall i \in [n] \setminus \mathbf{I}$, whereas non-deterministically picking $r_j \in \{0, 1\}$ and setting $x_j = r_j, \forall j \in \mathbf{I}$.

Let $|\mathbf{I}| = k$. As mentioned in the introduction, we can express $f_1(x)$ as

$$f_1(x) = m \cdot p_{\mathbf{I}}(x) + q_{\mathbf{I}}(x)$$

and we have

$$\sum_{\substack{(x_{i_1},\ldots,x_{i_k}) \in \mathbb{F}_2^k \\ x_j = 0, \forall j \notin \mathbf{I}}} f_1(x) = p_{\mathbf{I}}(x) \in \{0, 1\}$$

where $p_{\mathbf{I}}(x)$ does not contain any variable x_i where $i \in \mathbf{I}$ and none of the monomials in $q_{\mathbf{I}}(x)$ is divisible by m. Therefore, the above sum evaluates to 1 *iff* m is a monomial in f_1. It is easy to see that each accepting path of $\mathcal{M}$ is essentially a term in the above summation being evaluated to 1. Hence, m is a monomial in f_1 *iff* the number of accepting paths in $\mathcal{M}$ is *odd*. This finishes the proof of Part B. $\square$

Chapter 4

Preimage Attacks on Round Reduced KECCAK using Non-Linear Structures

In this chapter, we present the preimage attacks for round reduced KECCAK. In [GLS16], the authors try to set up linear equations between message bits (variables) and hash bits by controlling the diffusion due to θ and χ from producing any non-linear terms. Observation 4.1 is used to manage the diffusion due to θ. Lanes are fixed to constant to prevent χ from creating any non-linear terms. Furthermore, for KECCAK-384/512, the first row of the hash digest can be inverted due to Observation 4.2.

In most cases, the number of linear equations between the variables and hash values is strictly less than the hash length. Therefore, they repeat the whole procedure enough times by appropriately changing the constants in the system of linear equations. This gives a successful preimage attack. In [Li+17] [LS19], similar techniques are used to restrict χ from producing many non-linear terms. In our attacks, we allow χ to produce non-linear terms, but at the same time, we control the number of non-linear terms in the state.

4.1 Structure of KECCAK

KECCAK hash function is based on sponge construction [Ber+11b] which uses a padding function *pad*, a bitrate parameter r and a permutation function f as shown in Figure 4.1.

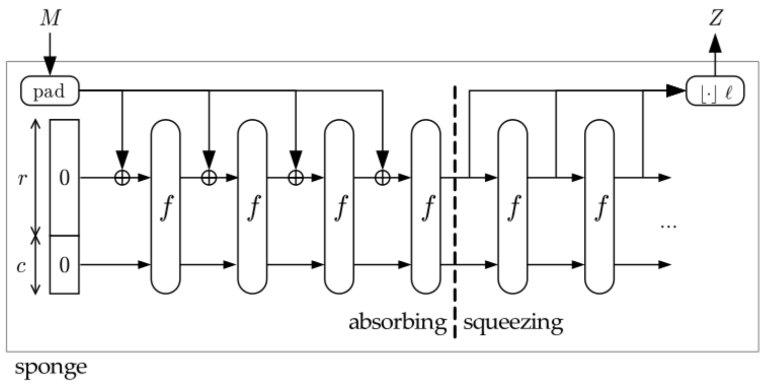

Figure 4.1: Sponge function [Ber+11b]

4.1.1 Sponge Construction

As shown in Figure 4.1, the sponge construction consists of two phases - absorbing and squeezing. It first applies the padding function *pad* on the input string M, which produces M' whose length is a multiple of r. In the absorbing phase, M' is split into blocks of r bits namely $m_1, m_2, ...m_k$. The initial state (IV) is a b bit string containing all 0. Here $b = r + c$ where c is called the capacity. The first r bits of IV is XOR-ed with the first block m_1 and is given as input to f. The output is XOR-ed with the next message block m_2 and then is given as input to f again. This process is continued till all the message blocks have been absorbed.

The squeezing phase extracts the required output, which can be of any length. Let ℓ be the required output length. If $\ell \leq r$, then the first ℓ bits of the output of the absorbing phase is the output of the sponge construction. Whereas, if $\ell > r$, then more blocks of r bits are extracted by repeatedly applying f on the output of the absorbing phase. This process is repeated enough times until we have extracted at least ℓ bits. The final output of the sponge construction is the first ℓ bits that have been extracted.

In the KECCAK hash family, the permutation function f is a KECCAK-$f[b]$ permutation, and the *pad* function appends 10*1 to input M. KECCAK-f is a specialization of KECCAK-p permutation.

$$\text{KECCAK-}f[b] = \text{KECCAK-p}[b, 12 + 2\gamma]$$

where $\gamma = \log_2(b/25)$.

The official version of KECCAK have $r = 1600 - c$ and $c = 2\ell$ where $\ell \in \{224, 256, 384, 512\}$ called KECCAK-224, KECCAK-256, KECCAK-384 and KECCAK-512.

4.1.2 KECCAK-p Permutation

KECCAK-p permutation is denoted by KECCAK-$p[b, n_r]$, where $b \in \{25, 50, 100, 200,$ $400,\ 800, 1600\}$ is the length of the input string and n_r is the number of rounds of the internal transformation. The parameter b is also called the width of the permutation. The b bit input string can be represented as a $5 \times 5 \times w$ 3-dimensional array known as state as shown in Figure 4.2. A lane in a state S is denoted by $S[x, y]$ which is the substring $S[x, y, 0] \mid S[x, y, 1] \mid \ \ldots \ \mid S[x, y, w - 1]$ where w is equal to $b/25$ and "|" is the concatenation function.

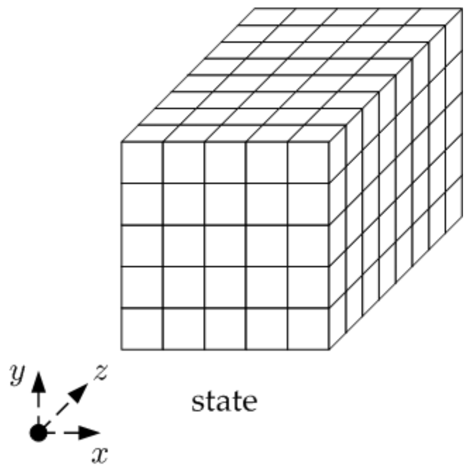

Figure 4.2: KECCAK state [Tea]

In each round, the state S goes through 5 step mappings θ,ρ,π,χ and ι, i.e. $Round(S, i_r) = \iota(\chi(\pi(\rho(\theta(S)))), i_r)$ where i_r is the round index. Except for χ, the rest of the step mappings are linear. In the following, S' is the state after applying the corresponding step mapping to S, "$\oplus$" denotes bitwise XOR and "$\cdot$" denotes bitwise AND.

1. θ: The θ step XOR's $S[x, y, z]$ with parities of its neighbouring columns in the following manner.

$$S'[x, y, z] = S[x, y, z] \oplus P[(x + 1) \bmod 5][(z - 1) \bmod 64] \\ \oplus P[(x - 1) \bmod 5][z]$$

where $P[x][z]$ is the parity of a column, i.e.,

$$P[x][z] = \bigoplus_{i=0}^{4} S[x, i, z]$$

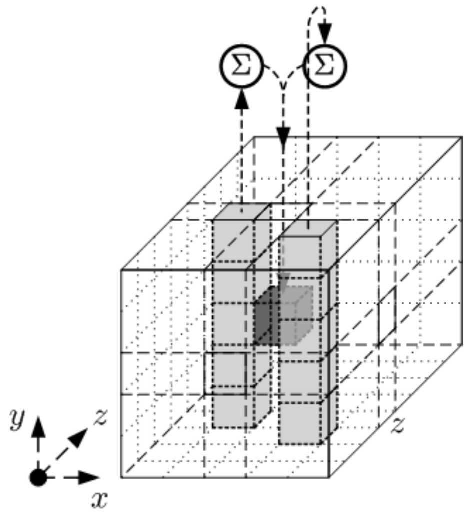

Figure 4.3: step θ [Tea]

2. ρ: The ρ step simply rotates each lane by a predefined value given in the table below, i.e.

$$S'[x, y] = S[x, y] \lll r[x][y]$$

where $\lll$ means bitwise rotation towards MSB of the 64-bit word.

4	18	2	61	56	14
3	41	45	15	21	8
2	3	10	43	25	39
1	36	44	6	55	20
0	0	1	62	28	27
$y\backslash x$	0	1	2	3	4

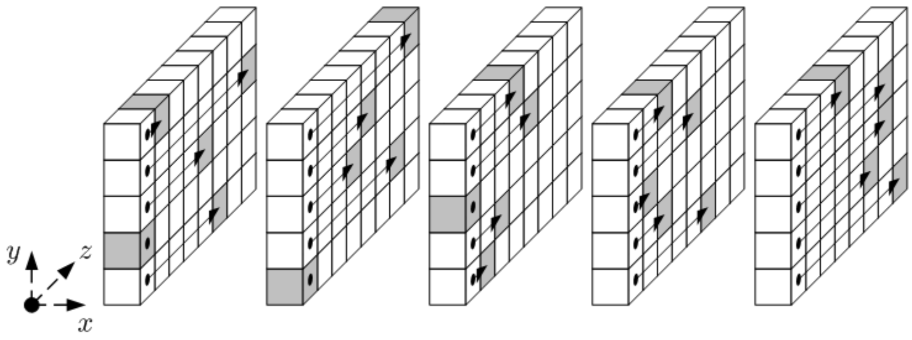

Figure 4.4: step ρ [Tea]

3. π: The π step interchanges the lanes of the state S.

$$S'[y, 2x + 3y] = S[x, y]$$

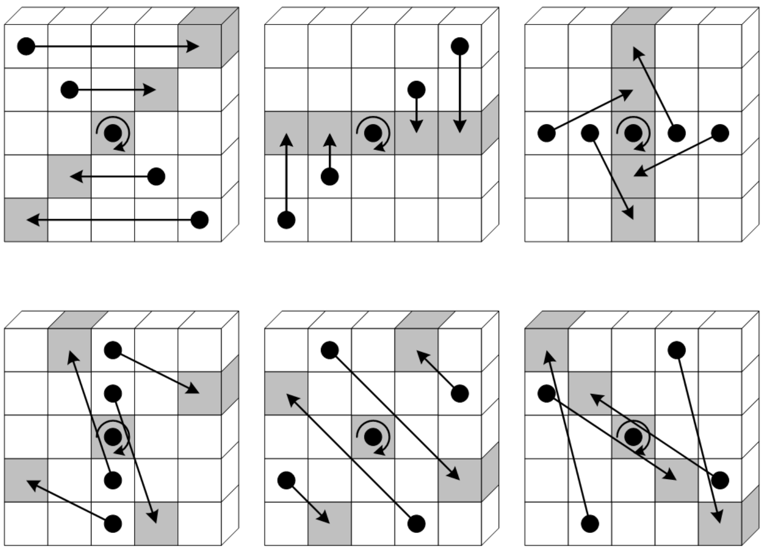

Figure 4.5: step π [Tea]

4. χ: The χ step is the only non-linear operation among the 5 step mappings

due to the quadratic term.

$$S'[x,y,z] = S[x,y,z] \oplus ((S[(x+1) \bmod 5, y, z] \oplus 1) \cdot \\ S[(x+2) \bmod 5, y, z])$$

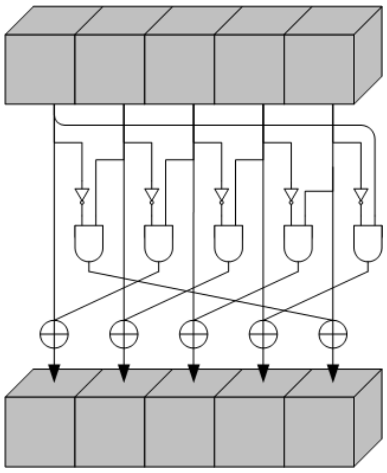
Figure 4.6: step χ [Tea]

5. ι: The ι step is the only step that depends on the round number.

$$S'[0,0] = S[0,0] \oplus RC_i$$

where RC_i is a constant which depends on i where i is the round number.

4.1.3 Observations

In this paper, we will be using the following observations made by Guo et al. [GLS16]. The χ step mapping is a row-dependent operation. Let a_0, a_1, a_2, a_3, a_4 be the 5 input bits to the χ operation and b_0, b_1, b_2, b_3, b_4 be the 5 output bits.

Observation 4.1. *Let d_0, d_1, d_2, d_3, d_4 be the elements of a column. Then, the parity of the column can be fixed to a constant c by choosing for any $i \in \{0,1,2,3,4\}$*

$$d_i = c \oplus \left(\bigoplus_{j=1}^{j=4} d_{i+j} \right)$$

Observation 4.2. *If the output of* χ *for an entire row is known, i.e.* $\chi([a_0, a_1, a_2, a_3, a_4]) = [b_0, b_1, b_2, b_3, b_4]$*, then we have*

$$a_i = b_i \oplus (b_{i+1} \oplus 1) \cdot (b_{i+2} \oplus (b_{i+3} \oplus 1) \cdot b_{i+4})$$

Observation 4.3. *If we are given two consecutive bits* b_i, b_{i+1} *of the output of* χ*, we can set up the following linear equation on the input bits.*

$$b_i = a_i \oplus (b_{i+1} \oplus 1) \cdot a_{i+2}$$

In the rest of the paper, all the message variables and hash values are represented in the form of lanes (array) of length 64, and we will use + symbol in place of $\oplus$. For a state A, $A[x, y]$ denotes a lane where $0 \leq x, y \leq 4$. In all the equations, the value inside the brackets '()' indicates the offset by which the lane is shifted. For example, $A[x, y](k)$ denotes lane $A[x, y]$ rotated by an offset of k. Every operation between two lanes is bitwise.

4.2 Preimage Attacks on 2 Rounds Keccak-512

In this subsection, we describe our preimage attack for 2-rounds Keccak-512. The best-known attack for this variant of Keccak is by Guo et al.[GLS16] with a complexity of 2^{384}. Their preimage attack is based on a linear structure by keeping four lanes as variables. We give two preimage attacks using six lanes as variables. In the first preimage attack, we keep the lanes in columns 1, 3, 4 as variables and get an attack of complexity 2^{337}, which can be improved to 2^{321}. However, the second preimage attack chooses a different set of lanes as variables and also has a complexity of 2^{321}.

4.2.1 Preimage Attack with Complexity 2^{337}

In Figure 4.7, we set the lanes in columns 1, 3 and 4 as variables, and the rest of the lanes are set to some constant. Therefore, we have $6 \times 64 = 384$ variables. To avoid the propagation by θ in the first round, we use Observation 4.1, i.e., $\bigoplus_{j=0}^{4} A[i,j] = \alpha_i, \forall i \in [0,2,3]$ where α_i is some constant and hence include $3 \times 64 = 192$ linear constraints to the system. Also, since the hash length is 512, we can invert the first row of the hash value due to Observation 4.2.

Observe that after the application of the χ operation in the first round, state (4) contains a lane with quadratic terms. Due to the θ of the second round, these will get propagated only to the neighbouring columns. Hence, the majority of the lanes in the state (5) contain only linear terms. However, while equating state (6) and state (7), we are only able to obtain $2 \times 64 = 128$ linear equations between the hash values and the variables. Observe that we have set up only 320 linear equations but have 384 variables.

Applying the techniques used in [GLS16], we can linearize the quadratic term and use them to create more linear equations between the hash value and the variables. Notice that in state (5), there is at most one quadratic term in each polynomial. This is because the state before the application of θ in the second round has only one lane containing polynomials with only one quadratic term. More precisely, $A[4,4]$ of state (4) contains a polynomial of the form $p_1 + \overline{p_2}.p_3$ where p_i's are linear polynomials. This non-linear polynomial can be linearized by adding one more linear equation to the system, say $p_3 = \beta$ where β is a constant. Therefore, if we linearize one quadratic term in state (4), we will be able to linearize 11 quadratic terms in state (5). But, only 3 out of the 11 linearized terms can be equated to the values in the state (7). Therefore, we can set up an additional 64 linear equations of which $3\lfloor 64/4 \rfloor = 48$ equations are between message bits and hash values. But, we need to include one more linear equation for the last message bit to be 1 to satisfy the padding condition of Keccak. Therefore, we have a system of linear equations in 384 variables and 384 equations. Since we have $128 + 48 - 1 = 175$ linear equations

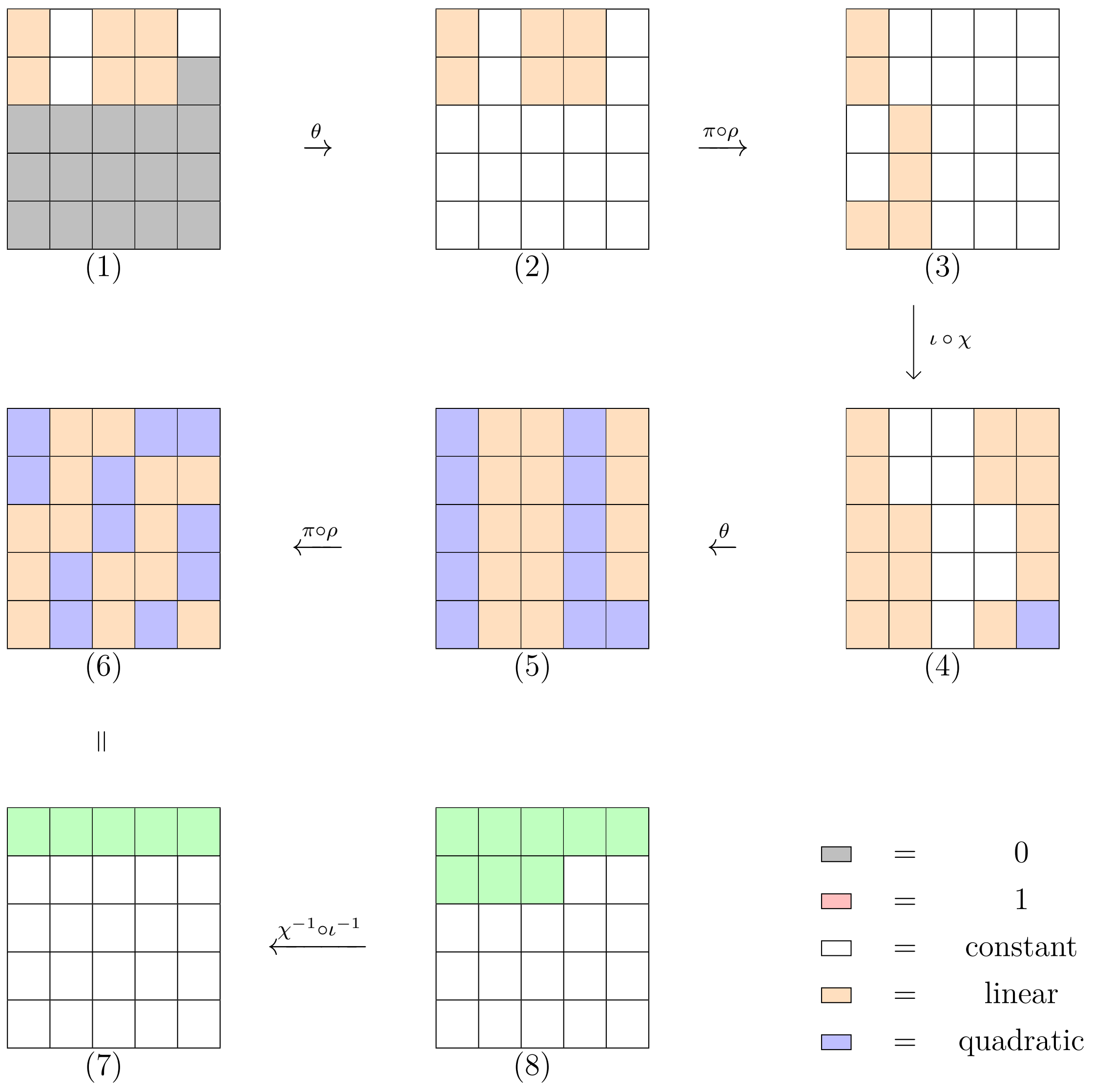

Figure 4.7: Preimage attack on 2-round Keccak-512

between hash values and variables, we get a valid preimage with probability $1/2^{337}$.

To get a successful preimage attack, we must repeat the above procedure for at least 2^{337} times where the system of linear equations is different each time. Observe that there are enough degrees of freedom to perform this, i.e. 192 bits from $A[1,0], A[1,1]$ and $A[4,0]$ and 192 bits from α_i for $i \in [0,2,3]$ which sums up to 384 bits. Therefore, we have a preimage attack for 2-rounds Keccak-512 with a complexity of 2^{337}.

4.2.2 Improved Analysis

In the previous analysis, by equating state (6) and (7), we were able to obtain 128 linear equations between the hash values and variables. Let us now focus on the second χ operation on the second row of state (6). Observe that the second and fourth lanes of the second row in state (6) are linear, whereas we know the values of the first three consecutive lanes of the output of the second χ operation. Using Observation 4.3, we can set up an additional 64 linear equations, which sums up to $128 + 64 - 1$ linear equations between the hash value and variables. Therefore, we have a primage attack for 2-rounds Keccak-512 with a complexity of 2^{321}.

By choosing a different set of lanes as variables, we have another preimage attack with complexity 2^{321}. In Figure 4.8, columns 1,2 and 4 are set as variables and the rest are set to constant. We also set $\bigoplus_{j=0}^{4} A[i,j] = \alpha_i, \forall i \in [0,1,3]$ where α_i is some constant, thus adding 192 linear equations to the system. Observe that in this case, we can set up $3 \times 64 - 1$ linear equations between the hash values and the variables. We must also include one more linear constraint for the last bit of the message to be 1 to satisfy the padding condition for Keccak. Therefore, we have a system of linear equations in 384 variables and 384 equations.

Since we are able to set up only 191 linear equations between the hash values and the variables, we get a valid preimage with probability $1/2^{321}$. Observe that there are enough degrees of freedom to repeat this procedure for 2^{321} due to 192 bits from $A[2,0], A[2,1]$ and $A[4,0]$ and 192 bits from α_i for $i \in [0,1,3]$ which sums up

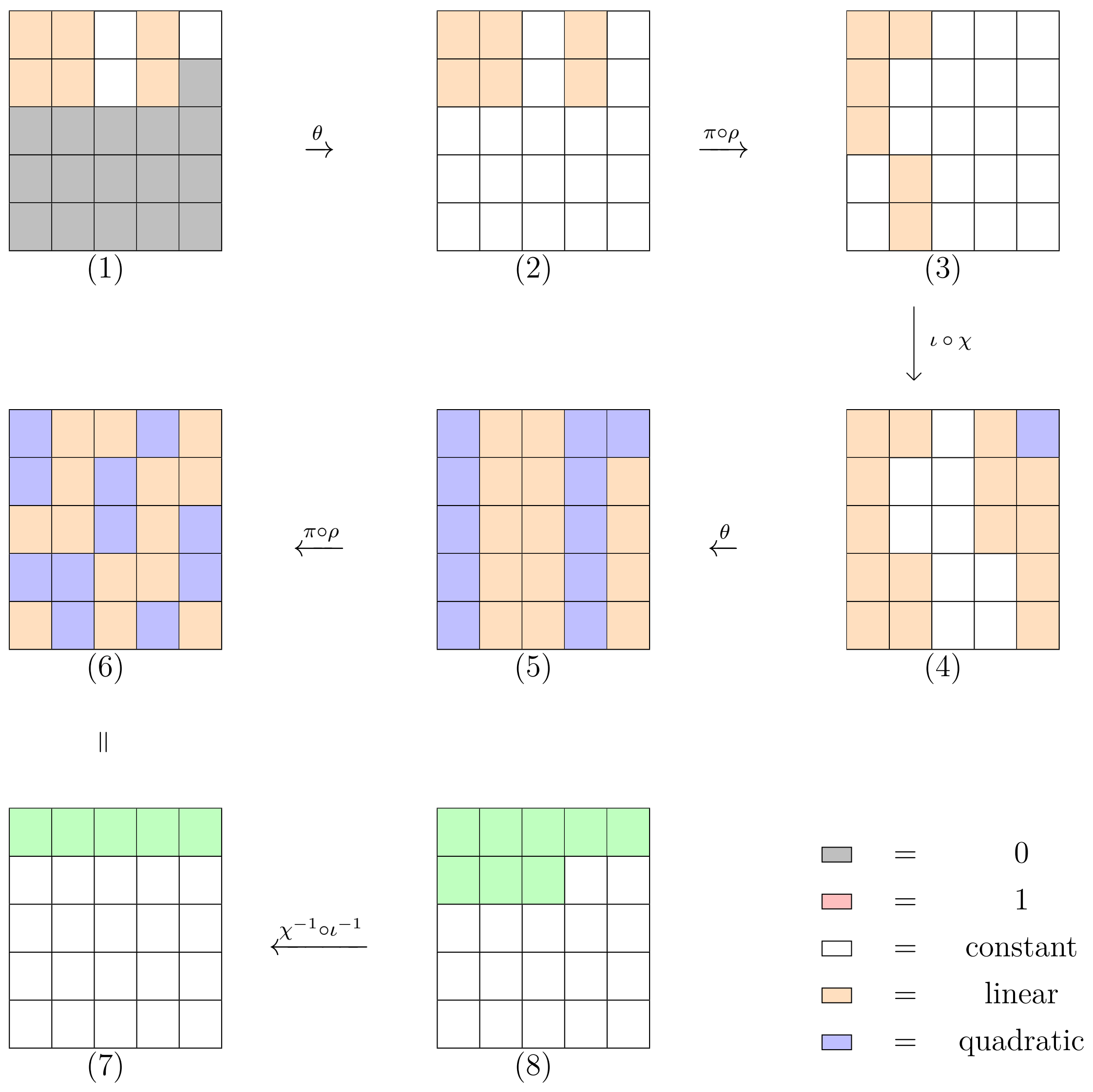

Figure 4.8: Better preimage attack on 2-round Keccak-512

to 384 bits. Therefore, we have a preimage attack for 2-rounds KECCAK-512 with a complexity of 2^{321}.

4.3 Preimage Attack on 2 Rounds KECCAK-384

The preimage attack given by Guo et al. [GLS16] for 2 rounds KECCAK-384 has a complexity of 2^{129} by constructing a linear structure with 6×64 variables. In our attack, we use 8×64 variables as shown in Figure 4.9. In order to avoid propagation by θ in first round, we add the following 3×64 linear constraints into the system, $\bigoplus_{j=0}^{4} A[i,j] = \alpha_i, \forall i \in [0,2,3]$ where α_i is some constant.

By equating state (5) and state (6), we get $2 \times 64 = 128$ linear equations between variables and hash values. Observe that we have only set up 320 linear equations but have $8 \times 64 = 512$ variables. Applying the linearization technique used in Section 4.2, we can set up an additional 3×64 linear equations of which $3\lfloor(3 \times 64)/4)\rfloor = 144$ equations are between message bits and hash values. After satisfying the padding rule, we have a complexity gain over brute force of $2^{128+144-1} = 2^{271}$ and hence a preimage attack of complexity $2^{384-271} = 2^{113}$. Observe that we have enough degrees of freedom to repeat this procedure for 2^{113} times. Note that this result cannot be compared with the preimage attack given by Kumar et al. [KMS18] because their attack has a space complexity of 2^{87}.

4.4 Preimage Attack for Higher Rounds

In the previous subsections, we were able to get a better preimage attack due to the fact the states are not filled with quadratic terms. If we were to find a similar attack for 3-rounds, we need to keep the following guidelines in mind.

1. The state after the application of second θ must be sparse of lanes with linear terms and comprised mostly of lanes with constant terms. This is because it would lead to a state with lesser quadratic terms after the application of χ of the second round.

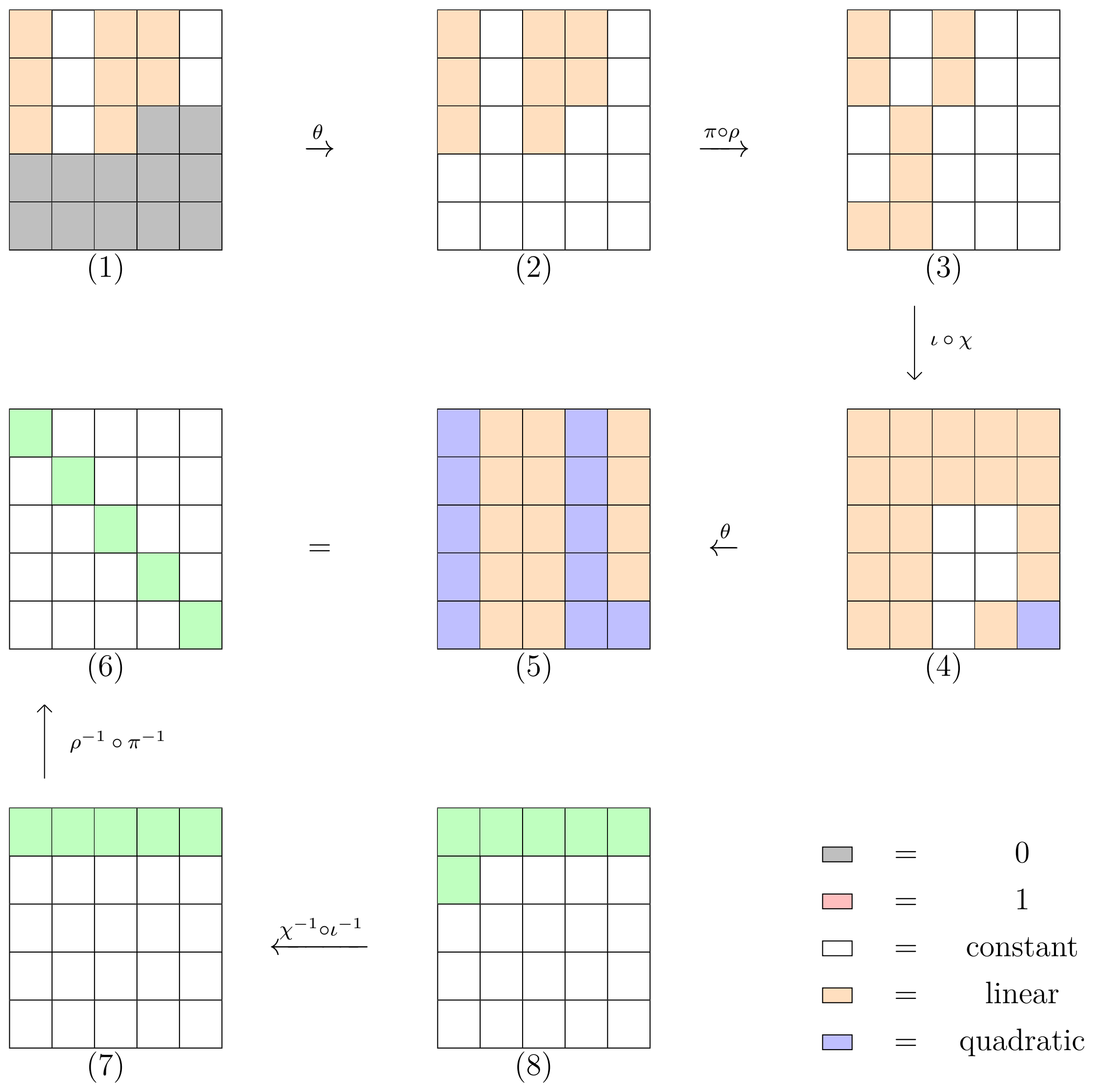

Figure 4.9: Preimage attack on 2-round Keccak-384

2. Even if the propagation due to the θ in the third round cannot be restricted, the state before the application of the third θ must contain all its quadratic terms either in a single column or in two columns adjacent to each other. This would lead to a state with at least one column containing linear terms only after the application of θ.

4.5 Preimage Attack on 3 Rounds Keccak-384

The following is our attack on Keccak-384 for 3 rounds which uses two message blocks as shown in Figure 4.10. The first message block is chosen in such a way that after the application of 3 round Keccak on this block, we get a state such that $A[1,3] = A[3,3] = 0$ and $A[1,4] = A[4,4] = 1$ where A is state (2) as shown in Figure 4.10. The first message block can be found by randomly choosing $2^{4\times 64}$ message block and expecting one of them to give the required output. This works because the output of a hash function is random, and therefore the complexity for brute force preimage attack is $1/2^l$ where l is the number of bits in the hash digest. The same technique has been used in [LS19] subsection 4.3.

The second message block contains $6 \times 64 = 384$ variables. We want to keep columns 2, 4 and 5 unchanged after the application of the first θ. For this, we first set $\bigoplus_{j=0}^{4} A[i,j] = \alpha_i$, for $i \in [0,2]$ and then set up an equation between column 1 and column 3 so that column 2 does not get affected after the application of first θ. This means that the α_i's are dependent. Similarly, c_2 and c_3 can be set according to α_i's such that columns 4 and 5 do not get affected after the first θ. Therefore, we have 2×64 linear equations in our system. c_1 can be fixed to some randomly chosen value.

To avoid propagation after second θ, we set up 3×64 linear equations to make the column parties equal to some constant β_i. Observe that after the application of the second χ, there are two lanes with quadratic terms in state (8). But after the application of the third θ, the fourth column will contain only linear terms. By equating state (9) and state (10), we can set up 63 linear equations between message

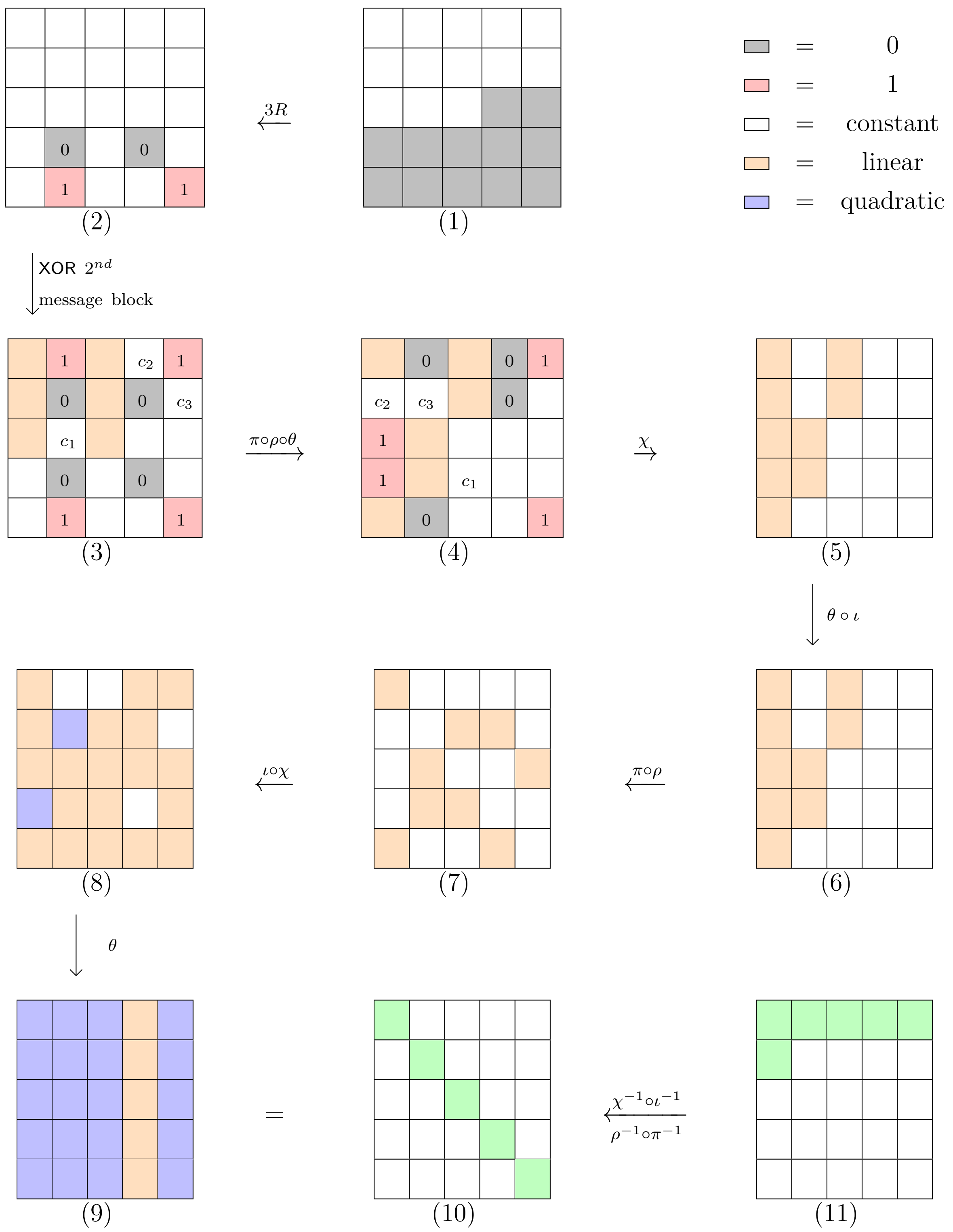

Figure 4.10: Preimage attack on 3-round Keccak-384

bits and hash values. Also, we have one more equation to keep the last message bit equal to 1. Therefore, we have a preimage attack with a time of complexity $2^{384-63} = 2^{321}$ because computing the first message block has a complexity of 2^{256}.

Note that there are enough degrees of freedom due to the 256 bits from α_i's and the β_i's, 64 bits from c_1 and enough bits from the first message block.

4.6 Preimage Attack on 3 Rounds Keccak-512

We use two message blocks and $4 \times 64 = 256$ variables for this attack as shown in Figure 4.11. The first message block is used so that we get enough degree of freedom to launch a preimage attack. Observe that after the application of θ in the first round, we require certain lanes to be 1/0 in state (4). To achieve this, we first set $A[1,0] \oplus A[1,1] = \alpha_1$ where α_1 is some constant. Then, we set up 64 linear equations of the form $\bigoplus_{i=0}^{4} (A[1,i] \oplus A[3,i](1) = e_2 + 1)$. Observe that due to this constraint, after the application of first θ, we will get $A[2,0] = A[2,4] = 1$ and $A[2,1] = 0$ where A is state (4). Similarly, by fixing x_6 and x_2 appropriately, we can get the required state (4).

To avoid propagation due to the θ in the second round, we add only 64 linear equations to the system to make the parity of the first columns in state (6) as a constant. Observe that after the application of θ of the third round, the lanes in the first two columns will contain only one quadratic term. So, if we linearize one quadratic term in $A[2,4]$ of state (9), then we have linearized five polynomials in column 2 of state (10). Similarly, if we linearize one quadratic term in $A[4,2]$ of state (9), then we have linearized five polynomials in column 1 of state (10).

But, out of these 6 linearized polynomials, only one can be used to create a linear equation between message bits and hash value by equating state (10) and state (11). Therefore, we have $\lfloor 64/2 \rfloor = 32$ linear equations between message bits and hash value and hence obtained a preimage of complexity $2^{512-32+1} = 2^{481}$. Due to the first message block, we have enough degree of freedom.

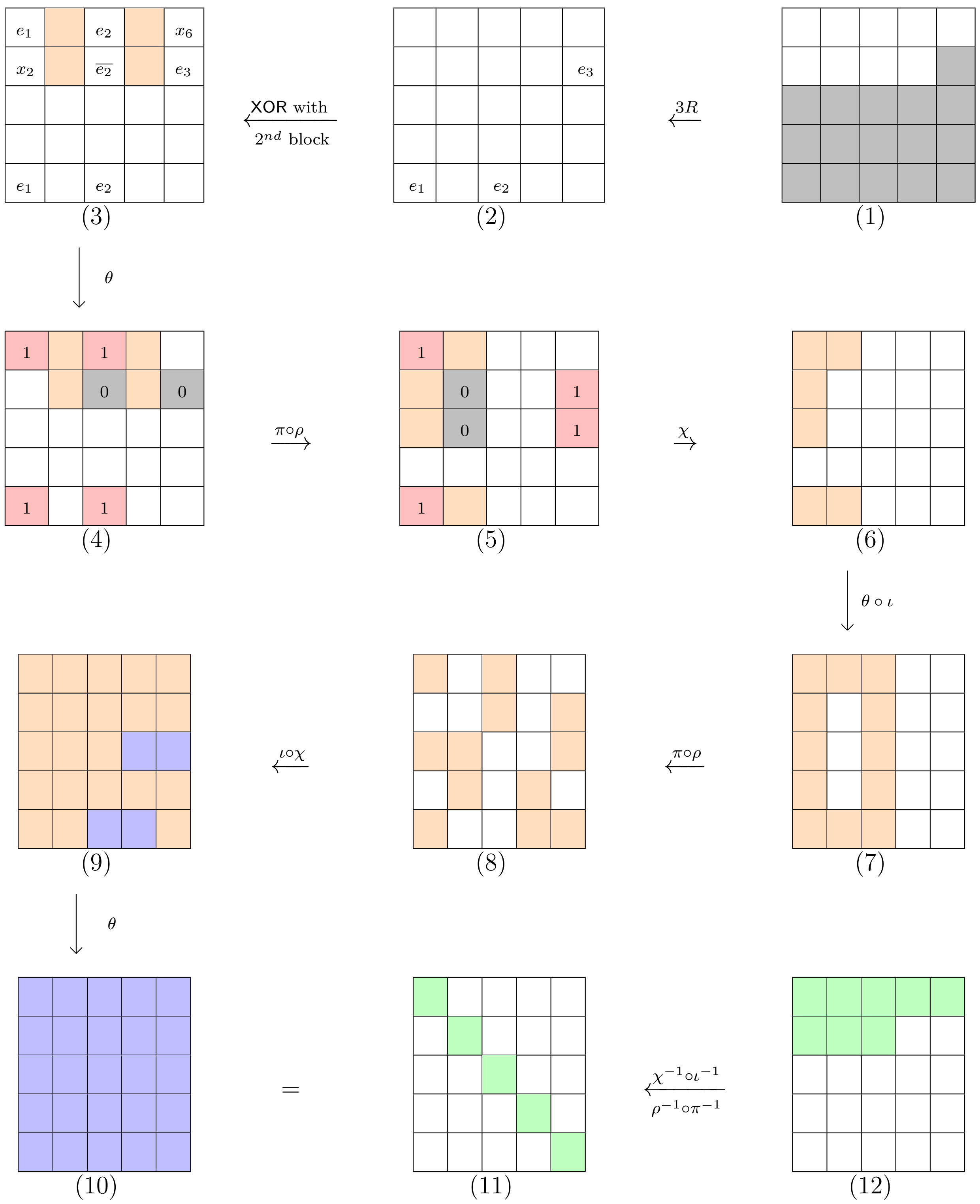

Figure 4.11: Preimage attack on 3-round KECCAK-512

4.6.1 Improved Analysis

Observe that if we carefully linearize one quadratic term from $A[2,4]$ and one from $A[4,2]$ of state (9), we also linearize one more polynomial in column 4 of state (10), i.e. we have also linearized a polynomial in the lane $A[3,3]$. Therefore, now we have $3\lfloor\frac{64}{5}\rfloor + 2 = 3 \times 12 + 2 = 38$. Therefore, we have an improved preimage attack of complexity $2^{512-38+1} = 2^{475}$.

4.7 Preimage Attack on 4 Rounds Keccak-384

This attack requires two message blocks and $6 \times 64 = 384$ variables as shown in Figure 4.12. As done in Section 4.5, the first message block is found by trying randomly many message blocks so that after the application of 4-rounds and XOR-ing the second message block, we get state (2). Observe that in state (2), there are two lanes with entries c and $\bar{c}$. We also require state (2) to satisfy one more equation.

$$d(-1) + \bar{b}(-2) + (g(-1) + (\bar{c} + a + b)(-2))(-2) + (a + b)(1) = \bar{k} \tag{4.1}$$

Therefore, we would require a complexity of 2^{128} to find the appropriate first message block. We will use the following strategy to obtain state (3). We include $A[0,0] = A[0,2]$ to the system of linear equations, fix $x_1 = 0$ and randomly assign value to x_7 whereas we fix $x_2 = \bar{c}, x_3 = d, x_5 = g$. Since we require state (3) after the application of θ, we have the following equations.

$$(a + b) + (A[2,0] + A[2,2] + e)(1) = \bar{c} \tag{4.2}$$

$$(A[2,0] + A[2,2] + e) + (x_6 + x_7 + i + j + k)(1) = g \tag{4.3}$$

$$(x_6 + x_7 + i + j + k) + (A[1,0] + A[1,2] + c)(1) = \bar{b} \tag{4.4}$$

$$(A[1,0] + A[1,2] + c) + (x_4 + f + h)(1) = d \tag{4.5}$$

$$(x_4 + f + h) + (a + b)(1) = \overline{k} \tag{4.6}$$

Therefore, we add equation (7) and (9) to the system of equations and fix x_6 and x_4 according to equation (8) and (10). Observe that due to the following equations, all equations from (2)-(6) are satisfied; particularly, equation (6) is satisfied due to equation (1).

$$A[2, 0] + A[2, 2] = (\overline{c} + a + b)(-1) + e \tag{4.7}$$

$$x_6 = g(-1) + (\overline{c} + a + b)(-2) + x_7 + i + j + k \tag{4.8}$$

$$A[1, 0] + A[1, 2] = \overline{b}(-1) + (g(-1) + (\overline{c} + a + b)(-2))(-1) + c \tag{4.9}$$

$$\begin{aligned} x_4 &= d(-1) + f + h + (\overline{b} + x_6 + x_7 + i + j + k)(-2) \\ &= d(-1) + f + h + \overline{b}(-2) + (g(-1) + (\overline{c} + a + b)(-2))(-2) \end{aligned} \tag{4.10}$$

Also, we include 2×64 linear equations for restricting the propagation due to θ in the second round. Observe that each polynomial in the state (9) has 11 quadratic terms. In [GLS16] subsection 6.3, Guo et al. gave a technique that carefully linearizes the quadratic terms such that if the number of free variables is t, we can construct $2\lfloor (t-5)/8 \rfloor$ linear equations between hash values and the variables. Let A denote state (8), B denote the state after χ of third round and C denote the state after θ of fourth round. From the definition of χ and θ and neglecting ι step for the sake of simplicity,

$$B[x, y, z] = A[x, y, z] \oplus (A[x + 1, y, z] \oplus 1) \cdot A[x + 2, y, z]$$

$$C[x, y, z] = B[x, y, z] \oplus \bigoplus_{y'=0}^{4} B[x - 1, y', z] \oplus \bigoplus_{y'=0}^{4} B[x + 1, y', z - 1]$$

We can linearize $B[x - 1, y, z]$ and $B[x, y, z]$ by guessing the value of $A[x + 1, y, z]$ for $0 \le y \le 4$. Similarly, we can linearize $B[x + 1, y, z - 1]$ and $B[x + 2, y, z - 1]$ by guessing the value of $A[x + 3, y, z - 1]$ for $0 \le y \le 4$. This helps us in linearizing

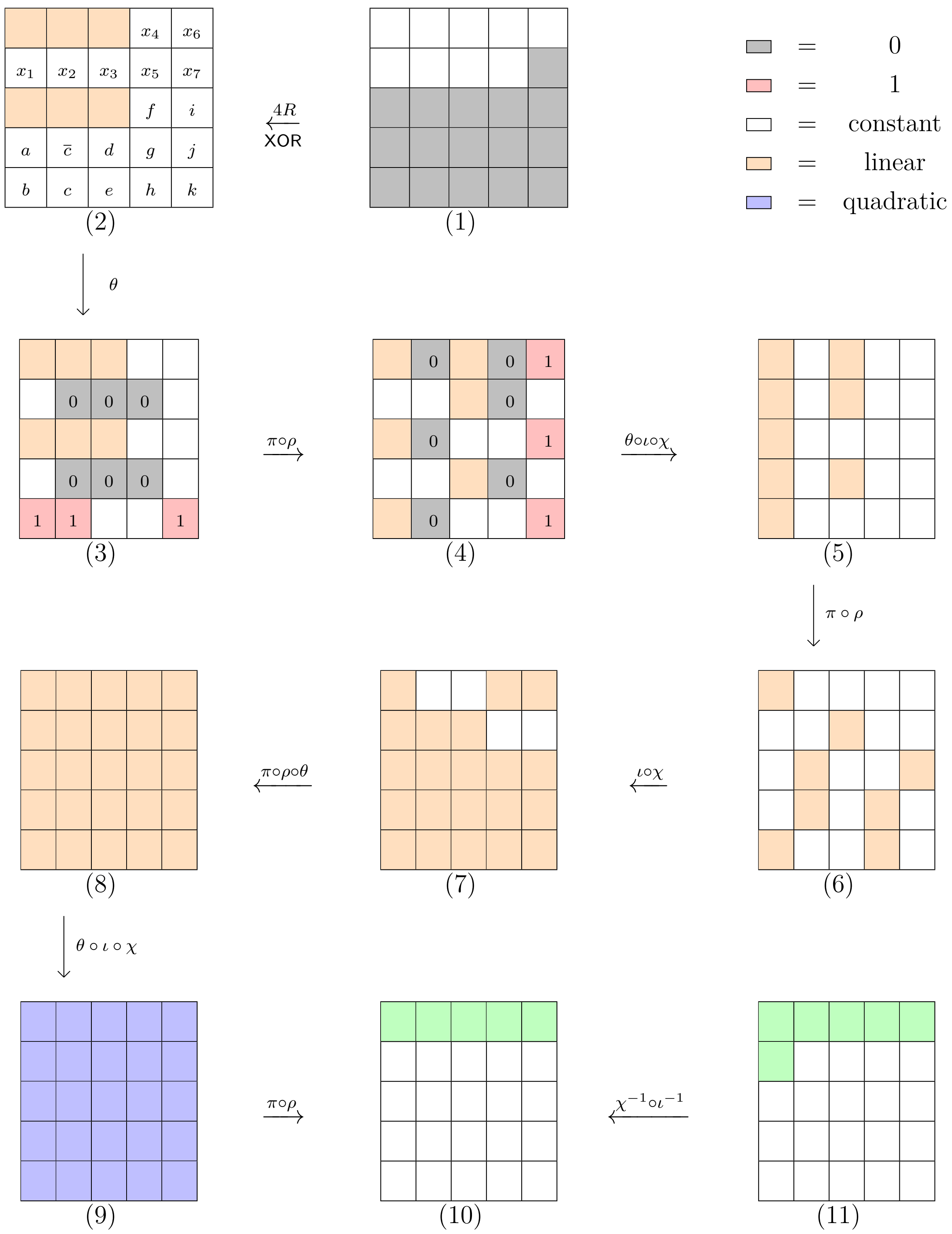

Figure 4.12: Preimage attack on 4-round KECCAK-384

$C[x,y,z]$, but observe that

$$C[x+1,y+1,z] = B[x+1,y+1,z] \oplus \bigoplus_{y'=0}^{4} B[x,y',z] \oplus \bigoplus_{y'=0}^{4} B[x+2,y',z-1]$$

which contain a quadratic part in $B[x+1,y+1,z]$. By linearizing this term, we set up 13 linear equations of which two equations are between message bits and hash values. Similarly, by carefully observing $C[x+2,y+2,z-1]$ and $C[x+3,y+3,z-1]$ and linearizing them, we can set up another 8 linear equations of which two equations are between message bits and hash values. For more details, refer [GLS16]. In our case, the number of free variables $t = 64$ and therefore, we can set up 14 linear equations between message bits and hash values. Observe that we have enough degree of freedom due to x_7, the parity of the two columns of the second θ and the rest from the first message block. Therefore, the complexity of our attack is 2^{371}.

Chapter 5

Weak-Keys and Key-Recovery Attacks for TinyJAMBU

In this chapter, we study TinyJAMBU from three important and different contexts – (i) the weak-key setting, (ii) understanding the exact degree of the feedback polynomial in the nonce variables (iii) the key-recovery attacks. Teng et al. [Ten+21] looked into the TinyJAMBU cipher's resistance to cube attacks. They showed key-recovery attack for 428 rounds and distinguishing attack for 438 rounds using small size cubes.

We show that there are at least 2^{108} keys for which TinyJAMBU can be distinguished from a random source for up to 476 rounds and key-recovery attack against 440 rounds.

5.1 Structure of TinyJAMBU

Wu and Huang designed TinyJAMBU [WH19; WH21] which is a variant of JAMBU [WH14]. It is a family of lightweight authenticated encryption algorithms and one among the 10 finalists in the NIST Lightweight Cryptography (LWC) Standardisation project [NIS18]. The family consists of three variants - TinyJAMBU-128, TinyJAMBU-192 and TinyJAMBU-256, as shown in Table 5.1.

The core permutation $\mathcal{P}_n$ updates the state using a non-linear feedback shift

Table 5.1: TinyJAMBU variants and their recommended parameters

Name	State size	Size of			Rounds	
		Key	Nonce	Tag	$\mathcal{Q}$	$\hat{\mathcal{Q}}$
TinyJAMBU-128	128	128	96	64	640	1024
TinyJAMBU-192	128	192	96	64	640	1152
TinyJAMBU-256	128	256	96	64	640	1280

register for n-rounds where the i^{th} rounds is described in Algorithm 1. We will use the notation $\mathcal{P}$ for $\mathcal{P}_1$. The permutations $\mathcal{Q}$ and $\hat{\mathcal{Q}}$ mentioned in Table 6.1 are $\mathcal{P}_n$ with different values of n.

```
$feedback = s_0 \oplus s_{47} \oplus \sim (s_{70} \cdot x_{85}) \oplus s_{91} \oplus k_{i \bmod klen}$;
for $i = 1$ to 127 do
    $s_i = s_{i-1}$;
end
$s_0 = feedback$;
```

Algorithm 1: StateUpdate(s, k, i)

5.1.1 Specification of TinyJAMBU-128

As shown in Table 6.1, TinyJAMBU-128 has a 128-bits state and key, whereas the number of nonce bits is 96. The tag size is 64 bits. The permutation function $\mathcal{Q}$ is $\mathcal{P}_{640}$ whereas $\hat{\mathcal{Q}}$ is $\mathcal{P}_{1024}$.

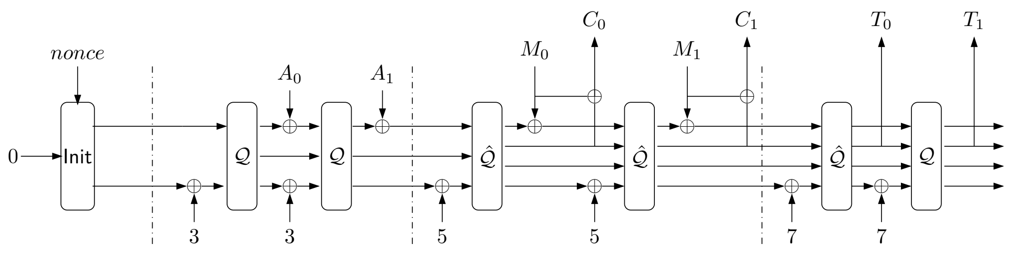

Figure 5.1: TinyJAMBU's mode of operation (encryption phase)

The authenticated encryption algorithm of TinyJAMBU can be divided into four phases as shown in Figure 5.1: Initialisation, associated data processing, encryption and finalisation.

Initialization: In this phase, the state is set to all zero and is updated by the keyed permutation $\hat{\mathcal{Q}}$. The 96 bit nonce is divided into 3 parts- $nonce_0, nonce_1$ and $nonce_2$ of equal size and updates the state using Algorithm 2.

```
for $i = 0$ to $2$ do
    $s_{36,\ldots,38} = s_{36,\ldots,38} \oplus 1$;
    Update state $s$ using $\mathcal{Q}$;
    $s_{96,\ldots,127} = s_{96,\ldots,127} \oplus nonce_i$;
end
```

Algorithm 2: NonceSetup$(s, nonce)$

Associated date processing: After the initialisation phase, the associated date (A_0, A_1) are processed by XORing them to the state and updating it using the keyed permutation $\mathcal{Q}$ as described in Algorithm 3

```
for $i = 0$ to $1$ do
    $s_{36,\ldots,38} = s_{36,\ldots,38} \oplus 3$;
    Update state $s$ using $\hat{\mathcal{Q}}$;
    $s_{96,\ldots,127} = s_{96,\ldots,127} \oplus A_i$;
end
```

Algorithm 3: AssociatedDateProcessing(s, A_0, A_1)

Encryption: In the encryption phase, the message M is encrypted to produce the ciphertext C. In the paper, we will always assume that the size of the message M denoted by ℓ is a multiple of 32. In Figure 5.1, we have assumed that M consists of 64 bits. In general, Algorithm 4 is performed $\ell/32$ times on $M_0, \ldots, M_{\ell/32-1}$ where M_i is the i^{th} block of M having size equal to 32.

```
for $i = 0$ to $\ell/32$ do
    $s_{36,\ldots,38} = s_{36,\ldots,38} \oplus 5$;
    Update state $s$ using $\hat{Q}$;
    $s_{96,\ldots,127} = s_{96,\ldots,127} \oplus M_i$;
    $C_i = s_{64,\ldots,95} \oplus M_i$;
end
```

Algorithm 4: Encryption(s, M)

Tag generation: Finally, the 64-bit tag $T = (T_0, T_1)$ is generated as shown in Algorithm 5.

```
for $i = 0$ to $1$ do
    $s_{36,\ldots,38} = s_{36,\ldots,38} \oplus 7$;
    Update state $s$ using $\hat{Q}$;
    $T_i = s_{64,\ldots,95}$;
end
```

Algorithm 5: TagGeneration(s, M)

5.2 Weak-Keys for TinyJAMBU

In this section, we will present a new cube distinguisher for 451 rounds TinyJAMBU that works for 2^{101} keys. We also show another distinguisher for 476 rounds for 2^{80} keys. We will first analyse the feedback polynomial in the permutation $\mathcal{P}$ which will help to find *good* cubes. Using these cubes, we will further study the key variables involved in the feedback polynomials, which will help us to find weak-keys for TinyJAMBU.

To be more precise, let us consider the scenario of a traditional cube attack as a distinguisher. Let $\boldsymbol{x}$ be the set of all public (nonce) variables, $\mathbf{k}$ be the set of key variables and $f(\boldsymbol{x}, \mathbf{k})$ be the output polynomial. Then, for any cube $\mathcal{C}_{\mathbf{I}}$ where $\mathbf{I} \subseteq [n]$

is the index set and n is the number of public variables, we have

$$f(\boldsymbol{x}, \mathbf{k}) = \left(\prod_{i \in \mathbf{I}} x_i \cdot p_{\mathbf{I}}(\boldsymbol{x}, \mathbf{k}) \right) + q_{\mathbf{I}}(\boldsymbol{x}, \mathbf{k}),$$

where $p_{\mathbf{I}}(\boldsymbol{x}, \mathbf{k})$ *does not* contain any variable x_i, where $i \in \mathbf{I}$ and every monomial of $q_{\mathbf{I}}(\boldsymbol{x}, \mathbf{k})$ is not divisible by $\prod_{i \in \mathbf{I}} x_i$. In a cube attack, one takes advantage of the fact that for a properly chosen cube $\mathcal{C}_{\mathbf{I}}$ and constant values $a_i, \forall i \in [n] \setminus \mathbf{I}$, the superpoly of the output polynomial $f(\boldsymbol{x}, \mathbf{k})$ is a *constant*, i.e., $p_{\mathbf{I}}(\mathbf{A}, \mathbf{k})$ is a constant, where $\mathbf{A} = \{a_i \mid i \in [n] \setminus \mathbf{I}\}$. Observe that for $p_{\mathbf{I}}(\mathbf{A}, \mathbf{k})$ be to a constant, its degree with respect to $\mathbf{k}$ must be 0.

Therefore, the above attack fails when there is no possible cube with appropriate constant values that has a superpoly equal to a constant. This happens when the number of rounds increases because the degree of the polynomials with respect to both public and key variables increases exponentially. Nevertheless, if we can find the superpoly $p_{\mathbf{I}}(\mathbf{A}, \mathbf{k})$ which is a polynomial in the key variables only, then we can use this information to find *weak-keys* such that $p_{\mathbf{I}}(\mathbf{A}, \mathbf{k})$ is a constant. These weak-keys are essentially ones that satisfy the equation $p_{\mathbf{I}}(\mathbf{A}, \mathbf{k}) = \mathit{constant}$.

But, to find out the superpolies is itself a difficult task. Using MILP, one can find superpoly. However, this technique is only possible for small rounds because the degree of the superpolies will be small. Therefore, instead of extracting the exact superpolies, we will show a method to decrease the degree of the superpoly by imposing some constraints on the key variables. By doing so, we will get weak-keys such that the superpoly is a constant with some probability.

5.2.1 Weak-Key Attacks using Cubes from [Ten+21]

In [Ten+21], the authors presented several cube attacks for TinyJAMBU. In their experiments, the authors devised five different cube attacks, DA1 to DA5, depending on the scenario. DA1 and DA2 are attacks against the initialisation phase, i.e.,

- DA1 uses only the first 64 bits of the nonce (i.e., nonce_0 and nonce_1 only) and can observe the keystream after the initialisation phase. But, the total number of permutation rounds being considered is $1024 + 3 \times 384$. This is according to [WH19], but Wu and Huang have updated the initialisation phase in [WH21] from using $\mathcal{P}_{384}$ to $\mathcal{P}_{640}$. This change was made after a recent differential forgery attack due to [Sah+20].

- DA2 uses all 96 bits of the nonce but observes the keystream after applying r rounds of permutation after the initialisation phase.

In DA3 to DA5, the authors use the plaintext bits to build new distinguishers. Since our focus is towards building distinguishers based on nonce bits only; we will be skipping DA3 to DA5 in further discussions.

In Table 5.2, we have given the best cubes mentioned in [Ten+21] for DA2 where the authors have used cube indices from 64 to 95, i.e., only $nonce_2$. The zero-sum value is observed at the 64^{th} index of the final state.

Cube-size	Cube indices	Rounds
8	69, 70, 71, 76, 81, 86, 91, 92	419
14	64, 65, 70, 72, 76, 77, 78, 81, 85, 86, 87, 88, 92, 95	435
18	66, 67, 68, 72, 73, 75, 77, 79, 81, 82, 83, 84, 87, 88, 89, 90, 93, 94	437
18	67, 68, 69, 70, 72, 73, 75, 79, 80, 81, 83, 84, 85, 88, 89, 90, 91, 95	438

Table 5.2: Cube attacks from [Ten+21]

In our experiments, we will be considering distinguishing attacks (as well as key-recovery in the next section) for the keyed permutation $\mathcal{P}_n$. The experiment is as follows.

1. We start with a state with all the bits set to 0.
2. The bits that are accessible to the attacker are $\{96, 97, \ldots, 127\}$, i.e., the nonce bits. (Refer to Algorithm 2).
3. After the public bits are set, the state is updated using $\mathcal{P}_n$.

4. Finally, the attacker is only given access to the bits at positions $\{64, 65, \ldots, 96\}$.

The designers of TinyJAMBU [WH21] have mentioned that, from their experiments, every nonce bit affects all the output bits after 512 rounds. Hence, building distinguishers using 32-sized cubes is not possible.

We use a 7 size cube $\{69, 70, 76, 81, 86, 91, 92\}$. In our simulations, we use a linear congruential algorithm and 48-bit integer arithmetic in C to generate random binary values. We use 1 million random keys to find the bias. We get $\Pr(\text{Superpoly} = 1) = 0.285$ (0.485) for 434 (456) rounds.

We present our experimental results in Figure 5.2.

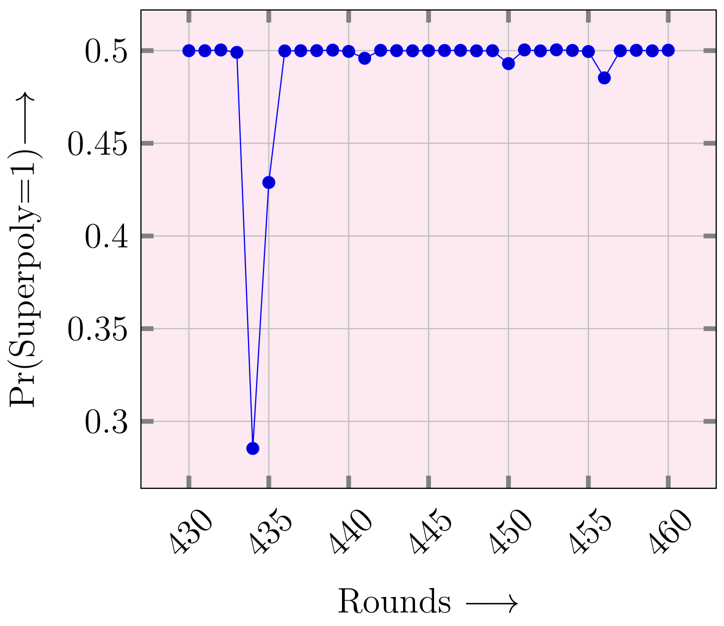

Figure 5.2: Probability of superpoly being 1 for different rounds for the cube $\{69, 70, 76, 81, 86, 91, 92\}$.

5.2.2 How to Find Good Cubes

The state bits of TinyJAMBU are updated using the nonce, associated data and plaintext using the permutation $\mathcal{P}$. Let us recall the feedback polynomial in permutation $\mathcal{P}$. It is a quadratic polynomial involving 5 state variables and 1 key variable. The only quadratic term in the polynomial is $s_{70} \cdot s_{85}$.

For the (weak-key) distinguishing attack, we will start with a state s with all 0 entries. As mentioned in Section 5.1.1, the 32 bit nonce is XORed at the 96^{th}

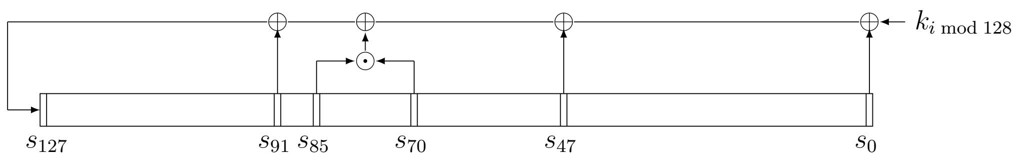

Figure 5.3: Feedback polynomial in $\mathcal{P}$

position of the state. In any distinguishing attack, the desired cube must have two *good* properties.

1. The size of the cube must be small.

2. The number of rounds it can be used as a distinguisher must be large.

To achieve these properties simultaneously, one way is to make sure that the degrees of the polynomials in the state are as small as possible for the initial rounds. Observe that the degree of the polynomials in the state can increase only due to the quadratic term in the permutation $\mathcal{P}$ function. Therefore, if we can control the quadratic term in the initial rounds, the degree of the polynomial can be minimised.

Since the *feedback* polynomial has a non-linear term $s_{70} \cdot s_{85}$ (i.e., the terms are 15 indices apart) we just need to ensure that in the initial rounds, we can *minimise* the number of index pairs $(i, i+15)$ such that at most one among s_i or s_{i+15} contains a nonce variable. In doing so, the *feedback* polynomial will become linear for up to certain rounds. Therefore, if we set a cube $\mathcal{C} = \{c_i \mid 0 \leq i < 32\}$ such that $(c_i, c_{i+15}) \nsubseteq \mathcal{C}, \forall i \in \{0, 1, \ldots, 16\}$, then we can ensure for a larger number of rounds, the state will contain linear polynomials only.

For example, suppose we consider two cube $\mathcal{C}$ and $\mathcal{C}'$ such that $(c_0, c_{15}) \subseteq \mathcal{C}$ whereas $c_0 \in \mathcal{C}', c_{15} \notin \mathcal{C}'$. As mentioned earlier, we will start with a state with all 0 bits and the 32-bit nonce is XORed at index 96^{th}. This implies that at round 0, we have $s_{96} = c_0$ and $s_{111} = c_{15}$ while considering the cube $\mathcal{C}$. Also, $s_i = 0$ for all $0 \leq i < 96$. After 26 rounds, the state would get updated in such a way that $s_{70} = c_0$ and $s_{85} = c_{15}$. Observe that all the polynomials of the state are linear up to this round. But, in the 27^{th} round, the feedback polynomial will contain the term $c_0 \cdot c_{15}$ and s_{127} will become quadratic. When we consider the cube $\mathcal{C}'$, then after 26 rounds, we will get $s_{70} = c_0$ and s_{85} will be some constant. Therefore, s_{127} in the 27th will also be *linear*.

5.2.3 Getting Cubes for Higher Rounds

The best distinguisher mentioned in [Ten+21] works for 438 rounds and uses a cube of size 18. In this subsection, we will present weak-keys for TinyJAMBU where the number of rounds is greater than 450. To begin with, we will use the cubes mentioned in Section 5.2.2, which gives us an advantage over other cubes by making the feedback polynomials linear for more rounds.

After picking a good cube, we will follow the steps mentioned in Section 5.2.1 to find key constraints so that we can build better distinguishers, i.e., we will carefully analyse the feedback polynomial at each round and try to set up constraints on the key variables so that either we are able to decrease the degree of the polynomial with respect to nonce variable or completely eliminate a term from the polynomial.

For example, let us consider the 14 sized cube $\mathcal{C}_{\mathbf{I}}$ where

$$\mathbf{I} = \{65, 66, 67, 68, 69, 70, 71, 72, 73, 74, 75, 76, 77, 78\}$$

The rest of the nonce variables are set to 0. Observe that the set $\mathbf{I}$ has the required property that ensures that $\mathcal{C}_{\mathbf{I}}$ is a good cube, i.e., $(i, i+15) \not\subseteq \mathbf{I}$. At the 59th rounds, the feedback polynomial is

$$k_0 \cdot k_{15} + k_0 \cdot x_{10} + k_0 + k_{15} + k_{21} + k_{58} + x_9 + x_{10} + 1$$

By setting $k_0 = 1$, the polynomial gets reduced to

$$k_{21} + k_{58} + x_9 + 1$$

Observe that even though the degree of the polynomial with respect to nonce variable has not decreased, the nonce variable x_{10} does not appear in the polynomial.

This allows us to control the mixing of nonce variables in subsequent rounds. Similarly, we set $k_1 = k_2 = k_3 = k_4 = 1$ for the next four rounds.

At the 65th round, the feedback polynomial is

$$k_6 \cdot k_{21} + k_6 + k_{21} \cdot x_1 + k_{21} + k_{27} + k_{64} + x_1 + 1$$

In this case, if we set $k_{21} = 1$, the polynomial gets reduced to $k_{27} + k_{64}$, i.e., the degree of the polynomial with respect to nonce variable gets reduced to 0 as well as the number of monomials. By analysing the feedback polynomial in the next 13 rounds, we set $k_i = 1, \forall i \in \{22, 23, \ldots, 34\}$.

Let us now focus on the 94th round. Here, we have

$$k_{12} + k_{13} \cdot k_{35} + k_{13} + k_{19} + k_{35} \cdot k_{50} + k_{35} \cdot x_1 + k_{35} \cdot x_8 + k_{50} + k_{56} + k_{93} + x_1 + x_8 + x_{14}$$

We set $k_{35} = 1$ to reduce the feedback polynomial. The same goes for $k_i, \forall i \in \{35, 36, \ldots, 42\}$. By analysing a few rounds after the 102nd round, we set $k_i = 1, \forall i \in \{58, 59, 60, 61, 62\}$. Giving an addition 10 key constraints $k_i = 1, \forall i \in \{64, \ldots, 73\}$, we get a zero sum distinguisher for 451 rounds. Thus, we have a total of 41 constraints on key for this case.

Next, we relax a few conditions on the key and check experimentally whether we are getting bias or not. We check that if we take $k_i = 1,\ \forall i \in \{3, 4, 24, 26, 28, 29, 33, 34, 35, 40, 41, 71, 72\}$, still we get zero sum distinguisher for 451 rounds. Thus instead of 41 constraints, we now need only 13 constraints.

Cube indices	I	Rounds	Prob.
65, 66, 67, 68, 69, 70, 71, 72, 73, 74, 75, 76, 77, 78	{3, 4, 24, 26, 28, 29, 33, 34, 35, 40, 41, 71, 72}	451	0
64, 65, 66, 68, 69, 70, 71, 72, 73, 74, 75, 76, 77, 78	{0, 1, 2, 3, 4, 21, 23, 24, 28, 29, 30, 32, 36, 37, 38, 39, 40, 42}	455	0.476
64, 65, 66, 67, 68, 70, 71, 72, 73, 74, 75, 76, 77, 78	{ 2, 3, 20, 21, 22, 25, 26, 27, 28, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41, 42, 57, 58, 59, 61, 64, 66, 70, 71, 72, 73, 74, 76, 77}	466	0.467
81, 82, 83, 84, 85, 86, 87, 88, 89, 90, 91, 93, 94, 95	{0, 1, 6, 7, 8, 13, 14, 15, 16, 17, 20, 37, 38, 41, 44, 45, 50, 51, 52, 57}	476	0.478

Table 5.3: Weak-key attacks for more than 450 rounds. Constraints are $k_i = 1$ for $i \in \mathbf{I}$. Fourth column represents the probability of the superpoly being 1.

In Table 5.3, we present our experimental results. We use 10^5 random keys to find the probabilities. All cubes are of size 14. One can see from the first row that superpoly is always equal to 0 for 451 rounds if the key satisfies 13 constraints. From the last row, it is clear that one can distinguish TinyJAMBU for 476 rounds if the secret key satisfies 20 constraints. Thus the size of the corresponding weak-key class is 2^{108}.

5.3 Exact Degree in Nonce of the Feedback Polynomial in $\mathcal{P}_n$

In this section, we will use the concept of monomial trails to find the exact degrees in nonce of the feedback polynomials for some rounds of TinyJAMBU. To find the exact degrees, we will use the algorithms mentioned in [Hu+20] and discuss the important changes required to adapt those algorithms to our scenario.

The exact degree of a polynomial $f(\boldsymbol{x})$ can be written as $deg(f) = \max\limits_{\pi_{\mathbf{u}}(\boldsymbol{x}) \rightarrow f(\boldsymbol{x})} (\mathsf{wt}(\mathbf{u}))$. Using Claim 2.14, we have a procedure to find the exact degree of $f(\boldsymbol{x})$ given as follows.

1. Find a highest degree monomial $\pi_{\mathbf{u}}(\boldsymbol{x})$ such that $\pi_{\mathbf{u}}(\boldsymbol{x}) \rightsquigarrow f(\boldsymbol{x})$.

2. If $|\pi_{\mathbf{u}}(\boldsymbol{x}) \rightsquigarrow f(\boldsymbol{x})|$ is odd, then return $\mathsf{wt}(\mathbf{u})$. Else, repeat the whole process until we find a new monomial whose degree is the largest and has an odd number of monomial trails.

We will use Mixed Integer Linear Programming (MILP) to find monomial trails and use Gurobi to solve the MILP instance. The objective function is to maximise the weight of $\mathbf{u}$ whereas the constraints will be set up in such that a solution $\mathbf{u}$ will ensure a monomial trail $\pi_{\mathbf{u}}(\boldsymbol{x}) \rightsquigarrow f(\boldsymbol{x})$. To count the number of monomial trails, we will use *PoolSearchMode* in Gurobi, which will help us to find multiple solutions. Here, we are using the fact that each solution of the model is a monomial trail $\pi_{\mathbf{u}}(\boldsymbol{x}) \rightsquigarrow f(\boldsymbol{x})$.

We now focus on the MILP model for finding the exact degree in nonce of the

$$
\begin{aligned}
\mathbf{V} = \{&[0,0,0,0,0,0,0,0,0,0,0,0], [0,0,0,0,0,1,0,0,0,0,0,1], [0,0,0,0,0,1,1,0,0,0,0,0], [0,0,0,0,0,1,1,0,0,0,0,1],\\
&[0,0,0,0,1,0,0,0,0,0,1,0], [0,0,0,0,1,0,1,0,0,0,0,0], [0,0,0,0,1,0,1,0,0,0,1,0], [0,0,0,0,1,1,0,0,0,0,1,1],\\
&[0,0,0,0,1,1,1,0,0,0,0,1], [0,0,0,0,1,1,1,0,0,0,1,0], [0,0,0,1,0,0,0,0,0,1,0,0], [0,0,0,1,0,1,0,0,0,1,0,1],\\
&[0,0,0,1,0,1,1,0,0,1,0,0], [0,0,0,1,0,1,1,0,0,1,0,1], [0,0,0,1,1,0,0,0,0,1,1,0], [0,0,0,1,1,0,1,0,0,1,0,0],\\
&[0,0,0,1,1,0,1,0,0,1,1,0], [0,0,0,1,1,1,0,0,0,1,1,1], [0,0,0,1,1,1,1,0,0,1,0,1], [0,0,0,1,1,1,1,0,0,1,1,0],\\
&[0,0,1,0,0,0,0,0,1,0,0,0], [0,0,1,0,0,1,0,0,1,0,0,1], [0,0,1,0,0,1,1,0,1,0,0,0], [0,0,1,0,0,1,1,0,1,0,0,1],\\
&[0,0,1,0,1,0,0,0,1,0,1,0], [0,0,1,0,1,0,1,0,1,0,0,0], [0,0,1,0,1,0,1,0,1,0,1,0], [0,0,1,0,1,1,0,0,1,0,1,1],\\
&[0,0,1,0,1,1,1,0,1,0,0,1], [0,0,1,0,1,1,1,0,1,0,1,0], [0,0,1,1,0,0,0,0,1,1,0,0], [0,0,1,1,0,0,1,0,0,0,0,0],\\
&[0,0,1,1,0,0,1,0,0,1,0,0], [0,0,1,1,0,0,1,0,1,0,0,0], [0,0,1,1,0,0,1,0,1,1,0,0], [0,0,1,1,0,1,0,0,1,1,0,1],\\
&[0,0,1,1,0,1,1,0,0,0,0,1], [0,0,1,1,0,1,1,0,0,1,0,1], [0,0,1,1,0,1,1,0,1,0,0,1], [0,0,1,1,0,1,1,0,1,1,0,0],\\
&[0,0,1,1,1,0,0,0,1,1,1,0], [0,0,1,1,1,0,1,0,0,0,1,0], [0,0,1,1,1,0,1,0,0,1,1,0], [0,0,1,1,1,0,1,0,1,0,1,0],\\
&[0,0,1,1,1,0,1,0,1,1,0,0], [0,0,1,1,1,1,0,0,1,1,1,1], [0,0,1,1,1,1,1,0,0,0,1,1], [0,0,1,1,1,1,1,0,0,1,1,1],\\
&[0,0,1,1,1,1,1,0,1,0,1,1], [0,0,1,1,1,1,1,0,1,1,0,1], [0,0,1,1,1,1,1,0,1,1,1,0], [0,0,1,1,1,1,1,0,1,1,1,1],\\
&[0,1,0,0,0,0,0,1,0,0,0,0], [0,1,0,0,0,0,1,0,0,0,0,0], [0,1,0,0,0,0,1,1,0,0,0,0], [0,1,0,0,0,1,0,1,0,0,0,1],\\
&[0,1,0,0,0,1,1,0,0,0,0,1], [0,1,0,0,0,1,1,1,0,0,0,0], [0,1,0,0,1,0,0,1,0,0,1,0], [0,1,0,0,1,0,1,0,0,0,1,0],\\
&[0,1,0,0,1,0,1,1,0,0,0,0], [0,1,0,0,1,1,0,1,0,0,1,1], [0,1,0,0,1,1,1,0,0,0,1,1], [0,1,0,0,1,1,1,1,0,0,0,1],\\
&[0,1,0,0,1,1,1,1,0,0,1,0], [0,1,0,0,1,1,1,1,0,0,1,1], [0,1,0,1,0,0,0,1,0,1,0,0], [0,1,0,1,0,0,1,0,0,1,0,0],\\
&[0,1,0,1,0,0,1,1,0,1,0,0], [0,1,0,1,0,1,0,1,0,1,0,1], [0,1,0,1,0,1,1,0,0,1,0,1], [0,1,0,1,0,1,1,1,0,1,0,0],\\
&[0,1,0,1,1,0,0,1,0,1,1,0], [0,1,0,1,1,0,1,0,0,1,1,0], [0,1,0,1,1,0,1,1,0,1,0,0], [0,1,0,1,1,1,0,1,0,1,1,1],\\
&[0,1,0,1,1,1,1,0,0,1,1,1], [0,1,0,1,1,1,1,1,0,1,0,1], [0,1,0,1,1,1,1,1,0,1,1,0], [0,1,0,1,1,1,1,1,0,1,1,1],\\
&[0,1,1,0,0,0,0,1,1,0,0,0], [0,1,1,0,0,0,1,0,1,0,0,0], [0,1,1,0,0,0,1,1,1,0,0,0], [0,1,1,0,0,1,0,1,1,0,0,1],\\
&[0,1,1,0,0,1,1,0,1,0,0,1], [0,1,1,0,0,1,1,1,1,0,0,0], [0,1,1,0,1,0,0,1,1,0,1,0], [0,1,1,0,1,0,1,0,1,0,1,0],\\
&[0,1,1,0,1,0,1,1,1,0,0,0], [0,1,1,0,1,1,0,1,1,0,1,1], [0,1,1,0,1,1,1,0,1,0,1,1], [0,1,1,0,1,1,1,1,1,0,0,1],\\
&[0,1,1,0,1,1,1,1,1,0,1,0], [0,1,1,0,1,1,1,1,1,0,1,1], [0,1,1,1,0,0,0,1,1,1,0,0], [0,1,1,1,0,0,1,0,1,1,0,0],\\
&[0,1,1,1,0,0,1,1,0,0,0,0], [0,1,1,1,0,0,1,1,0,1,0,0], [0,1,1,1,0,0,1,1,1,0,0,0], [0,1,1,1,0,1,0,1,1,1,0,1],\\
&[0,1,1,1,0,1,1,0,1,1,0,1], [0,1,1,1,0,1,1,1,0,0,0,1], [0,1,1,1,0,1,1,1,0,1,0,1], [0,1,1,1,0,1,1,1,1,0,0,1],\\
&[0,1,1,1,0,1,1,1,1,1,0,0], [0,1,1,1,0,1,1,1,1,1,0,1], [0,1,1,1,1,0,0,1,1,1,1,0], [0,1,1,1,1,0,1,0,1,1,1,0],\\
&[0,1,1,1,1,0,1,1,0,0,1,0], [0,1,1,1,1,0,1,1,0,1,1,0], [0,1,1,1,1,0,1,1,1,0,1,0], [0,1,1,1,1,0,1,1,1,1,0,0],\\
&[0,1,1,1,1,0,1,1,1,1,1,0], [0,1,1,1,1,1,0,1,1,1,1,1], [0,1,1,1,1,1,1,0,1,1,1,1], [0,1,1,1,1,1,1,1,0,0,1,1],\\
&[0,1,1,1,1,1,1,1,0,1,1,1], [0,1,1,1,1,1,1,1,1,0,1,1], [0,1,1,1,1,1,1,1,1,1,0,1], [0,1,1,1,1,1,1,1,1,1,1,0],\\
&[1,0,0,0,0,0,1,0,0,0,0,0], [1,0,0,0,0,1,1,0,0,0,0,1], [1,0,0,0,1,0,1,0,0,0,1,0], [1,0,0,0,1,1,1,0,0,0,1,1],\\
&[1,0,0,1,0,0,1,0,0,1,0,0], [1,0,0,1,0,1,1,0,0,1,0,1], [1,0,0,1,1,0,1,0,0,1,1,0], [1,0,0,1,1,1,1,0,0,1,1,1],\\
&[1,0,1,0,0,0,1,0,1,0,0,0], [1,0,1,0,0,1,1,0,1,0,0,1], [1,0,1,0,1,0,1,0,1,0,1,0], [1,0,1,0,1,1,1,0,1,0,1,1],\\
&[1,0,1,1,0,0,1,0,1,1,0,0], [1,0,1,1,0,1,1,0,1,1,0,1], [1,0,1,1,1,0,1,0,1,1,1,0], [1,0,1,1,1,1,1,0,1,1,1,1],\\
&[1,1,0,0,0,0,1,1,0,0,0,0], [1,1,0,0,0,1,1,1,0,0,0,1], [1,1,0,0,1,0,1,1,0,0,1,0], [1,1,0,0,1,1,1,1,0,0,1,1],\\
&[1,1,0,1,0,0,1,1,0,1,0,0], [1,1,0,1,0,1,1,1,0,1,0,1], [1,1,0,1,1,0,1,1,0,1,1,0], [1,1,0,1,1,1,1,1,0,1,1,1],\\
&[1,1,1,0,0,0,1,1,1,0,0,0], [1,1,1,0,0,1,1,1,1,0,0,1], [1,1,1,0,1,0,1,1,1,0,1,0], [1,1,1,0,1,1,1,1,1,0,1,1],\\
&[1,1,1,1,0,0,1,1,1,1,0,0], [1,1,1,1,0,1,1,1,1,1,0,1], [1,1,1,1,1,0,1,1,1,1,1,0], [1,1,1,1,1,1,1,1,1,1,1,1]\}
\end{aligned}
$$

Figure 5.4: Set **V**

feedback polynomial of $\mathcal{P}_n$. To make the concepts easy to understand, we will consider the permutation $\mathcal{P}$ taking as input the state $\mathbf{s}$ and key $\mathbf{k}$ and output an updated state $\mathbf{s}'$ but the same key $\mathbf{k}$. Observe that the keyed permutation $\mathcal{P}_n$ can be written as

$$\mathcal{P}_n = \underbrace{\mathcal{P} \circ \mathcal{P} \circ \cdots \circ \mathcal{P}}_{n\ times}$$

and the feedback polynomial at the n^{th} round is $\pi_{\boldsymbol{e}_{128}}(\mathcal{P}_n(\mathbf{s}, \mathbf{k}))$ where

$$\boldsymbol{e}_{128} = [0, 0, \ldots, 0, \underset{128^{th}\ index}{1}, 0, \ldots, 0, 0]$$

From the definition of monomial trail, a trail exists from $\pi_{\mathbf{u}^{(0)}}(\mathbf{s})$ to $\pi_{\boldsymbol{e}_{128}}(\mathcal{P}_n(\mathbf{s}, \mathbf{k}))$

$$\begin{aligned}
v'_0 + v'_1 + v'_3 + v'_4 + v'_5 &>= u'_0 + u'_1 + u'_3 + u'_4 + u'_5 \\
u'_1 &>= v'_1 \\
u'_3 &>= v'_3 \\
u'_4 &>= v'_4 \\
u'_5 &>= v'_5 \\
u'_2 &>= v'_2 \\
v'_0 + v'_1 + v'_2 + v'_4 + v'_5 &>= u'_0 + u'_1 + u'_2 + u'_4 + u'_5 \\
u'_3 + v'_2 &>= u'_2 \\
u'_2 + v'_3 &>= u'_3 \\
u'_0 + u'_3 + u'_5 + 2 &>= v'_0 + v'_1 + v'_4 \\
u'_0 + u'_2 + u'_4 + 2 &>= v'_0 + v'_1 + v'_5 \\
u'_0 + u'_1 + u'_2 + u'_4 + u'_5 &>= v'_0 \\
u'_0 + u'_1 + u'_3 + 2 &>= v'_0 + v'_4 + v'_5 \\
u'_0 + u'_4 + u'_5 + 3 &>= v'_0 + v'_1 + v'_2 + v'_3 \\
u'_0 + u'_3 + u'_4 + 2 &>= v'_0 + v'_1 + v'_5 \\
u'_0 + u'_1 + u'_5 + 3 &>= v'_0 + v'_2 + v'_3 + v'_4 \\
u'_0 + u'_1 + u'_4 + 3 &>= v'_0 + v'_2 + v'_3 + v'_5 \\
u'_0 + u'_2 + u'_5 + 2 &>= v'_0 + v'_1 + v'_4 \\
u'_0 + u'_1 + u'_3 + u'_4 + u'_5 &>= v'_0 \\
u'_0 + u'_1 + u'_2 + 2 &>= v'_0 + v'_4 + v'_5 \\
u'_0 + 5 &>= v'_0 + v'_1 + v'_2 + v'_3 + v'_4 + v'_5
\end{aligned}$$

Figure 5.5: Inequalities for $\mathcal{F}$

(denoted as $\pi_{\mathbf{u}^{(0)}}(\mathbf{s}) \rightsquigarrow \pi_{\boldsymbol{e}_{128}}(\mathcal{P}_n(\mathbf{s},\mathbf{k}))$) if there is a set of binary vectors $\mathbf{u}^{(1)}, \mathbf{u}^{(2)}, \ldots,$ $\mathbf{u}^{(n-1)}$ such that

$$\pi_{\mathbf{u}^{(i-1)}}(\mathcal{P}_{i-1}(\mathbf{s},\mathbf{k})) \rightarrow \pi_{\mathbf{u}^{(i)}}(\mathcal{P}_i(\mathbf{s},\mathbf{k})), \forall i \in [128]$$

where $\mathbf{u}^{(128)} = \boldsymbol{e}_{128}$ and $\mathbf{s}$ is the initial state such that $s_{96+i} = x_i, \forall i \in \{0, 1, \ldots, 32\}$ and the rest are 0. Therefore, we only need to find out linear constraints consisting of $\mathbf{u}^{i-1}$ and $\mathbf{u}^{i}$ such that the constraints are satisfied if and only if $\pi_{\mathbf{u}^{(i-1)}}(\mathcal{P}_{i-1}(\mathbf{s},\mathbf{k})) \rightarrow \pi_{\mathbf{u}^{(i)}}(\mathcal{P}_i(\mathbf{s},\mathbf{k}))$. For the sake of simplicity, let us consider the notation $(\boldsymbol{y},\mathbf{k}) :=$ $\pi_{\mathbf{u}^{(i-1)}}(\mathcal{P}_{i-1}(\mathbf{s},\mathbf{k}))$, then $(\boldsymbol{z},\mathbf{k}) := \pi_{\mathbf{u}^{(i)}}(\mathcal{P}_i(\mathbf{s},\mathbf{k})) = \mathcal{P}(\boldsymbol{y},\mathbf{k})$. Also, we will use $\mathbf{u} :=$ $\mathbf{u}_{(i-1)}$ and $\mathbf{v} := \mathbf{u}^{(i)}$. So, we need to find linear constraints $\mathbf{u}$ and $\mathbf{v}$ such that the constraints are satisfied if and only if $\pi_{\mathbf{u}}(\boldsymbol{y}, \mathbf{k}) \rightarrow \pi_{\mathbf{v}}(\boldsymbol{z}, \mathbf{k})$.

Observe that from the description of $\mathcal{P}$, we have $z_i = y_{i+1}, \forall i \in \{0, 1, \ldots, 126\}$ and z_{127} depends on five entries of y and a single key entry. In other words, only one entry in z is different from y and it depends only on a few entries of the input. Let us capture this in a function $\mathcal{F}$ which takes as input $(y_0, y_{47}, y_{70}, y_{85}, y_{91}, k_i)$ and outputs $(z_{127}, z_{46}, z_{69}, z_{84}, z_{90}, k_i)$ such that

$$z_{i+1} = y_i, \forall i \in \{47, 70, 85, 91\}$$

$$z_{127} = y_0 + y_{47} + y_{70} \cdot y_{85} + y_{91} + k_i$$

Apart from $(z_{127}, z_{46}, z_{69}, z_{84}, z_{90}, k_i)$, the rest of the z_j's and k_j's are equal to some y_j's or k_j's or they are not involved in the function $\mathcal{F}$. Therefore, in the MILP model, we can set those z_j's and k_j's to their corresponding y_j's or k_j's. We only need to figure out constraints on $\mathbf{u}'$ and $\mathbf{v}'$ such that $\prod_{\mathbf{u}'}(y_0, y_{47}, y_{70}, y_{85}, y_{91}, k_i) \rightarrow \prod_{\mathbf{v}'}(z_{127}, z_{46}, z_{69}, z_{84}, z_{90}, k_i)$.

We begin by enumerating all binary vectors of dimension 6 into a set $\mathbf{V}$ (see Figure 5.4) such that

$$[\mathbf{u}', \mathbf{v}'] \in \mathbf{V} \iff \pi_{\mathbf{u}'}(y_0, y_{47}, y_{70}, y_{85}, y_{91}, k_i) \rightarrow \pi_{\mathbf{v}'}(z_{127}, z_{46}, z_{69}, z_{84}, z_{90}, k_i)$$

Using the set $\mathbf{V}$, we can generate the linear constraints that define the convex hull of $\mathbf{V}$. This can be achieved by using the *inequality_generator()* function in SAGE [Sag17]. Observe that the convex hull generated does not contain any spurious points because the points in $\mathbf{V}$ are on the surface of a 12 dimensional unit cube. Therefore, the convex hull is bounded by the unit cube unless the hull is unbounded, which is not the case. Since the interior of the unit cube does not contain any integer points, the convex hull won't contain any spurious points.

Since the number of inequalities affects the running time of the MILP solver, one needs to make sure they work with a minimal number of constraints. For this, we go through the inequalities generated through SAGE and check one by one whether any inequality can be removed. By doing so, we arrive at a system of 27 linear inequalities that describes $\pi_{\mathbf{u}'}(y_0, y_{47}, y_{70}, y_{85}, y_{91}, k_i) \to \pi_{\mathbf{v}'}(z_{127}, z_{46}, z_{69}, z_{84}, z_{90}, k_i)$ as shown in Figure 5.5.

Now we have the required inequalities (which we will denote an $\mathcal{L}$), we will describe the MILP model for finding the largest monomial that has a trail to the feedback polynomial of $\mathcal{P}_n$. The objective function is to maximize $\sum_{i=96}^{127} u_i^{(0)}$. This is because the nonce variables are present at those indexes. We must also set $u_i^{(0)} == 0, \forall i < 96$ as we are not interested in monomials containing s_i where $i < 96$. Using the set $\mathcal{L}$, we will create linear inequalities for $\mathbf{u}^{(i-1)}$ and $\mathbf{u}^{(i)}$ which will ensure that $\pi_{\mathbf{u}^{(i-1)}}(\mathcal{P}_{i-1}(s,k)) \to \pi_{\mathbf{u}^{(i)}}(\mathcal{P}_i(s,k))$. Finally, since we are interested in the feedback polynomial of $\mathcal{P}_n$, we will set $u_{127}^{(n)} == 1$ and the rest to 0.

Solving the above model using Gurobi[1], will only give us information about the largest term that has a monomial trail. To conclude that the term is also a monomial in the feedback polynomial, we need to count the number of monomial trails. Observe that an optimal solution in the above model is a valid monomial trail, i.e., $\mathbf{u}^{(i)}$'s gives us all the information of the trail. Therefore, it is enough to count the number of solutions in the model by only fixing $\mathbf{u}^{(0)}$ to the one returned by optimal. The number of solutions can be obtained by using the *PoolSearchMode* of Gurobi which searches for all the solutions. One can see from Figure 5.6 that the degree of the feedback polynomial after 381 rounds is 32.

5.4 Key-Recovery Attacks

In this section, we will present key-recovery attacks against TinyJAMBU. Like other key-recovery attacks, our attacks consist of two phases: offline and online phases. In the offline phase, the attacker finds polynomials consisting of key variables by using methods such as cube attacks, MILP, etc. Once the attacker is equipped with these polynomials, in the offline phase, he will try to obtain the evaluations of these

[1]https://www.gurobi.com

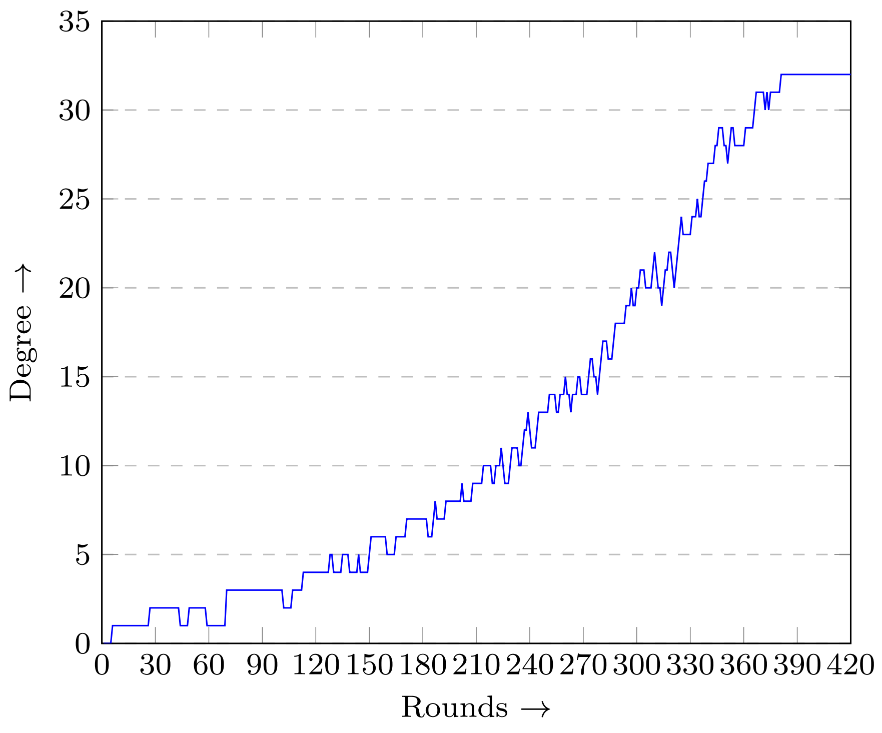

Figure 5.6: Degree of feedback polynomial

polynomials and recover certain key bits.

5.4.1 Offline Phase: Finding the Polynomials

In the offline phase of our attack, we will use MILP to extract the polynomials with key variables. Let $\mathcal{C}_{\mathbf{I}}$ be a cube with index set $\mathbf{I}$ and the output polynomial written with respect to this cube be

$$f(\boldsymbol{x}, \mathbf{k}) = \mathbf{t} \cdot p_{\mathbf{I}}(\boldsymbol{x}, \mathbf{k}) + q_{\mathbf{I}}(\boldsymbol{x}, \mathbf{k})$$

where $\mathbf{t} = \prod_{i \in \mathbf{I}} x_i$. Let $\mathbf{A} = \{a_i \mid i \notin \mathbf{I}\}$ be the set of constants for the non-cube variables. Our objective is to find the superpoly $p_{\mathbf{I}}(\mathbf{A}, \mathbf{k})$ which will be used in the online phase. We will again use the concept of monomial trails to find the superpolies. This technique was also used in [Hu+20] to find the superpolies for Trivium.

In our attack, we will consider $\mathbf{A}$ to be all zeroes. Recall that in Section 5.3, we presented a MILP model that would find the degree of the largest monomial containing nonce variables only that has a monomial trail to the feedback polynomial. This is achieved by setting up a system of linear inequalities with the objective function being "maximise $\sum_{i=96}^{127} u_i^{(0)}$ ". Then, by counting the number of monomial trails, we can confirm whether it is monomial in the feedback polynomial or not. We will simply modify the above MILP model that will help us find every monomial and build the entire superpoly.

We first begin by modifying the MILP model. The objective function must be changed to maximizing $\sum_{i=128}^{255} u_i^{(0)}$. Recall that we considered the input to $\mathcal{P}$ as $(\mathbf{s}, \mathbf{k})$ where $\mathbf{s}$ is the state and $\mathbf{k}$ is the key. Therefore, the objective function is looking for the largest term consisting of key variables. Since, we are interested in the superpoly with respect to some $\mathbf{t}$, we need to add the following equalities as constraints also.

$$\begin{aligned} &u_i^{(0)} == 1, \forall i \in \mathbf{I} \\ &u_i^{(0)} == 0, \forall i \in \{0, 1, \dots, 127\} \setminus \mathbf{I} \end{aligned}$$

The rest of the inequalities are exactly the same as the ones used in Section 5.3. Observe that a solution to the above MILP model gives us a monomial $\boldsymbol{x}_{\mathbf{I}}\mathbf{k}_{\mathbf{J}}$ for some index set $\mathbf{J}$ such that $|\mathbf{J}|$ is the largest and $\boldsymbol{x}_{\mathbf{I}}\mathbf{k}_{\mathbf{J}}$ has a monomial trail to $\mathcal{P}_n(\mathbf{s}, \mathbf{k})^{(127)}$. To confirm that this is a monomial, we must again count the number of solutions and store it if it is a monomial. Now, to find the next monomial, we must add a new constraint so that the MILP solver won't return the same solution again. To be precise, suppose the MILP solver returns a solution $\mathbf{u}^{(0)}$. For the sake of simplicity, let $\mathbf{J} = \{i \mid u_i^0 = 1, i \geq 128\}$. Then, we add the constraint,

$$\sum_{i \in \mathbf{J}} (1 - \mathbf{u}_i^{(0)}) >= 0$$

This constraint ensures that the MILP solver won't return the same solution twice.

In Table 5.4, we present a few superpolies of the feedback polynomial with respect to cubes of size 31 for $\mathcal{P}_{376}$ by solving the MILP model mentioned above.

Non-cube indices	Superpoly
4	$k_{13} + k_{14} + k_{20}$
5	$k_{14}k_{28} + k_{14}k_{22} + k_7k_{14} + k_{14}k_{20} + k_{14}k_{65} + k_2k_{14} + k_{13}k_{14} + k_2k_{22} + k_{14}k_{42} + k_2k_{72} + k_2k_6 + k_7k_{20} + k_{44}k_{57} + k_2k_7 + k_1k_{57} + k_1k_{13} + k_{20}k_{22} + k_2k_{87} + k_2k_{50} + k_{50}k_{72} + k_{20}k_{28} + k_{20}k_{65} + k_{13}k_{44} + k_{13}k_{65} + k_{13}k_{87} + k_{14}k_{45} + k_{35}k_{72} + k_1k_{45} + k_2k_{28} + k_2k_{65} + k_{50}k_{87} + k_{13}k_{72} + k_6k_{72} + k_{57}k_{65} + k_{14}k_{34} + k_{44}k_{45} + k_{35}k_{87} + k_6k_{87} + k_{35}k_{45} + k_{45}k_{65} + k_2 + k_{87} + k_{72} + k_{20} + k_{78} + k_{34} + k_{42} + k_{65} + k_{57} + k_{71}$
6	$k_4k_7k_{14} + k_1k_{22} + k_{22}k_{44} + k_2k_{14} + k_4k_7 + k_2k_{44} + k_7k_{22} + k_1k_2 + k_7k_{22} + k_2k_{44} + k_1k_{22} + k_{22}k_{44} + k_4k_7 + k_1 + k_{44} + k_{22} + k_{28}$
14	$k_7k_{14} + k_7$
13	$k_4k_7k_{14} + k_4k_{14} + k_4k_7 + k_7k_{22} + k_1k_{22} + k_{22}k_4 + k_{14} + k_{22} + k_4 + k_{28} + k_{65}$
16	$k_2k_7k_{14} + k_7k_{12}k_{14} + k_7k_{14} + k_6k_{14} + k_{14}k_{50} + k_7k_{35} + k_{14} + k_{50} + k_7 + k_6 + k_{35}$
17	$k_4k_{14} + k_4 + 1$
20	$k_1 + k_{14} + k_{44} + k_{65}$
28	$k_{14} + k_{72} + k_{56} + k_{12} + k_{35} + k_{50} + k_{87} + k_{34} + k_{19} + k_{93}$
29	$k_1 + k_{14} + k_{44}$
31	$k_1 + k_2 + k_{44} + k_{45}$

Table 5.4: Superpolies for $\mathcal{P}_{376}$

5.4.2 Online Phase: Recovering the Key Bits

In the online phase, the attacker is given a blackbox access to TinyJAMBU, i.e., the attacker is allowed to choose the value of 32-bit nonce and is given the bits at position $64, 65, \ldots, 95$ of the state updated by $\mathcal{P}_n$. Observe that there are 5 linearly independent polynomials in Table 5.4. This implies that the attacker can get the evaluations of these superpolies by performing cube tester for 440 round TinyJAMBU. Therefore, the attacker can recover the entire key in time $2^{123} + 5 \cdot 2^{31}$.

Chapter 6

New Zero-Sum Distinguishers for Ascon

In this chapter, we present several distinguishers for 5-round Ascon-128. In [Roh+21], the authors found a distinguisher with complexity 2^{16} for 5-round Ascon by identifying several cubes based on division property.

We present a zero-sum distinguisher with 2^{15}-time complexity for 5-round Ascon-128 by experimentally finding a degree-15 monomial that does not exists in the polynomial representing one of the output bits. We also present several 13 and 14 sized cubes that can be used as a distinguisher for 5-round Ascon with very high confidence.

6.1 Structure of Ascon

Dobraunig et al. designed Ascon. It is a permutation-based family of authenticated encryption with associated data algorithms (AEAD). The Ascon AEAD algorithm takes as inputs a secret key K, a nonce N, a block header AD (a.k.a associated data) and a message M. It then outputs a ciphertext C of the same length as M and an authentication tag T, which authenticates the associated data AD and the message M. There are two variants of Ascon, namely Ascon-128 and Ascon-128a.

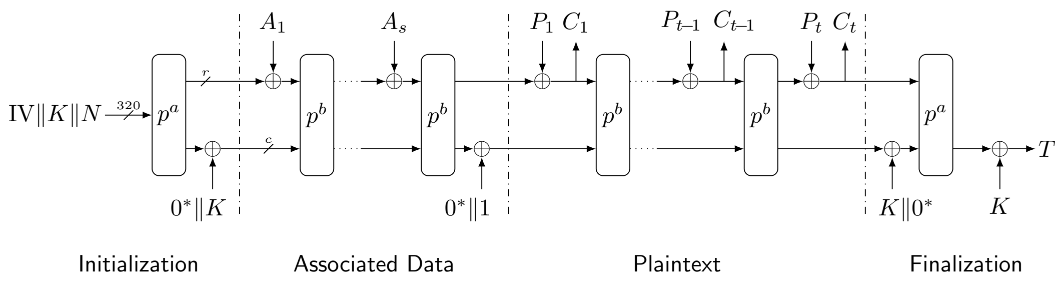

Figure 6.1: ASCON's mode of operation (encryption phase)

Table 6.1: ASCON variants and their recommended parameters

Name	State size	Rate r	Size of			Rounds		IV
			Key	Nonce	Tag	p^a	p^b	
ASCON-128	320	64	128	128	128	12	6	`80400c0600000000`
ASCON-128a	320	128	128	128	128	12	8	`80800c0800000000`

6.1.1 The ASCON Permutation

The core permutation p of ASCON is based on substitution permutation network (SPN) design paradigm. It operates on a 320-bit state arranged into five 64-bit words and is defined as $p : p_L \circ p_S \circ p_C$. The state at the input of r-th round is denoted by $X_0^r\|X_1^r\|X_2^r\|X_3^r\|X_4^r$ while $Y_0^r\|Y_1^r\|Y_2^r\|Y_3^r\|Y_4^r$ represents the state after the p_S layer. We use $X_i^r[j]$ (resp. $Y_i^r[j]$) to denote the j-th bit (starting from left) of X_i^r (resp. Y_i^r). We now describe the three steps p_C, p_S, and p_L in detail (superscripts are removed for simplicity).

Addition of constants (p_C). We add an 8-bit constant to the bits $56, \cdots, 63$ of word X_2 at each round.

Substitution layer (p_S). We apply a 5-bit Sbox on each of the 64 columns. Let $(x_0, x_1, x_2, x_3, x_4)$ and $(y_0, y_1, y_2, y_3, y_4)$ denote the input and output of the Sbox, respectively. Then the algebraic normal form (ANF) of the Sbox is given in Equation (6.1). Note that here x_i and y_i are the bits of word X_i and Y_i, respectively.

$$\begin{cases} y_0 = x_4x_1 + x_3 + x_2x_1 + x_2 + x_1x_0 + x_1 + x_0 \\ y_1 = x_4 + x_3x_2 + x_3x_1 + x_3 + x_2x_1 + x_2 + x_1 + x_0 \\ y_2 = x_4x_3 + x_4 + x_2 + x_1 + 1 \\ y_3 = x_4x_0 + x_4 + x_3x_0 + x_3 + x_2 + x_1 + x_0 \\ y_4 = x_4x_1 + x_4 + x_3 + x_1x_0 + x_1 \end{cases} \tag{6.1}$$

Linear diffusion layer (p_L). Each 64-bit word is updated by a linear operation Σ_i which is defined in Equation (6.2). Here $\ggg$ is the right cyclic shift operation over a 64-bit word.

$$\begin{cases} X_0 \leftarrow \Sigma_0(Y_0) = Y_0 + (Y_0 \ggg 19) + (Y_0 \ggg 28) \\ X_1 \leftarrow \Sigma_1(Y_1) = Y_1 + (Y_1 \ggg 61) + (Y_1 \ggg 39) \\ X_2 \leftarrow \Sigma_2(Y_2) = Y_2 + (Y_2 \ggg 1) + (Y_2 \ggg 6) \\ X_3 \leftarrow \Sigma_3(Y_3) = Y_3 + (Y_3 \ggg 10) + (Y_3 \ggg 17) \\ X_4 \leftarrow \Sigma_4(Y_4) = Y_4 + (Y_4 \ggg 7) + (Y_4 \ggg 41) \end{cases} \tag{6.2}$$

6.2 New Distinguishers for 5 Rounds Ascon-128

As shown in Figure 6.1, X_0^0 is set to IV whereas, X_1^0, X_2^0 are set to key bits (secret bits) and X_3^0, X_4^0 to nonce bits (public bits). Since the ciphertext C_1 is obtained by XOR-ing the message P_1 to X_0^5, we will consider zero-sum distinguisher at X_0^5 positions only.

In [Roh+21], the authors found a distinguisher with complexity 2^{16} for 5-round Ascon by setting $X_3^0 = X_4^0$ and finding an upper bound on the algebraic degree in nonce variables using division property. In our experiments, we also set $X_3^0 = X_4^0$.

The road-map. We first show that there is a degree-15 monomial that *is not present* in $X_0^5[1]$, and thus, we find a 15 sized cube as a zero-sum distinguisher. Observe that it is impossible to construct the exact polynomials of X_0^5 with respect to the key and cube variables because the number of key variables is 128, and this naturally results in a huge number of monomials. We will use $\mathbf{k}$ to denote the key variables and $\boldsymbol{x}$ to denote the public/cube variables.

From [Roh+21], we know that the degree of $X_0^5[1]$ with respect to the cube variables is 15. Instead of finding the *exact* polynomial in $X_0^5[1]$, we will come up with an *approximate polynomial* (which in some sense is related to the exact polynomial, see Property 6.1) that contains the cube variables only. Since the approximate polynomial contains only cube variables, it has very few monomials compared to the exact polynomial and this allows us to compute it. The approximate polynomial has the following property.

Property 6.1. *If the exact polynomial has a monomial $p \cdot q$ where $p \in \mathbb{F}_2[\boldsymbol{x}]$ and $q \in \mathbb{F}_2[\mathbf{k}]$, then the approximate polynomial will contain p.*

For example, suppose the exact polynomial is $f(v_1, v_2, x_1, x_2, x_3) = x_1x_2x_3 + x_2x_3 + x_1 + v_1v_2 + x_2v_1 + v_2$, then the approprimate polynomial $\tilde{f}(v_1, v_2) = v_1v_2 + v_1 + v_2$ has Property 6.1.

In other words, if a monomial p is missing from the approximate polynomial, then it is guaranteed that $p \cdot q$ for any $q \in \mathbb{F}_2[\mathbf{k}]$ would not be present in the exact output polynomial. But, this does not say anything about the presence or absence of $p \cdot q'$ in the exact polynomial where $q' \in \mathbb{F}_2[\boldsymbol{x}]$. We do not have to worry about such terms because $p \cdot q'$ can be ignored by setting the appropriate nonce variables to 0, to satisfy $q' = 0$.

To build these approximate polynomials, we will work over the ring of integers $\mathbb{Z}$, *rather than* $\mathbb{F}_2$. The reason for this choice is that in SAGE [Sag17], the computations over $\mathbb{F}_2$ polynomials are time consuming compared to computations over $\mathbb{Z}$ polynomial, especially when the degrees are large. We start by building the exact polynomial over $\mathbb{Z}$ up to 2 rounds, i.e., we will consider both key and cube variables and replace XOR with integer $+$ and $\cdot$ with integer $\times$. This will give rise to two issues.

1. Firstly, the coefficients of the monomials will blow up. This can be handled

Rounds	Cube size	Cube indices ($X_3^0 = X_4^0$)	Output indices (X_0^5)
5	13	0, 1, 2, 3, 4, 5, 7, 8, 10, 11, 12, 13, 16	4
		0, 1, 2, 3, 5, 6, 7, 8, 10, 11, 12, 13, 16	4
		0, 1, 2, 4, 5, 6, 7, 8, 10, 11, 12, 13, 16	4
5	14	0, 1, 2, 3, 4, 5, 6, 7, 8, 10, 11, 12, 13, 14	1, 4
		0, 1, 2, 3, 4, 5, 6, 7, 8, 10, 11, 12, 13, 16	4, 15, 24, 36
		0, 1, 2, 3, 4, 5, 6, 7, 8, 10, 11, 12, 13, 18	4

Table 6.2: List of cubes for 5-round ASCON-128

by replacing all non-zero coefficients with 1.

2. Secondly, the monomials are no longer multi-linear. This also can be fixed by reducing the polynomial by modulo $x_i^2 - x_i, \forall i$.

Observe that as an alternative, we can simply work over $\mathbb{F}_2$, find the exact polynomials and then convert them into polynomials over $\mathbb{Z}$. But, this approach seems to be time-consuming in SAGE.

Next, we get rid of the key variables by evaluating the polynomial at $k_i = 1, \forall i$. Again, the coefficients may blow up which is handled by replacing all non-zero coefficients with 1. Observe that these new polynomials have Property 6.1. We will apply the rest of the 3 rounds on these approximate polynomials to get the approximate polynomials for $X_0^5[1]$. In our experiment, while considering cube indices $0, 1, \ldots, 14$, the approximate polynomial for $X_0^5[1]$ *does not* contain the monomial $\prod_{i=0}^{14} x_i$. This gives us a zero-sum distinguisher for 5 rounds with complexity 2^{15}. Observe that this experiment is essentially solving Problem 1.1 for 5-round ASCON with the monomial m being $\prod_{i=0}^{14} x_i$. The source code is available at: `https://github.com/Mahe94/ascon_monomial_detection.git`.

In Table 6.2, we provide more cubes which can serve as a distinguisher for 5-round ASCON-128. We start by randomly guessing a few cubes (i.e., a subset of bit indices in X_3^0 and X_4^0) of size ℓ that is *strictly smaller* than 16, and set the rest of the bits of the nonce (non-cube bits) to 0. For each $u \in \{0,1\}^\ell$, we set the cube variables to u and run 5-round ASCON-128. The output X_0^5 with respect to each

u is summed up to get the cube sum. We analyse the cube sum at X_0^5 bits for 2^{15} randomly generated keys and observe the following:

1. For all the 14 sized cubes, the sum at the output mentioned in Table 6.2 was 0 for every key.

2. For 13 sized cubes, the sum was 0 with *high probability*.

All indices mentioned in Table 6.2 have an offset of 0. Since, the 14 sized cubes are giving 0 as the cube-sum for all 2^{15} randomly chosen keys, we can use them as a distinguisher for 5-round Ascon with very high confidence, owing a complexity of 2^{14} and beating the best-known cube distinguisher [Roh+21], by a factor of 2^2.

Chapter 7

Efficient Reductions and Algorithms for Variants of Subset Sum

7.1 Hardness Results

In this section, we prove some hardness results.

Some of the algorithms presented in this chapter consider that the number of solutions is *bounded* by a parameter k. This naturally raises the question of whether the SSUM problem is hard when the number of solutions is bounded. We will show that this is true even for the case when $k = 1$, i.e., uSSSUM is NP-hard under *randomised* reduction.

Theorem 7.1 (Hardness of uSSSUM). *There exists a randomized reduction which takes a* SSUM *instance* $\mathcal{M} = (a_1, \ldots, a_n, t) \in \mathbb{Z}_{\geq 0}^{n+1}$*, as an input, and produces multiple* SSUM *instances* $\mathcal{SS}_\ell = (b_1, \ldots, b_n, t^{(\ell)})$*, where* $\ell \in [2n^2]$*, such that if*

- $\mathcal{M}$ *is a YES instance of* SSUM $\implies \exists \ell$ *such that* $\mathcal{SS}_\ell$ *is a YES instance of* uSSSUM.
- $\mathcal{M}$ *is a NO instance of* SSUM $\implies \forall \ell, \mathcal{SS}_\ell$ *is a NO instance of* uSSSUM.

Proof. The core of the proof is based on the Lemma 2.1 (Isolation lemma). The reduction is as follows. Let $w_1, \ldots, w_n$ be chosen *uniformly at random* from $[2n]$. We define $b_i = 4n^2 a_i + w_i, \forall i \in [n]$ and the ℓ^{th} SSUM instance as $\mathcal{SS}_\ell = (b_1, \ldots, b_n, t^{(\ell)} = 4n^2 t + \ell)$. Observe that all the new instances are different only in the target values $t^{(\ell)}$.

Suppose $\mathcal{M}$ is a YES instance, i.e., $\exists S \subseteq [n]$ such that $\sum_{i \in S} a_i = t$. Then, for $\ell = \sum_{i \in S} w_i$, the $\mathcal{SS}_\ell$ is a YES instance, because

$$\sum_{i \in S} b_i - t^{(\ell)} \;=\; 4n^2 \left(\sum_{i \in S} a_i - t \right) \;-\; \left(\ell - \sum_{i \in S} w_i \right) \;=\; 0\ .$$

If $\mathcal{M}$ is a NO instance, consider any ℓ and $S \subseteq [n]$. Since $\mathcal{M}$ is a NO instance, $4n^2(\sum_{i \in S} a_i - t)$ is a non-zero multiple of $4n^2$, whereas $|\ell - \sum_{i \in S} w_i| < 4n^2$, which implies that

$$4n^2(\sum_{i \in S} a_i - t) - (\ell - \sum_{i \in S} w_i) \neq 0 \implies \sum_{i \in S} b_i \neq t^{(\ell)}\ .$$

Hence, $\mathcal{SS}_\ell$ is also a NO instance.

We now show that if $\mathcal{M}$ is a YES instance, then one of $\mathcal{SS}_\ell$ is a uSSSUM. Let $\mathcal{F}$ contains all the solutions to the SSUM instance $\mathcal{M}$, i.e., $\mathcal{F} = \{S | S \subseteq [n], \sum_{i \in S} a_i = t\}$. Since w_i's are chosen uniformly at random, Lemma 2.1 says that there exists a *unique* $S \in \mathcal{F}$, such that $w(S) = \sum_{i \in S} w_i$, is *minimal* with probability at least $1/2$. Let us denote this minimal value $w(S)$ as ℓ^*. Then, $\mathcal{SS}_{\ell^*}$ is uSSSUM because S is the only subset such that $\sum_{i \in S} w_i = \ell^*$. □

Next, we present a simple deterministic Cook's reduction from SSUM to $2 - \mathsf{SimulSubsetSum}$. It is obvious to see that $2 - \mathsf{SimulSubsetSum} \in \mathsf{NP}$ which implies that $2 - \mathsf{SimulSubsetSum}$ is NP-complete.

Theorem 7.2 (Hardness of $2 - \mathsf{SimulSubsetSum}$)**.** *There is a deterministic polynomial time reduction from* SSUM *to* $2 - \mathsf{SimulSubsetSum}$.

Proof. Let $(a_1, \ldots, a_n, t)$ be an instance of SSUM. Consider the following $2 - \mathsf{SimulSubsetSum}$ instances, $\mathcal{S}_b = [(a_1, \ldots, a_n, t), (1, 0, \ldots, 0, b)]$, where $b \in \{0, 1\}$.

If the SSUM instance is NO, then *both* the $2-\mathsf{SimulSubsetSum}$ are also NO. If the SSUM instance is a YES, then we argue that one of the S_b instances must be YES. If SSUM instance has a solution which contains a_1, then $\mathcal{S}_1$ is a YES instance, whereas if it does not contain a_1, then S_0 is a YES instance. □

Extension to $\log -\mathsf{SimulSubsetSum}$**.** The above reduction can be trivially extended to reduce SSUM to SimulSubsetSum, with the number of SSUM instances $k = O(\log n)$. In that case we will work with instances $S_{\boldsymbol{b}}$, for $\boldsymbol{b} \in \{0,1\}^k$. Since the number of instances is $2^k = \mathsf{poly}(n)$, the reduction goes through.

We will now show that $2-\mathsf{SimulSubsetSum}$ reduces to SSUM, which again can be generalised to SimulSubsetSum, for any number of SSUM instances k.

Theorem 7.3 ($2-\mathsf{SimulSubsetSum}$ is *easier* than SSUM)**.** *There is a deterministic polynomial time reduction from* $2-\mathsf{SimulSubsetSum}$ *to* SSUM*.*

Proof. Let $[(a_1,\ldots,a_n,t_1),(b_1,\ldots,b_n,t_2)]$ be a $2-\mathsf{SimulSubsetSum}$ instance where without loss of generality $t_1 \le t_2$. Also, we can assume that $t_1 \le \sum_{i=1}^n a_i$, otherwise it does not have a solution.

Now, consider the SSUM instance $(\gamma b_1 + a_1,\ldots,\gamma b_n + a_n, \gamma t_2 + t_1)$, where $\gamma := 1 + \sum_{i=1}^n a_i$. If the $2-\mathsf{SimulSubsetSum}$ instance is YES, this implies that there exist $S \subseteq [n]$ such that $\sum_{i\in S} a_i = t_1$ and $\sum_{i\in S} b_i = t_2$. This implies that $\sum_{i\in S} \gamma b_i + a_i = \gamma t_2 + t_1$ and hence the SSUM instance is also YES.

Now, assume that the SSUM instance is YES, i.e., there exists $S \subseteq [n]$ such that $\sum_{i\in S} \gamma b_i + a_i = \gamma t_2 + t_1$. This implies that $\gamma(t_2 - \sum_{i\in S} b_i) + (t_1 - \sum_{i\in S} a_i) = 0$. If $t_1 \neq \sum_{i\in S} a_i$, then from the previous equality, $(t_1 - \sum_{i\in S} a_i)$ is a non-zero multiple of $\gamma \implies |\, t_1 - \sum_{i\in S} a_i \,| \ge \gamma$. However, by our assumption,

$$t_1 \;\le\; \sum_{i=1}^n a_i \;\implies\; t_1 - \sum_{i\in S} a_i \;\le\; \sum_{i\in[n]\setminus S} a_i \;<\; 1 + \sum_{i\in[n]} a_i \;=\; \gamma\,.$$

Moreover, $t_1 - \sum_{i\in S} a_i > -\gamma$, holds trivially, since $\gamma > \sum_{i\in[n]} a_i$ and $t_1 > 0$. Therefore, $|\, t_1 - \sum_{i\in S} a_i \,| < \gamma$ which implies that *both* $t_1 - \sum_{i\in S} a_i = 0$ and $t_2 - \sum_{i\in S} b_i = 0$. Hence, the $2-\mathsf{SimulSubsetSum}$ instance is also YES. □

7.2 Time-Efficient Algorithms

7.2.1 Time-Efficient Algorithm for $\mathsf{Hamming}-k-\mathsf{SSSUM}$

In this section, we present an $\tilde{O}(k(n+t))$-time deterministic algorithm for outputting all the hamming weight of the solutions, given a $\mathsf{Hamming}-k-\mathsf{SSSUM}$ instance, i.e., there are only at most k-many solutions to the SSUM instance $(a_1, \ldots, a_n, t) \in \mathbb{Z}_{\geq 0}^{n+1}$. The basic idea is simple: We want to create a polynomial whose roots are of the form μ^{w_i}, so that we can first find the roots μ^{w_i} (over $\mathbb{F}_q$), and from them we can find w_i. To achieve that, we work with k-many polynomials $f_j := \prod_{i=1}^{n}(1+\mu^j \cdot x^{a_i})$, for $j \in [k]$. Note that the coefficient of x^t in f_j is of the form $\sum_{i \leq k} \lambda_i \cdot \mu^{jw_i}$ (Claim 7.5). By Newton's Identities (Lemma 2.3) and Vieta's formulas (Lemma 2.5), we can now *efficiently* construct a polynomial whose roots are μ^{w_i}. The details are given below.

Proof of Theorem 1.9. We start with some notations that we will use throughout the proof.

Basic notations. Assume that the SSUM instance $(a_1, \ldots, a_n, t) \in \mathbb{Z}_{\geq 0}^{n+1}$ has *exactly* m $(m \leq k)$ many solutions, and they have ℓ many *distinct* hamming weights $w_1, \ldots, w_\ell$; since two solutions can have same hamming weight, $\ell \leq m$. Moreover, assume that there are λ_i many solutions which appear with hamming weight w_i, for $i \in [\ell]$. Thus, $\sum_{i \in [\ell]} \lambda_i = m \leq k$.

Choosing prime q and a primitive root μ. We will work with a fixed q in this proof, where $q > n+k+t := M$ (we will mention why such a requirement later). We can find a prime q in $\tilde{O}(n+k+t)$ time since we can go over every element in the interval $[M, 6/5 \cdot M]$, in which we know a prime exists (Theorem 2.9) and primality testing is efficient [AKS04]. Once we find q, we choose μ such that μ is a *primitive root* over $\mathbb{F}_q$, i.e., $\text{ord}_q(\mu) = q-1$. This μ can be found in $\tilde{O}((n+k+t)^{1/4+\epsilon})$ time using Theorem 2.8. Thus, the total time complexity of this step is $\tilde{O}(n+k+t)$.

The polynomials. Define the k-many univariate polynomials as follows:

$$f_j(x) \;:=\; \prod_{i\in[n]} \left(1+\mu^j x^{a_i}\right) \,,\, \forall\, j \,\in\, [k]\,.$$

We remark that we do not know ℓ apriori, but we can find m efficiently.

Claim 7.4 (Finding the exact number of solutions)**.** *Given a* $\mathsf{Hamming}-k-\mathsf{SSSUM}$ *instance, one can find the exact number of solutions, m, deterministically, in $\tilde{O}((n+t))$ time.*

Proof. Use [JW18] (see Lemma A.2, for the general statement) which gives a deterministic algorithm to find the coefficient of x^t of $\prod_{i\in[n]}(1+x^{a_i})$ over $\mathbb{F}_q$; this takes time $\tilde{O}((n+t))$. □

Since we know the exact value of m, we will just work with f_j for $j \in [m]$, which suffices for our algorithmic purpose. Here is an important claim about coefficients of x^t in f_j's.

Claim 7.5. $C_j = \mathrm{coef}_{x^t}(f_j(x)) \;=\; \sum_{i\in[\ell]} \lambda_i \cdot \mu^{jw_i}$*, for each* $j \in [m]$.

Proof. If $S \subseteq [n]$ is a solution to the instance with hamming weight, say w, then this will contribute μ^{jw} to the coefficient of x^t of $f_j(x)$. Since, there are ℓ many weights $w_1, \ldots, w_\ell$ with multiplicity $\lambda_1, \ldots, \lambda_\ell$, the claim easily follows. □

Using Lemma A.2, we can find $C_j \mod q$ for each $j \in [m]$ in $\tilde{O}((n+t\log(\mu j)))$ time, owing total $\tilde{O}(k(n+t))$, since $q = O(n+k+t)$, $\mu \le q-1$, and $\sum_{j\in[m]}\log j = \log(m!) \le \log(k!) = \tilde{O}(k)$.

Using the Newton's Identities (Lemma 2.3), we have the following relations, for $j \in [m]$:

$$E_j\left(\mu^{w_1},\ldots,\mu^{w_\ell}\right) \equiv j^{-1}\cdot\left(\sum_{i=1}^{j}(-1)^{i-1}E_{j-i}\left(\mu^{w_1},\ldots,\mu^{w_k}\right)\cdot P_i\left(\mu^{w_1},\ldots,\mu^{w_\ell}\right)\right) \bmod q \qquad (7.1)$$

In the above, by $E_j(\mu^{w_1}, \ldots, \mu^{w_\ell})$, we mean $E_j(\underbrace{\mu^{w_1}, \ldots, \mu^{w_1}}_{\lambda_1 \text{ times}}, \ldots, \underbrace{\mu^{w_\ell}, \ldots, \mu^{w_\ell}}_{\lambda_\ell \text{ times}})$, and similar for P_j. Since $q > k$, $j^{-1} \bmod q$ exists, and thus the above relations are valid. Here is another important and obvious observation, just from the definition of P_j's:

Observation 7.6. *For $j \in [k]$, $C_j \equiv P_j(\mu^{w_1}, \ldots, \mu^{w_\ell}) \bmod q$.*

Note that we know $E_0 = 1$ and P_j's (and $j^{-1} \bmod q$) are already computed. To compute E_j, we need to know $E_1, \ldots, E_{j-1}$ and additionally we need $O(j)$ many additions and multiplications. Suppose, $T(j)$ is the time to compute $E_1, \ldots, E_j$. Then, the trivial complexity is $T(m) \le \tilde{O}(k^2) + \tilde{O}(k(n+t))$. But one can do better than $\tilde{O}(k^2)$ and make it $\tilde{O}(k)$ (i.e solve the recurrence, using FFT), owing the total complexity to $T(m) \le \tilde{O}(k(n+t))$ (since $q = O(n+k+t)$). For details, see Appendix A.1.3.

Once, we have computed E_j, for $j \in [m]$, define a new polynomial

$$g(x) := \sum_{j=0}^{m} (-1)^j \cdot E_j(\mu^{w_1}, \ldots, \mu^{w_\ell}) \cdot x^j .$$

Using Lemma 2.5, it is immediate that $g(x) = \prod_{i=1}^{\ell}(x - \mu^{w_i})^{\lambda_i}$. Further, by definition, $\deg(g) = m$. From g, now we want to extract the roots, namely $\mu^{w_1}, \ldots, \mu^{w_\ell}$ over $\mathbb{F}_q$. We do this, by checking whether $(x - \mu^i)$ divides g, for $i \in [n]$ (since $w_i \le n$). Using Lemma 2.6, a single division with remainder takes $\tilde{O}(k)$, therefore, the total time to find all the w_i is $\tilde{O}(nk) = \tilde{O}(nk)$.

Here, we *remark* that we do not use the deterministic root finding or factoring algorithms (for e.g. [Sho90; BKS15]), since it takes $\tilde{O}(mq^{1/2}) = \tilde{O}(k \cdot (k+t)^{1/2})$ time, which could be larger than $\tilde{O}(k(n+t))$.

Reason for choosing q and μ. In hindsight, there are three important properties of the prime q that will suffice to successfully output the w_i's using the above described steps:

1. Since, Lemma A.2 *requires* to compute the inverses of numbers upto t, hence, we would want $q > t$.

2. While computing $E_j(\mu^{w_1}, \ldots, \mu^{w_k})$ using Lemma 2.3 in the above, one should be able to compute the inverse of all j's less than equal to m. So, we want $q > m$,.

3. To obtain w_i from $\mu^{w_i} \bmod q$, we want $\mathrm{ord}_q(\mu) > n$ (for definition see Definition 2.7). Since $w_i \leq n$, this would ensure that we have found the correct w_i.

Here, we remark that we do not need to concern ourselves about the 'largeness' of the coefficients of C_j and make it nonzero $\bmod q$, as required in [JW18]. For the first two points, it suffices to choose $q > k + t$. Since μ is a primitive root over $\mathbb{F}_q$, this guarantees that $\mathrm{ord}_q(\mu) = q - 1 > n$ and thus we will find w_i from μ^{w_i} correctly.

Total time complexity. The complexity to find the correct m, q and μ is $\tilde{O}(n + k + t)$. Finding the coefficients of g takes $\tilde{O}(k(n + t))$ and then finding w_i from g takes $\tilde{O}(nk)$ time. Thus, the total complexity remains $\tilde{O}(k(n + t))$. $\square$

▶ Remark. The above algorithm can be extended to find the multiplicities λ_i's in $\tilde{O}(k(n + t) + k^{3/2})$ by finding the largest λ_i, by binary search, such that $(x - \mu^{w_i})^{\lambda_i}$ divides $g(x)$. Finding each λ_i takes $\tilde{O}(m \log(\lambda_i))$ over $\mathbb{F}_q$, for the same q as above, since the polynomial division takes $\tilde{O}(m)$ time and binary search introduces a multiplicative $O(\log(\lambda_i))$ term. Since, $\sum_{i \in [\ell]} \log(\lambda_i) = \log\left(\prod_{i \in [\ell]} \lambda_i\right)$, using AM-GM, $\prod_{i \in [\ell]} \lambda_i \leq (m/\ell)^{\ell}$, which is maximized at $\ell = \sqrt{m} \leq \sqrt{k}$, implying $\sum_{i \in [\ell]} \log(\lambda_i) \leq O(\sqrt{k} \log k)$. Since $m \leq k$, this explains the additive $k^{3/2}$ term in the complexity.

```
Input: A $k-\mathsf{SSSUM}$ instance $a_1, \ldots, a_n, t$
Output: Hamming weights of all subsets $S \subseteq [n]$ such that $\sum_{i\in S} a_i = t$
Using Lemma A.2, find the number of solutions $m$ for the $k-\mathsf{SSSUM}$
 instance $a_1, \ldots, a_n, t$ and terminate if $m = 0$;
Choose a prime $q$ from the interval $[k+t, 6/5\cdot(k+t)]$ ;
Find a primitive root $\mu$ over $\mathbb{F}_q$;
for $j \in [m]$ do
    Using $q$ in Lemma A.2, find $C_j = \mathrm{coef}_{x^t}(f_j(x))$ where
     $f_j(x) = \prod_{i\ =n}(1+\mu^j x^{a_i})$;
end
Compute $E_0, E_1, \ldots, E_m$ from $P_1, \ldots, P_m$ where $P_i \equiv C_i \mod q$ using FFT;
$W = \{\}$;
for $i \in [n]$ do
    if $(x-\mu^i) \mid g(x)$ then
        $W = W \cup \{i\}$;
    end
end
return $W$;
```

Algorithm 6: Algorithm for $\mathsf{Hamming}-k-\mathsf{SSSUM}$

7.2.2 Time-Efficient Algorithm for Subset Product

In this section, we give a randomized $\tilde{O}(n + t^{o(1)})$ expected time algorithm for Subset Product. Essentially, we factor all the entries in the instance in $\tilde{\mathrm{O}}(n + t^{o(1)})$ expected time. Once we have the exponents, it suffices to solve the corresponding SimulSubsetSum instance. Now, we can use the efficient randomised algorithm for SimulSubsetSum (Theorem 7.7) to finally solve Subset Product. So, first we give an efficient algorithm for SimulSubsetSum.

Theorem 7.7 (Algorithm for SimulSubsetSum)**.** *There is a randomized $\tilde{O}(kn + \prod_{i\in[k]}(2t_i + 1))$-time algorithm that solves* SimulSubsetSum*, with target instances*

$t_1, \ldots, t_k$.

Proof. Let us assume that the input to the $\mathsf{SimulSubsetSum}$ problem are k SSUM instances of the form $(a_{1j}, \ldots, a_{nj}, t_j)$, for $j \in [k]$. Define a k-variate polynomial $f(\boldsymbol{x})$, where $\boldsymbol{x} = (x_1, \ldots, x_k)$, as follows:

$$f(\boldsymbol{x}) \; = \; \prod_{i=1}^{n} \left(1 + \prod_{j=1}^{k} x_j^{a_{ij}} \right) .$$

Here is an immediate but important claim. We denote the monomial $\boldsymbol{m} := \prod_{i=1}^{k} x_i^{t_i}$ and $\mathrm{coef}_{\boldsymbol{m}}(f)$ as the coefficient of $\boldsymbol{m}$ in the polynomial $f(\boldsymbol{x})$.

Claim 7.8. *There is a solution to the* $\mathsf{SimulSubsetSum}$ *instance, i.e.,* $\exists S \subseteq [n]$ *such that* $\sum_{i \in S} a_{ij} = t_j, \forall j \in [k]$ *iff* $\mathrm{coef}_{\boldsymbol{m}}(f(\boldsymbol{x})) \neq 0$.

Therefore, it is enough to compute the coefficient of $f(\boldsymbol{x})$. The rest of the proof focuses on computing $f(\boldsymbol{x})$ efficiently, to find $\mathrm{coef}_{\boldsymbol{m}}(f)$.

Let p be prime such that $p \in [N+1, (n+N)^3]$, where $N := \prod_{i=1}^{k}(2t_i+1)$. Define an ideal $\mathcal{I}$, over $\mathbb{Z}[\boldsymbol{x}]$ as follows: $\mathcal{I} := \langle x_1^{t_1+1}, \ldots, x_k^{t_k+1}, p \rangle$. Since, we are interested in $\mathrm{coef}_{\boldsymbol{m}}(f)$, it suffices to compute $f(\boldsymbol{x}) \bmod \langle x_1^{t_1+1}, \ldots, x_k^{t_k+1} \rangle$, and we do it over a field $\mathbb{F}_p$ (which introduces error); for details, see the proof in the end (Randomness and error probability paragraph).

Using Lemma A.4, we can compute all the coefficient of $\ln(f(\boldsymbol{x})) \bmod \mathcal{I}$ in time $\tilde{O}(kn + \prod_{i=1}^{k} t_i)$. It is easy to see that the following equalities hold.

$$f(\boldsymbol{x}) \mod \mathcal{I} \; \equiv \; \exp\left(\ln(f(\boldsymbol{x}))\right) \mod \mathcal{I} \; \equiv \; \exp\left(\ln(f(\boldsymbol{x})) \bmod \mathcal{I}\right) \mod \mathcal{I} \, .$$

Since we have already computed $\ln(f(\boldsymbol{x})) \bmod \mathcal{I}$, the above equation implies that it is enough to compute the exponential which can be done using Lemma A.3. This also takes time $\tilde{O}(kn + \prod_{i=1}^{k}(2t_i+1))$.

Randomness and error probability. Note that there are $\Omega(n+N)^2$ primes in the interval $[N+1,(n+N)^3]$. Moreover, since $\text{coef}_{\boldsymbol{m}}(f) \le 2^n$, at most n prime factors can divide $\text{coef}_{\boldsymbol{m}}(f(\boldsymbol{x}))$. Therefore, we can pick a prime p randomly from this interval in $\mathsf{poly}(\log(n+N))$ time and the probability of p dividing the coefficient is $O(n+N)^{-1}$. In other words, the probability that the algorithm fails is bounded by $O\left((n+N)^{-1}\right)$. This concludes the proof.

$\square$

We now compare the above result with some obvious attempts to solve the SimulSubsetSum problem, before moving into solving the Subset Product problem.

A detailed comparison with time complexity of [Kan10]. Kane [Kan10, Section 3.3] showed that the above problem can be solved deterministically in $C^{O(k)}$ time and $O(k \log C)$ space, where $C := \sum_{i,j} a_{ij} + \sum_j t_j + 1$, which could be as large as $(n+1)\cdot(\sum_{j\in[k]} t_j) + 1$, since a_{ij} can be as large as t_j. As argued in [JVW21, Corollary 3.4 and Remark 3.5], the constant in the exponent, inside the order notation, can be as large as 3 (in fact directly using [Kan10] gives larger than 3; but modified algorithm as used in [JVW21] gives 3). Use AM-GM inequality to get

$$\left((n+1)\cdot(\sum_j t_j)+1\right)^{3k} > \left(\frac{2}{k}\cdot\sum_j t_j + 1\right)^{3k} \overset{\text{AM-GM}}{\ge} \prod_{j=1}^{k}(2t_j+1)^3 .$$

Assuming $N = \prod_{j=1}^{k}(2t_j+1)$, our algorithm is near-linear in N while Kane's algorithm [Kan10] takes at $O(N^3)$ time; thus ours is almost a cubic improvement.

Comparison with the trivial algorithm. It is easy to see that a trivial $O(n \cdot (t_1+1)(t_2+1)\ldots(t_k+1))$ time *deterministic* algorithm for SimulSubsetSum exists. Since, $t_i \ge 1$, we have

$$\frac{n}{2}\cdot\prod_{i\in[k]}(1+t_i) \ge \frac{n}{2}\cdot 2^k \ge kn\,, \quad \text{and} \quad \frac{n}{2}\cdot\prod(1+t_i) \ge \frac{n}{2^{k+1}}\cdot\prod(2t_i+1)\,.$$

Here, we used $2(1+x) > (2x+1)$, for any $x \ge 1$. Therefore, $n\cdot\prod_{i\in[k]}(1+t_i) \ge$

$kn + n/2^{k+1} \cdot \prod(2t_i + 1)$. Thus, when $k = o(\log n)$, our complexity is better.

Proof of Theorem 1.10

Once we have designed the algorithm for SimulSubsetSum, we design a time-efficient algorithm for Theorem 1.10.

Proof. Let $(a_1, \ldots, a_n, t) \in \mathbb{Z}_{\geq 0}^{n+1}$ be the input for Subset Product problem. Without loss of generality, we can assume that all the a_i divides t because if some a_i does not divide t, it will never be a part of any solution and we can discard it. Let us first consider the prime factorisation of t and a_j, for all $j \in [n]$. We will discuss about its time complexity in the next paragraph. Let

$$t \;=\; \prod_{j=1}^{k} p_j^{t_j}\,, \qquad a_i \;=\; \prod_{j=1}^{k} p_j^{e_{ij}}, \forall\, i \;\in\; [n]\,,$$

where p_j are distinct primes and t_j are positive integers and $e_{ij} \in \mathbb{Z}_{\geq 0}$. Since, $p_i \geq 2$, trivially, $\sum_{i=1}^{k} t_i \leq \log(t)$, and $\sum_{i=1}^{k} e_{ij} \leq \log(t), j \in [n]$. Also, the number of distinct prime factors of t is at most $O(\log(t)/\log\log(t))$; therefore, $k = O(\log(t)/\log\log(t))$.

Time complexity of factoring To find all the primes that divide t, we will use the factoring algorithm given by Lenstra and Pomerance [LP92] which takes expected $t^{o(1)}$ [1] time to completely factor t into prime factors p_j (including the exponents t_j). Using the primes p_j and the fact that $0 \leq e_{ij} \leq \log(t)$, computing e_{ij} takes $\log^2(t)\log\log(t)$ time, by performing binary search to find the largest x such that $p_j^x \mid a_i$. So, the time to compute all exponents $e_{i,j}, \forall i \in [n], j \in [k]$ is $O(nk\log^2(t)\log\log(t))$. Since, $k \leq O(\log t/\log\log(t))$, the total time complexity is $\tilde{O}(n + t^{o(1)})$.

[1]Expected time complexity is $\exp(O(\sqrt{\log t \log\log t}))$, which is smaller than $t^{O(1/\sqrt{\log\log t})} = t^{o(1)}$, which will be the time taken in the next step. Moreover, we are interested in *randomized* algorithms, hence expected run-time is $t^{o(1)}$

Setting up SimulSubsetSum Now suppose that $S \subseteq [n]$ is a solution to the Subset Product problem, i.e., $\prod_{i \in S} a_i = t$. This implies that

$$\sum_{i \in S} e_{ij} \; = \; t_j \, , \quad \forall \, j \, \in \, [k] \, .$$

In other words, we have a SimulSubsetSum instance where the j^{th} SSUM instance is $(e_{1j}, e_{2j}, \ldots, e_{nj}, t_j)$, for $j \in [k]$. The converse is also trivially true. We now show that there exists an $\tilde{O}(kn + \prod_{i \in [k]}(2t_i + 1))$ time algorithm to solve SimulSubsetSum.

Randomized algorithm for Subset Product Using Theorem 7.7, we can decide the SimulSubsetSum problem with targets $t_1, \ldots, t_k$ in $\tilde{O}(kn + \prod_{i \in [k]}(2t_i + 1))$ time (randomized) while working over $\mathbb{F}_p$ for some suitable p (we point out towards the end). Since $k \leq O(\log(t)/\log\log(t))$, we need to bound the term $\prod_{i \in [k]}(2t_i + 1)$. Note that,

$$\begin{aligned} \prod_{i \in [k]} (2t_i + 1) \; &= \; \sum_{S \subseteq [k]} 2^{|S|} \cdot \left(\prod_{i \in S} t_i \right) \\ &\leq \; 2^{2k} \cdot \left(\prod_{i \in [k]} t_i \right) \; . \end{aligned}$$

We now focus on bounding the term $\prod_{i \in [k]} t_i$. By AM-GM,

$$\begin{aligned} \prod_{i \in [k]} t_i \leq \left(\frac{\sum_{i \in [k]} t_i}{k} \right)^k &\leq \left(\frac{\log(t)}{k} \right)^k \\ &\leq 2^{O\left(\sqrt{k \log(t)}\right)} \qquad [\textit{Lemma } 2.10] \\ &\leq 2^{O\left(\sqrt{\log(t)^2 / \log\log(t)}\right)} \\ &\leq t^{O(1/\sqrt{\log\log(t)})} = t^{o(1)} \end{aligned}$$

Note that the prime p in the Theorem 7.7 was $p \in [N+1, (n+N)^3]$, where $N := \prod_{i=1}^{k}(2t_i + 1) - 1$. As shown above, we can bound $N = t^{o(1)}$. Thus, $p \leq O((n + t^{o(1)})^3)$, as desired. Therefore, the total time complexity is $\tilde{O}(n \log(t)/\log\log(t) + t^{o(1)}) = \tilde{O}(n + t^{o(1)})$. This finishes the proof. $\square$

Removing the expected-time If one wants to understand the worst-case analysis, we can use the polynomial time reduction from Subset Product to SimulSubsetSum in Section 7.4. Of course, we will not get prime factorisation; but the pseudo-prime factors will also be good enough to set up the SimulSubsetSum with similar parameters as above, and the SimulSubsetSum instance can be similarly solved in $\tilde{O}(n+t^{o(1)})$ time. Since the reduction takes $n^2\mathsf{poly}(\log t)$ time, the total time complexity becomes $\tilde{O}(n^2+t^{o(1)})$.

7.3 Space-Efficient Algorithms

7.3.1 Space-Efficient Algorithm for $k-\mathsf{SSSUM}$

In this section, we will present a low space algorithm (Algorithm 7) for finding all the realisable sets for $k-\mathsf{SSSUM}$. Unfortunately, proof of Theorem 1.9 *fails* to give a low space algorithm, since Lemma A.2 requires $\Omega(t)$ space (eventually it needs to store all the coefficients mod x^{t+1}). Instead, we work with a multivariate polynomial $f(x,y_1,\ldots,y_n)=\prod_{i=1}^{n}(1+y_ix^{a_i})$ over $\mathbb{F}_q$, for a large prime $q=O(nt)$ and its multiple evaluations $f(\alpha,c_1,\ldots,c_n)$, where $(\alpha,c_1,\ldots,c_n)\in\mathbb{F}_q^{n+1}$.

Observe that, the coefficient of x^t in f is a multivariate polynomial $p_t(y_1,\ldots,y_n)$; each of its monomial carries the *necessary information* of a solution, for the instance $(a_1,\ldots,a_n,t)$. More precisely, S is a realisable set of $(a_1,\ldots,a_n,t)\iff\prod_{i\in S}y_i$ is a monomial in p_t. And, the sparsity (number of monomials) of p_t is at most k. Therefore, it boils down to reconstructing the multivariate polynomial p_t efficiently. We cannot use the trivial multiplication since it takes $\tilde{O}(2^nt)$ time! Instead, we use ideas from [Kan10] and [KS01].

Proof of Theorem 1.11. Here are some notations that we will follow throughout the proof.

Basic notations. Let us assume that there are exactly m $(m\le k)$ many realisable sets $S_1,\ldots,S_m$, each $S_i\subseteq[n]$. We remark that for our algorithm we do not need to apriori calculate m.

The multivariate polynomial. For our purpose, we will be working with the following $(n+1)$-variate polynomial:

$$f(x, y_1, \ldots, y_n) \; := \; \prod_{i \in [n]} \left(1 + y_i x^{a_i}\right) \; .$$

Since, we have a $k - \mathsf{SSSUM}$ instance $(a_1, \ldots, a_n, t)$, $\operatorname{coef}_{x^t}(f)$ has the following properties.

1. It is an n-variate polynomial $p_t(y_1, \ldots, y_n)$ with sparsity *exactly* m.
2. p_t is a multilinear polynomial in $y_1, \ldots, y_n$, i.e., individual degree of y_i is at most 1.
3. The total degree of p_t is at most n.
4. if $S \subseteq [n]$ is a realisable set, then $\mathbf{y}_S := \prod_{i \in S} y_i$, is a monomial in p_t.

In particular, the following is an immediate but important observation.

Observation 7.9. $p_t(y_1, \ldots, y_n) \; = \; \sum_{i \in [m]} \boldsymbol{y}_{S_i}$.

Therefore, it suffices to know the polynomial p_t. However, we cannot treat y_i as new variables and try to find the coefficient of x^t since the trivial multiplication algorithm (involving $n+1$ variables) takes $\exp(n)$-time. This is because, $f(x, y_1, \ldots, y_n) \mod x^{t+1}$ can have $2^n \cdot t$ many monomials as coefficient of x^i, for any $i \leq t$ can have 2^n many multilinear monomials.

However, if we substitute $y_i = c_i \in \mathbb{F}_q$, for some prime q, we claim that we can figure out the value $p_t(c_1, \ldots, c_n)$ from the coefficient of x^t in $f(x, c_1, \ldots, c_n)$ efficiently (see Claim 7.11). Once we have figured it out, we can simply interpolate using the following theorem to reconstruct the polynomial p_t. Before going into the technical details, we state the sparse interpolation theorem below; for simplicity, we consider multilinearity (though [KS01] holds for general polynomials as well).

Theorem 7.10 ([KS01]). *Given a blackbox access to a multilinear polynomial $g(x_1, \ldots, x_n)$ of degree d and sparsity at most s over a finite field $\mathbb{F}$ with $|\mathbb{F}| \geq (nd)^6$, there is a $\mathsf{poly}(snd)$-time and $O(\log(snd))$-space algorithm that outputs all the monomials of g.*

▶ Remark. We represent one monomial in terms of indices (to make it consistent with the notion of realisable set), i.e., for a monomial $x_1x_5x_9$, the corresponding indices set is $\{1, 5, 9\}$. Also, we do not include the indices in the space complexity, as mentioned earlier.

Brief analysis on the space complexity of [KS01]. Klivans and Spielman [KS01], did not explicitly mention the space complexity. However, it is not hard to show that the required space is indeed $O(\log(snd))$. [KS01] shows that substituting $x_i = y^{k^{i-1} \bmod p}$, for some $k \in [2s^2n]$ and $p > 2s^2n$, makes the exponents of the new univaraite polynomial (in y) *distinct* (see [KS01, Lemma 3]); the algorithm actually tries for all k and find the correct k. Note that the degree becomes $O(s^2nd)$. Then, it tries to first find out the coefficients by simple univariate interpolation [KS01, Section 6.3]. Since we have blackbox access to $g(a_1, \ldots, a_n)$, finding out a single coefficient, by univariate interpolation (which basically sets up linear equations and solves it) takes $O(\log(snd))$ space and $\mathsf{poly}(snd)$ time only. In the last step, to find one coefficient, we can use the standard univariate interpolation algorithm which uses the Vandermonde matrices and one entry of the inverse of the Vandermonde is log-space computable [2].

At this stage, we know the coefficients (one by one), but we do not know which monomials the coefficients belong to. However, it suffices to substitute $x_i = 2y^{k^{i-1} \bmod p}$. Using this, we can find the correct value of the first exponent in the monomial. For e.g. if after the correct substitution, y^{10} appears with coefficient say 5, next step, when we change just x_1, if it does not affect the coefficient 5, y_1 is not there in the monomial corresponding to the monomial which has co-

[2]In fact Vandermonde determinant and inverse computations are in $\mathsf{TC}^0 \subset \mathsf{LOGSPACE}$, see [MT98].

efficient 5, otherwise it is there (here we also use that it is multilinear and hence the change in the coefficient must be reflected). This step again requires univariate interpolation, and one has to repeat this experiment with respect to each variable to know the monomial exactly corresponding to the coefficient we are working with. We can reuse the space for interpolation and after one round of checking with every variable, it outputs one exponent at this stage. This requires $O(\log(snd)$-space and $\mathsf{poly}(snd)$ time.

With a more careful analysis, one can further improve the field requirement to $|\mathbb{F}| \geq (nd)^6$ only (and not dependent on s); for details, see [KS01, Theorem 5 & 11].

Now we come back to our subset sum problem. Since we want to reconstruct an n-variate m sparse polynomial p_t which has degree at most n, it suffices to work with $|\mathbb{F}| \geq n^{12}$. However, we also want to use Kane's identity (Lemma 2.2), which requires $q > \deg(f(x, c_1, \ldots, c_n)) + 2$, and $\deg(f(x, c_1, \ldots, c_n)) \leq nt$. Denote $M := \max(nt + 3, n^{12})$. Thus, it suffices to we work with $\mathbb{F} = \mathbb{F}_q$ where $q \in [M, (6/5) \cdot M]$, such prime exists (Theorem 2.9) and easy to find deterministically in $\mathsf{poly}(nt)$ time and $O(\log(nt))$ space using [AKS04]. In particular, we will substitute $y_i = c_i \in [0, q-1]$.

Claim 7.11. *Fix $c_i \in [0, q-1]$, where $q \in [M, (6/5) \cdot M]$. Then, there is a $\mathsf{poly}(nt)$-time and $O(\log(nt))$ space algorithm which computes $p_t(c_1, \ldots, c_n)$ over $\mathbb{F}_q$.*

Proof. Note that, we can evaluate each $1 + c_i x^{a_i}$, at some $x = \alpha \in \mathbb{F}_q$, in $\tilde{O}(\log nt)$ time and $O(\log(nt))$ space. Multiplying n of them takes $\tilde{O}(n \log(nt))$-time and $O(\log(nt))$ space.

Once we have computed $f(\alpha, c_1, \ldots, c_n)$ over $\mathbb{F}_q$, using Kane's identity (Lemma 2.2), we can compute $p_t(c_1, \ldots, c_n)$, since

$$p_t(c_1, \ldots, c_n) \;=\; -\sum_{\alpha \in \mathbb{F}_q^*} \alpha^{q-1-t} f(\alpha, c_1, \ldots, c_n)$$

As each evaluation $f(\alpha, c_1, \ldots, c_n)$ takes $\tilde{O}(n \log(nt))$ time, and we need $q - 1$ many additions, multiplications and modular exponentiations, the total time to compute is $\mathsf{poly}(nt)$. The required space still remains $O(\log(nt))$. $\square$

Once, we have calculated $p_t(c_1, \ldots, c_n)$ efficiently, now we try different values of $(c_1, \ldots, c_n)$ to reconstruct p_t using Theorem 7.10. Since, p_t is a n-variate at most k sparse polynomial with a degree at most n, it still takes $\mathsf{poly}(knt)$ time and $O(\log(knt))$ space. This finishes the proof. $\square$

```
Input: A $k-\mathsf{SSSUM}$ instance $a_1, \ldots, a_n, t$
Output: All realisable subsets $S \subseteq [n]$ such that $\sum_{i \in S} a_i = t$
Pick a prime $q \in \{M, (6/5) \cdot M\}$ where $M = max(nt + 3, n^{12})$;
Let $\mathcal{O}$ be the algorithm mentioned in Theorem 7.10;
for each $p_t(c_1, \ldots, c_n)$ query requested by $\mathcal{O}$ do
    Send $-(\sum_{\alpha \in \mathbb{F}_q^*} \alpha^{q-1-t} f(\alpha, c_1, \ldots, c_n))$ to $\mathcal{O}$;
end
$p_t$ be the polynomial return by $\mathcal{O}$;
$\mathcal{F} = \{\}$;
for each monomial $\boldsymbol{y}_S$ in $p_t$ do
    $\mathcal{F} = \mathcal{F} \cup \{S\}$
end
return $\mathcal{F}$;
```
Algorithm 7: Algorithm for $k-\mathsf{SSSUM}$

7.3.2 Space-Efficient Algorithm for Subset Product

The proof of Theorem 1.12 uses the idea of reducing Subset Product to an instance of SimulSubsetSum and then solving SimulSubsetSum by computing the coefficient of

$$f(\boldsymbol{x}) = \prod_{i=1}^{n} \left(1 + \prod_{j=1}^{k} x_j^{a_{ij}}\right)$$

where $\boldsymbol{x} = (x_1, \ldots, x_k)$ using an extension of Lemma 2.2. We cannot directly use [Kan10] as it requires large space ($O(n \log(nt))$ space to be precise) to store the SimulSubsetSum instance. Instead we compute the coefficient of $f(\boldsymbol{x})$ without storing the SimulSubsetSum instance using Lemma 7.12.

The low space algorithm presented in this proof depends on the generalisation of Lemma 2.2. Here we present Kane's identity for *bi*variate polynomials which can be easily extended to k-variate polynomials.

Lemma 7.12 (Identity lemma [Kan10]). *Let* $f(x,y) = \sum_{i=0}^{d_1}\sum_{j=0}^{d_2} c_{i,j}x^iy^j$ *be a polynomial of degree at most* d_1+d_2 *with coefficients* $c_{i,j}$ *being integers. Let* $\mathbb{F}_q$ *be the finite field of order* $q = p^m > \max(d_1,d_2)+1$. *For* $0 \le t_1 \le d_1, 0 \le t_2 \le d_2$, *define*

$$r_{t_1,t_2} = \sum_{x\in\mathbb{F}_q^*}\sum_{y\in\mathbb{F}_q^*} x^{q-1-t_1}y^{q-1-t_2}f(x,y) = c_{t_1,t_2} \in \mathbb{F}_q$$

Proof. Let n be a positive integer, then the two following identities hold:

1. Identity 1:- $\sum_{x\in\mathbb{F}_q^*} x^n = -1$ if $q-1\,|\,n$ because $x^n = x^{(q-1)m} = 1$ due to Fermat's Little theorem.

2. Identity 2:- $\sum_{x\in\mathbb{F}_q^*} x^n = 0$, if $q-1 \nmid n$. This is because we can rewrite the summation as $\sum_{i=0}^{q-2} g^{i\cdot n} = \dfrac{g^{n(q-1)}-1}{g^n-1} = 0$ where g is a generator of $\mathbb{F}_q^*$.

Let us now consider $\sum_{x\in\mathbb{F}_q^*}\sum_{y\in\mathbb{F}_q^*} x^{q-1-t_1}y^{q-1-t_2}f(x,y)$.

$$\begin{aligned}
\sum_{x\in\mathbb{F}_q^*}\sum_{y\in\mathbb{F}_q^*} x^{q-1-t_1}y^{q-1-t_2}f(x,y) &= \sum_{x\in\mathbb{F}_q^*}\sum_{y\in\mathbb{F}_q^*} x^{q-1-t_1}y^{q-1-t_2}\left(\sum_{i=0}^{d_1}\sum_{j=0}^{d_2} c_{i,j}x^iy^j\right)\\
&= \sum_{i=0}^{d_1}\sum_{j=0}^{d_2} c_{i,j}\left(\sum_{x\in\mathbb{F}_q^*}\sum_{y\in\mathbb{F}_q^*} x^{q-1-t_1+i}y^{q-1-t_2+j}\right)\\
&= \sum_{\substack{i\in[0,d_1]\setminus\{t_1\}\\ j\in[0,d_2]\setminus\{t_2\}}} c_{i,j}\left(\sum_{x\in\mathbb{F}_q^*}\sum_{y\in\mathbb{F}_q^*} x^{q-1-t_1+i}y^{q-1-t_2+j}\right)\\
&\quad + c_{t_1,t_2}\\
&= c_{t_1,t_2}
\end{aligned}$$

Observe that when $i \in [0,d_1]\setminus\{t_1\}$, we have $\sum_{x\in\mathbb{F}_q^*} x^{q-1-t_1+i} = 0$ because $|i-t_1| \le d_1 < q-1 \implies q-1+i-t_1$ is not a multiple of $q-1$. The same goes for $j \in [0,d_2]\setminus\{t_2\}$.

□

▶ Remark. Lemma 7.12 can be easily extended to k variables which was used by the authors of [Kan10] to solve SimulSubsetSum with k many SSUM instances in space $O(k\log(n\sum_{i=1}^{k} t_i))$ and time $(poly(n, t_1, \dots, t_k))^{O(k)}$. In this case, the order of the finite field must be greater than $\max(d_1, \dots, d_k) + 1$ where d_i's are the individual degrees of the polynomial.

Issue with directly invoking [Kan10]. Using Theorem 1.13, we can reduce a Subset Product instance $(a_1, \dots, a_n, t)$ to a SimulSubsetSum instance containing k SSUM instances $(e_{1i}, \dots, e_{ni}, t_i), \forall i \in [k]$ where $k \leq \log(t)$. The space required for the SimulSubsetSum instance is the number of bits in e_{ij}, t_j. We know that $a_i = \prod_{j=1}^{k} p_i^{e_{ij}} \implies 2^{\sum_{j=1}^{k}\log(e_{ij})} \leq 2^{\sum_{j=1}^{k} e_{ij}} \leq a_i$ because $p_i \geq 2, \forall i \in [k]$. Therefore, we have $\sum_{i,j}\log(e_{i,j}) \leq \sum_{i=1}^{n}\log(a_i) \leq n\log(t)$. Similarly, $\sum_i \log(t_i) \leq \log(t)$. Therefore, the space required for the SimulSubsetSum is $O(n\log(t))$. And, if we directly use the low-space algorithm for SimulSubsetSum from [Kan10], the total space complexity would become $O((n + \log(nt)) \cdot \log(t))$.

To avoid the n-factor in the space complexity, we will not be storing the entire SimulSubsetSum instance. Instead, for each summation in the k variate version of Lemma 7.12, we will compute the values of e_{ij} and t_j and discard them after using them. To be precise, for $\overline{g} = (t_1, \dots, t_k)$, we have

$$c_{\overline{g}} = (-1)^k \sum_{\boldsymbol{x} \in (\mathbb{F}_q^*)^k} f(\boldsymbol{x}) \cdot \prod_{i=1}^{k} x_i^{q-1-t_i}$$

where $f(\boldsymbol{x}) = \prod_{i=1}^{n}\left(1 + \prod_{j=1}^{k} x_j^{e_{ij}}\right)$ and $c_{\overline{g}} = \text{coef}_{\overline{g}}(f(\boldsymbol{x}))$. The values of e_{ij} and t_i is only required in $f(\boldsymbol{x})$ and $\prod_{i=1}^{k} x_i^{q-1-t_i}$ respectively. Since, e_{ij} and t_i are the powers of p_i in a_j and t respectively, we can't use pseudo-prime-factorisation as this would require us to use $O(n\log(t))$ space to compute a pseudo-prime-factor set. Therefore, we will use naive prime-factorisation algorithm that runs in $\tilde{O}(t)$ time which is affordable because we are interested in $\mathsf{poly}(knt)$.

Choosing the prime q. Observe that the total degree of $f(\boldsymbol{x})$ is $\sum_{ij} e_{ij} \leq n \cdot (\sum_i t_i) \leq n \log t$ because $0 \leq e_{ij} \leq t_j$ and $\sum_i t_i \leq \log(t)$. Therefore, the maximum individual degree is bounded by $n \log(t)$. Since, Lemma 7.12 requires a prime q that depends on the maximum individual degree of the polynomial, it suffices to work with $N = \lceil n \log(t) \rceil$ and $q > N$. Observe that we need to compute the coefficient modulo q, therefore, we need to ensure that q does not divide the coefficient. To achieve this, we will use Lemma 7.12 for different primes $q \in [N+1, (n+N)^3]$ which contains $\Omega(n+N)^2$ prime. This works because the coefficient can be at most 2^n, therefore, it will have at most n prime factors. So, at least one prime in the range will not divide the coefficient.

Computing $f(\boldsymbol{x})$ and $\prod_{i=1}^{k} x_i^{q-1-t_i}$ using low space. We will make sure that t_i is the exponent of the i^{th} *smallest* prime factor of t. To find an e_{ij}, we will first find the i^{th} smallest prime p_i that divides t and then compute the largest power of p_i that divides a_j. Once, we find e_{ij}, we can use it to compute $\prod_{j=1}^{k} x_j^{e_{ij}}$ part of $f(\boldsymbol{x})$ and discard it as shown in Algorithm 8. Similarly, we can compute $\prod_{i=1}^{k} x_i^{q-1-t_i}$.

Space and Time complexity. Observe that Algorithm 8 uses only $O(\log(nt))$ space for variables that are used throughout the algorithm and reuses $O(\log(t))$ space while computing t_i, e_{ij}, p_ℓ values. It uses $k \log(nt) = O(\log^2(nt))$ space for $\overline{y}$, therefore, the total space complexity is $O(\log^2(nt))$. Whereas the time complexity is $\mathsf{poly}(nt)$ because each loop runs for $\mathsf{poly}(nt)$ iterations and finding the exponents take $\tilde{O}(t)$ time.

7.4 An Efficient Reduction from Subset Product to SimulSubsetSum

In this section, we will present a deterministic polynomial time reduction from Subset Product to SimulSubsetSum. In Section 7.2.2, we have given a pseudo-polynomial time reduction from Subset Product to SimulSubsetSum by performing prime factori-

```
Input: A Subset Product instance $(a_1, \dots, a_n, t) \in \mathbb{N}^{n+1}$
Output: Decides whether the Subset Product instance has a solution
$k = 0$;
for each prime $p_i \mid t$ do
    $k = k + 1$;
end
$N = \lceil n \log(t) \rceil$;
for each prime $q \in [N + 1, (n + N)^3]$ do
    $c_g = 1$;
    for each $\overline{y} \in (\mathbb{F}_q^*)^k$ do
        $prodx_1 = 1$;
        for $i \in [k]$ do
            Compute $i^{th}$ smallest prime that divides $t$ and find $t_i$;
            $prodx_1 = prodx_1 * y_i^{q-1-t_i}$;
            Discard $t_i$;
        end
        $f = 1$;
        for $i \in [n]$ do
            $prodx_2 = 1$;
            for $j \in [k]$ do
                Compute $j^{th}$ smallest prime $p_j$ that divides $t$;
                Using $p_j$ compute $e_{ij}$ which is the largest integer such that
                  $p_j^{e_{ij}} \mid a_i$;
                $prodx_2 = prodx_2 * y_j^{e_{ij}}$;
                Discard $p_j$ and $e_{ij}$;
            end
            $f = f * (1 + prodx_2)$;
        end
        $c_g = f * prodx_1$;
    end
    if $c_g \neq 0$ then
        return True;
    end
end
return False;
```

Algorithm 8: Algorithm for solving Subset Product using low space

sation of the input $(a_1, \ldots, a_n, t)$. The polynomial time reduction also requires to factorise the input, but the factors are not necessarily prime. To be precise, we define pseudo-prime-factorisation which can be achieved in polynomial time.

Definition 7.13 (Pseudo-prime-factorization)**.** *A set of integers $\mathcal{P} \subset \mathbb{N}$ is said to be pseudo-prime-factor set of $(a_1, \ldots, a_n) \in \mathbb{N}^n$ if*

1. *the elements of $\mathcal{P}$ are pair-wise coprime, i.e., $\forall p_1, p_2 \in \mathcal{P}, gcd(p_1, p_2) = 1$,*
2. *there are only non-trivial factors of a_i's in $\mathcal{P}$, i.e., $\forall p \in \mathcal{P}, \exists i \in [n]$ such that $p \,|\, a_i$,*
3. *every a_i's can be uniquely expressed as a product of powers of elements of $\mathcal{P}$, i.e., $\forall i \in [n], a_i = \prod_{p \in \mathcal{P}} p^{e_p}, \forall i \in [n]$ where $e_p \geq 0$.*

For a given $(a_1, \ldots, a_n)$, $\mathcal{P}$ may not be unique. A trivial example of a pseudo-prime-factor set of $\mathcal{P}$ for $(a_1, \ldots, a_n)$ is the set of all distinct prime factors of $\prod_{i=1}^n a_i$. The following is an important claim which will be used to give a *polynomial* time reduction from Subset Product to SimulSubsetSum.

Claim 7.14. *For any pseudo-prime-factor set $\mathcal{P}$ of $(a_1, \ldots, a_n)$, we have $|\mathcal{P}| \leq k$ where k is the number of distinct prime factors of $\prod_{i=1}^n a_i$.*

Proof. The proof uses a simple pigeonhole principle argument. Let $g_1, \ldots, g_k$ be the distinct prime factors of $\prod_{i=1}^n a_i$. From the definition of $\mathcal{P}$, we know that $g_1, \ldots, g_k$ are the only distinct prime factors of $\prod_{p \in \mathcal{P}} p$. Therefore, if there are more than k numbers in $\mathcal{P}$, then there must exist $p_1, p_2 \in \mathcal{P}$ such that $gcd(p_1, p_2) \neq 1$ which violates pair-wise coprime property of $\mathcal{P}$. □

▶ Constructing $\mathcal{P}$ suffices. We now show that having a pseudo-prime-factor set $\mathcal{P}$ for $(a_1, \ldots, a_n, t)$ helps us to reduce a Subset Product instance $(a_1, \ldots, a_n, t)$ to SimulSubsetSum with number of instances $|\mathcal{P}|$, in polynomial time. Without loss of generality, we can assume that $a_i \,|\, t$ and $a_i, t \leq 2^m, \forall i \in [n]$ for some m. Trivially, $m \leq \log t$. So, using Claim 7.14, we have $|\mathcal{P}| \leq (n+1) \cdot m = \mathsf{poly}(n \log t)$.

From Definition 7.13, we have unique non-negative integers e_{ij} and t_j such that $t = \prod_{j \in |\mathcal{P}|} p_j^{t_j}$ and $a_i = \prod_{j \in |\mathcal{P}|} p_j^{e_{ij}}, \forall i \in [n]$. Since, $a_i \,|\, t$, we have $e_{ij} \le t_j \le m, \forall i \in [n], j \in [|\mathcal{P}|]$ and they can be computed in $\mathsf{poly}(m,n)$ time.

Let us consider the $|P| - \mathsf{SimulSubsetSum}$ instance where the i^{th} SSUM instance is $(e_{1i}, e_{2i}, \ldots, e_{ni}, t_i)$. Then, due to unique factorization property of $\mathcal{P}$, the $\mathsf{Subset\ Product}$ instance is YES, i.e., $\exists S \subseteq [n]$ such that $\prod_{i \in S} a_i = t$ iff the $\mathsf{SimulSubsetSum}$ instance with number of instances $|\mathcal{P}|$, is a YES.

7.4.1 Polynomial Time Algorithm for Computing Pseudo-Prime-Factors

We will now present a deterministic polynomial time algorithm for computing a pseudo-prime-factor set $\mathcal{P}$ for $(a_1, \ldots, a_n)$. We will use the notation $\mathcal{P}(a_1, \ldots, a_n)$ to denote a pseudo-prime-factor set for $(a_1, \ldots, a_n)$. Also, let $\mathcal{S}(a_1, \ldots, a_n)$ be the set of all pseudo-prime-factor sets; this is a finite set.

The following lemma is a crucial component in Algorithm 9. We use $a//b$ to denote a/b^e such that $b^{e+1} \nmid a$.

Lemma 7.15. *Let $(a_1, \ldots, a_n)$ be n integers. Then,*

1. *If a_1 is coprime with $a_i, \forall i > 1$, then for any $\mathcal{P}(a_2, \ldots, a_n) \in \mathcal{S}(a_2, \ldots, a_n)$, $\mathcal{P}(a_2, \ldots, a_n) \cup \{a_1\} \in \mathcal{S}(a_1, \ldots, a_n)$.*

2. *$\mathcal{P}(g, a_1//g, a_2//g, \ldots, a_n//g) \in \mathcal{S}(a_1, \ldots, a_n)$, for given a_i, $i \in [n]$ and any factor g of some a_i.*

Proof. The first part of the lemma is trivial. For the second part, let g be a non-trivial factor of some a_i and

$$\mathcal{P} \;:=\; \{p_1, \ldots, p_k\} \;\in\; \mathcal{S}(g, a_1//g, a_2//g, \ldots, a_n//g)\,,$$

be any pseudo-prime-factor set. Then, p_i's are pair-wise coprime and since each p_i divides either g or $a_i//g$ for some $i \in [n]$, it also divides some a_i because g is a factor of some a_i. Also, we have *unique* non-negative integers e_{ip}, e_{gp} s.t.

$$a_i//g = \prod_{p \in \mathcal{P}} p^{e_{ip}}, \forall i \in [n] \text{ and } g = \prod_{p \in \mathcal{P}} p^{e_{gp}} .$$

Combining these equation, we get $a_i = a_i//g * g^{f_{ig}} = \prod_{p \in \mathcal{P}} p^{e_{ip}+e_{gp}*f_{ig}}$. Here f_{ig} is the maximum power of g that divides a_i. Therefore, $\{p_1, \ldots, p_k\}$ is also a pseudo-prime-factor set for $(a_1, \ldots, a_n)$. $\square$

Pre-processing. Using Lemma 7.15, Algorithm 9 performs a divide-and-conquer approach to find $\mathcal{P}(a_1, \ldots, a_n)$. Observe that we can always remove duplicate elements and 1's from the input since it *does not* change the pseudo-prime-factors. Also, we can assume without loss of generality that $a_i//a_1 =: a_i, \forall i > 1$ because of the second part in Lemma 7.15, with $g = a_1$, since it gives us $\mathcal{P}(a_1, a_2//g, \ldots, a_n//g)$ and we know it suffices to work with these inputs.

If a_1 is coprime to the rest of the a_i's, then the algorithm will recursively call itself on $(a_2, \ldots, a_n)$ and combine $\mathcal{P}(a_2, \ldots, a_n)$ with $\{a_1\}$. Else, there exist an $i > 1$ such that $gcd(a_1, a_i) \neq 1$. So, the algorithm finds a factor g of a_1 using Euclid's GCD algorithm and computes $\mathcal{P}(g, a_1//g, \ldots, a_n//g)$. At every step, we remove duplicates and 1's. Hence, the correctness of Algorithm 9 is immediate, assuming it terminates.

To show the termination and time complexity of Algorithm 9, we will use the *'potential function'* $\mathbb{P}(I) := \prod_{a \in I} a$, where I is the input and show that at each recursive call, the value of the potential function is halved. Initially, the value of the potential function is $\prod_{i=1}^{n} a_i$. We also remark that since the algorithm removes duplicates and 1's; the potential function can *never* increase by the removal step and so it never matters in showing the decreasing nature of $\mathbb{P}$.

1. a_1 is coprime to the rest of the a_i's: In this case, the recursive call has input $(a_2, \ldots, a_n)$. Since, $a_1 \geq 2$, the value of potential function is

$$\mathbb{P}(a_2, \ldots, a_n) \; = \; \prod_{i=2}^{n} a_i \; < \; (\prod_{i=1}^{n} a_i)/2 \; = \; \mathbb{P}(a_1, \ldots, a_n)/2$$

2. a_1 shares a common factor with some a_i. Let $g = gcd(a_1, a_i) \neq 1$. Since, we have assumed $a_i//a_1 = a_i$, this implies that a_i is not a multiple of a_1. This implies that $2 \leq g \leq a_1/2$. Therefore, the new value of potential function is

$$\begin{aligned}
\mathbb{P}(g, a_1//g, \ldots, a_n//g) &= g\prod_{j=1}^{n} a_j//g \\
&\leq (a_1//g) \times ((a_i//g) \times g) \times \prod_{j\in[n]\setminus\{1,i\}} a_j \\
&\leq \frac{a_1}{g} \cdot \prod_{j=2}^{n} a_j \\
&\leq (\prod_{j=1}^{n} a_j)/2 = \mathbb{P}(a_1, \ldots, a_n)/2 \,.
\end{aligned}$$

We used the fact that since, $2 \leq g \,|\, a_i$, therefore, $g \times (a_i//g) \leq a_i$.

Time complexity. In both cases, the value of the potential function is halved. So, the depth of the recursion tree (in-fact, it is just a line) is at most $\log(\prod_{i=1}^{n} a_i) \leq m{\cdot}n$. Also, in each recursive call, the input size is increased at most by one but the integers are still bounded by 2^m. This implies that input size, for any recurrence call, can be at most $(m + 1) \cdot n$. Since there is no branching, the total time complexity is $\mathsf{poly}(m, n) = \mathsf{poly}(\log t, n)$.

7.5 Extending Theorems to Unbounded Subset Sum

In this section, we efficient algorithm for UBSSUM. The UBSSUM is an unbounded variant of SSUM problem, which is also NP-hard [Joh85].

Definition 7.16 (Unbounded Subset Sum (UBSSUM)). *Given $(a_1, \ldots, a_n, t) \in \mathbb{Z}_{\geq 0}^{n+1}$, the* UBSSUM *problem asks whether there exists $\beta_1, \ldots, \beta_n$ such that β_i are non-negative integers and $\sum_{i=1}^{n} \beta_i a_i = t$.*

Similar to the SSUM, the UBSSUM problem also has a $O(nt)$ dynamic program-

```
Input: $(a_1, a_2 \ldots, a_n) \in \mathbb{N}^n$ which are $m$-bit integers such that
       $a_i // a_1 = a_i > 1, \forall i > 1$
Output: Pseudo-prime-factor set $\mathcal{P}$ for $(a_1, a_2, \ldots, a_n)$
if $n == 0$ then
    return $\emptyset$;
end
if $\exists i > 1$ such that $gcd(a_1, a_i) \neq 1$ then
    $g = gcd(a_1, a_i)$;
    $I = \{g\}$;
    for $i \in [n]$ do
        $a'_i = a_i // g$;
        if $a'_i \notin I$ and $a'_i \neq 1$ then
            $I = I \cup \{a'_i\}$
        end
    end
    return $\mathcal{P}(I)$;
end
else
    return $\mathcal{P}(a_2, \ldots, a_n) \cup \{a_1\}$;
end
```

Algorithm 9: Algorithm for Pseudo-prime-factor set

ming algorithm. Interestingly, this problem has a $O(n + \min_i a_i^2)$-time deterministic algorithm [HR96]. Recently, Bringmann [Bri17] gave an $\tilde{O}(t)$ deterministic algorithm for UBSSUM. We now define two variants of the UBSSUM problem which is very similar to $k - \mathsf{SSSUM}$ and $\mathsf{Hamming} - k - \mathsf{SSSUM}$.

Definition 7.17 ($k - \mathsf{SUBSSUM}$). *Given $(a_1, \ldots, a_n, t) \in \mathbb{Z}_{\geq 0}^{n+1}$, the $k - \mathsf{SUBSSUM}$ problem asks to output all $(\beta_1, \ldots, \beta_n)$ where β_i are non-negative integers and $\sum_{i=1}^n \beta_i a_i = t$ provided the number of such solutions is at most k.*

Definition 7.18 ($\mathsf{Hamming} - k - \mathsf{SUBSSUM}$). *Given a $k - \mathsf{SUBSSUM}$ instance $(a_1, \ldots, a_n, t) \in \mathbb{Z}_{\geq 0}^{n+1}$, $\mathsf{Hamming} - k - \mathsf{SUBSSUM}$ asks to output all the hamming weights of the solutions, i.e., $\sum_{i=1}^n \beta_i$.*

Remark 7.19. *We want $\vec{a} \cdot \vec{v} = t$, where $\vec{v} \in \mathbb{Z}_{\geq 0}^n$. Similarly, like in the SSUM case (i.e., $\vec{v} \in \{0,1\}^n$), we want $|v|_1$, which is exactly the quantity $\sum_i \beta_i$, as above. Thus, this definition can be thought of as a natural extension of the hamming weight of the solution, in the unbounded regime.*

We will present a deterministic polynomial time reduction from UBSSUM to SimulSubsetSum, which will be used later in this section.

Theorem 7.20 (UBSSUM reduces to SimulSubsetSum)**.** *There exists a deterministic polynomial time reduction from* UBSSUM *to* SimulSubsetSum*.*

Proof. Let $(a_1, \ldots, a_n, t) \in \mathbb{Z}_{\geq 0}^{n+1}$ be an instance of UBSSUM. The reduction generates the following SSUM instance

$$(\underbrace{a_1, 2a_1, 4a_1, \ldots, 2^{\gamma}a_1}_{\gamma+1 \text{ entries}}, \underbrace{a_2, 2a_2, \ldots, 2^{\gamma}a_2}_{\gamma+1 \text{ entries}}, \ldots, \underbrace{a_n, 2a_n, \ldots, 2^{\gamma}a_n}_{\gamma+1 \text{ entries}}, t)$$

of size $n(\gamma+1)$ where $\gamma = \lfloor \log(t) \rfloor$.

Let $(\beta_1, \ldots, \beta_n)$ be a solution to the UBSSUM instance, i.e., $\sum_{i=1}^{n} \beta_i a_i = t$. Since, β_i, a_i, t are all non-negative integers, we have $\beta_i \leq t, \forall i \in [n]$. Therefore, β is at most $(\gamma+1)$-bit integer. Let $\beta_i^{(j)}$ be the j^{th} bit of β_i, then we have

$$t = \sum_{i=1}^{n} \beta_i a_i = \sum_{i=1}^{n} \left(\sum_{j=0}^{\gamma} \beta_i^{(j)} 2^j \right) \cdot a_i = \sum_{i=1}^{n} \sum_{j=0}^{\gamma} \beta_i^{(j)} \cdot (2^j a_i)$$

which implies that the SSUM instance also has a solution. Similarly, we can show the reverse direction, i.e., if SSUM instance has a solution, then UBSSUM is also has a solution. This concludes the proof. □

▶ Remark. Observe that in Theorem 7.20, there is a one-to-one correspondence between the solutions of the UBSSUM and the solutions of the SimulSubsetSum instance. Therefore, the reduction *preserves* the number of solutions. Also, any $T(n,t)$ time algorithm that solve SSUM gives an $T(n\log(t), t)$-time algorithm to solve UBSSUM.

We will now show that the Theorem 1.9-1.11 can be extended to, in the UBSSUM regime.

Theorem 7.21. *There is an $\tilde{O}(k(n+t))$-time deterministic algorithm for* Hamming$-$ $k-$ SUBSSUM*.*

Proof sketch. The algorithm is almost similar to Algorithm 6, except the definitions of the polynomials $f_j(x)$. Here also, we fix q and μ similarly. We require the exact number of solutions $m (m \leq k)$ in Section 7.2.1 (see Claim 7.4). To do that, define the polynomial f_0:

$$f_0(x) := \prod_{i=1}^{n} \left(\frac{1}{1 - x^{a_j}} \right) = \left(\prod_{i=1}^{n} (1 - x^{a_i}) \right)^{-1} = \left(\prod_{i=1}^{n} (1 + x^{a_i} + x^{2a_i} + \ldots) \right)$$
$$=: (h_0(x))^{-1} .$$

In the above, we used the *inverse identity* $1/(1-x) = \sum_{i \geq 0} x^i$. Expanding the above, is easy to see that $\text{coef}_{x^t}(f_0(x)) = m$, where m is the *exact* number of solutions to the $k-\mathsf{SUBSSUM}$. Note that, we can compute $f_0^{-1} = h_0(x) \bmod x^{t+1}$, over $\mathbb{F}_q$ efficiently in $\tilde{O}(k+t)$ time. Finding inverse is *easy* and can be done efficiently (see [VG13, Theorem 9.4]).

Once, we know m, we define m many polynomials f_j, for $j \in [m]$, as follows.

$$f_j(x) := \prod_{i=1}^{n} \left(\frac{1}{1 - \mu^j x^{a_j}} \right) = \left(\prod_{i=1}^{n} (1 - \mu^j x^{a_i}) \right)^{-1} =: (h_j(x))^{-1}$$

It is not hard to observe that $\text{coef}_{x^t}(f_j(x)) = \sum_{i \in [\ell]} \lambda_i \cdot \mu^{j w_i}$, where $w_1, \ldots, w_\ell$ are the distinct hamming weights with multiplicities $\lambda_1, \ldots, \lambda_\ell$ (similar to Observation 7.6). To find the coefficients of $f_j(x)$, we first compute the coefficients of $h_j(x)$, using Lemma A.2, in $\tilde{O}(k(n+t))$ time and find its inverses, using [VG13, Theorem 9.4], which can again be done in $\tilde{O}(k(n+t))$ time. Once we have computed the coefficients of $f_j(x)$, the rest proceeds the same as Section 7.2.1.

Theorem 7.22. *There is a* $\mathsf{poly}(knt)$*-time and* $O(\log(knt))$*-space deterministic algorithm which solves* $k - \mathsf{SUBSSUM}$.

Proof idea. The algorithm first reduces UBSSUM to SSUM using Theorem 7.20 which preserves the number of solutions but the size of the SSUM instance is now $n \log t$. Then, it runs Algorithm 7 on the SSUM instance to find all its solutions.

From the solutions of the SSUM instance, it constructs all the solutions of the UBSSUM instance. This gives $\mathsf{poly}(knt)$-time and $O(\log(knt\log t)) = O(\log(knt))$-space algorithm.

Chapter 8

Conclusions

In this thesis, we present a hardness result for Problem 1.1, new preimage attacks against Keccak, distinguishers in the weak key setting and key-recovery attacks against TinyJAMBU, distinguishers for Ascon and algorithms for variants of subset sum problem. We now present some questions which require further rigorous investigations.

8.1 Scope for Further Work

1. In Chapter 3, we are *unable* to show NP-hardness of L. We do not know whether $\oplus\mathsf{P}$ contains NP or not. In fact, $\mathsf{P}^{\oplus\mathsf{P}}$ is not known to contain NP. Therefore, it will be interesting to show NP-hardness of the above problem. Also, it would be interesting to study the complexity of this problem in the quantum setting.

2. Is it possible to use the non-linear structure techniques in the cryptanalysis of other sponge-based cryptosystems?

3. One of the major issues while implementing the monomial prediction technique using MILP is the blow-up in the number of variables as the number of rounds increases. Can we come up with tricks to control this blow-up?

4. Can we improve the time complexity of Theorem 1.11? Because of using

Theorem 7.10, the complexity for interpolation is already cubic. Whether some other algebraic (non-algebraic) techniques can improve the time complexity, while keeping it low space, is not at all clear.

5. Can we use these algebraic-number-theoretic techniques, to give a *deterministic* $\tilde{O}(n+t)$ time algorithm for decision version of SSUM?

6. Can we improve Section 7.2.1 to find both the hamming weights w_i as well as the multiplicities λ_i, in $\tilde{O}(k(n+t))$ time?

7. Can we improve the complexity of Theorem 7.7 to $\tilde{O}(n+\sum_{i=1}^{k} t_i)$?

8. What can we say about the hardness of SimulSubsetSum with k subset sum instances where $k=\omega(\log(n))$?

References

[Nag52] Jitsuro Nagura. “On the interval containing at least one prime number”. In: *Proceedings of the Japan Academy* 28.4 (1952), pp. 177–181.

[Bel57] Richard E Bellman. *Dynamic Programming.* 1957.

[RS62] J Barkley Rosser and Lowell Schoenfeld. “Approximate formulas for some functions of prime numbers”. In: *Illinois Journal of Mathematics* 6.1 (1962), pp. 64–94.

[Ber70] Elwyn R Berlekamp. “Factoring polynomials over large finite fields”. In: *Mathematics of computation* 24.111 (1970), pp. 713–735.

[Bre76] Richard P Brent. “Multiple-precision zero-finding methods and the complexity of elementary function evaluation”. In: *Analytic computational complexity.* Elsevier, 1976, pp. 151–176.

[MH78] Ralph Merkle and Martin Hellman. “Hiding information and signatures in trapdoor knapsacks”. In: *IEEE transactions on Information Theory* 24.5 (1978), pp. 525–530.

[GJ79] Michael R Garey and David S Johnson. *Computers and intractability.* Vol. 174. freeman San Francisco, 1979.

[HW+79] Godfrey Harold Hardy, Edward Maitland Wright, et al. *An introduction to the theory of numbers.* Oxford university press, 1979.

[CZ81] David G Cantor and Hans Zassenhaus. “A new algorithm for factoring polynomials over finite fields”. In: *Mathematics of Computation* (1981), pp. 587–592.

[Dix81] John D Dixon. “Asymptotically fast factorization of integers”. In: *Mathematics of computation* 36.153 (1981), pp. 255–260.

[LLL82] Arjen Klaas Lenstra, Hendrik Willem Lenstra, and László Lovász. “Factoring polynomials with rational coefficients”. In: *Mathematische Annalen* 261.4 (1982), pp. 515–534.

[Sha82] Adi Shamir. “A polynomial time algorithm for breaking the basic Merkle-Hellman cryptosystem”. In: *Foundations of Computer Science, 1982. SFCS'08. 23rd Annual Symposium on.* IEEE. 1982, pp. 145–152.

[Adl83] Leonard M Adleman. “On breaking generalized knapsack public key cryptosystems”. In: *Proceedings of the fifteenth annual ACM symposium on Theory of computing.* 1983, pp. 402–412.

[Kan83] Ravi Kannan. “Improved algorithms for integer programming and related lattice problems”. In: *Proceedings of the fifteenth annual ACM symposium on Theory of computing.* ACM. 1983, pp. 193–206.

[Rob83] Guy Robin. “Estimation de la fonction de Tchebychef θ sur le k-ième nombre premier et grandes valeurs de la fonction ω (n) nombre de diviseurs premiers de n”. In: *Acta Arithmetica* 42.4 (1983), pp. 367–389.

[GM84] Rajiv Gupta and M Ram Murty. “A remark on Artin’s conjecture”. In: *Inventiones mathematicae* 78.1 (1984), pp. 127–130.

[Odl84] A Odlyzko. “Cryptanalytic attacks on the multiplicative knapsack cryptosystem and on Shamir’s fast signature scheme”. In: *IEEE Transactions on Information Theory* 30.4 (1984), pp. 594–601.

[Joh85] David S Johnson. “The NP-completeness column: an ongoing guide”. In: *Journal of Algorithms* 6.3 (1985), pp. 434–451.

[LO85] Jeffrey C Lagarias and Andrew M Odlyzko. “Solving low-density subset sum problems”. In: *Journal of the ACM (JACM)* 32.1 (1985), pp. 229–246.

[Bab86] László Babai. “On Lovász’lattice reduction and the nearest lattice point problem”. In: *Combinatorica* 6.1 (1986), pp. 1–13.

[MS86] Judy H. Moore and Gustavus J. Simmons. “Cycle Structures of the DES with Weak and Semi-Weak Keys”. In: *Advances in Cryptology - CRYPTO ’86, Santa Barbara, California, USA, 1986, Proceedings.* Ed. by Andrew M. Odlyzko. Vol. 263. Lecture Notes in Computer Science. Springer, 1986, pp. 9–32.

[Kan87] Ravi Kannan. “Minkowski’s convex body theorem and integer programming”. In: *Mathematics of operations research* 12.3 (1987), pp. 415–440.

[MVV87] Ketan Mulmuley, Umesh V Vazirani, and Vijay V Vazirani. “Matching is as easy as matrix inversion”. In: *Proceedings of the nineteenth annual ACM symposium on Theory of computing.* 1987, pp. 345–354.

[Sch87] Claus-Peter Schnorr. “A hierarchy of polynomial time lattice basis reduction algorithms”. In: *Theoretical computer science* 53.2-3 (1987), pp. 201–224.

[BLR90] Manuel Blum, Michael Luby, and Ronitt Rubinfeld. “Self-Testing/Correcting with Applications to Numerical Problems”. In: *Proceedings of the 22nd Annual ACM Symposium on Theory of Computing, May 13-17, 1990, Baltimore, Maryland, USA*. Ed. by Harriet Ortiz. ACM, 1990, pp. 73–83.

[Len+90] Arjen K Lenstra et al. “The number field sieve”. In: *Proceedings of the twenty-second annual ACM symposium on Theory of computing.* 1990, pp. 564–572.

[Sho90] Victor Shoup. “On the deterministic complexity of factoring polynomials over finite fields”. In: *Information Processing Letters* 33.5 (1990), pp. 261–267.

[LP92] Hendrik W Lenstra and Carl Pomerance. “A rigorous time bound for factoring integers”. In: *Journal of the American Mathematical Society* 5.3 (1992), pp. 483–516.

[BB93] Ishai Ben-Aroya and Eli Biham. “Differential Cryptanalysis of Lucifer”. In: *Advances in Cryptology - CRYPTO ’93, 13th Annual International Cryptology Conference, Santa Barbara, California, USA, August 22-26, 1993, Proceedings*. Ed. by Douglas R. Stinson. Vol. 773. Lecture Notes in Computer Science. Springer, 1993, pp. 187–199.

[Mat93] Mitsuru Matsui. “Linear cryptanalysis method for DES cipher”. In: *Workshop on the Theory and Application of of Cryptographic Techniques*. Springer. 1993, pp. 386–397.

[Pan+93] Victor Pan et al. “A new approach to fast polynomial interpolation and multipoint evaluation”. In: *Computers & Mathematics with Applications* 25.9 (1993), pp. 25–30.

[Evd94] Sergei Evdokimov. “Factorization of polynomials over finite fields in subexponential time under GRH”. In: *International Algorithmic Number Theory Symposium*. Springer. 1994, pp. 209–219.

[Knu94] Lars R. Knudsen. “Truncated and Higher Order Differentials”. In: *Fast Software Encryption: Second International Workshop. Leuven, Belgium, 14-16 December 1994, Proceedings*. Ed. by Bart Preneel. Vol. 1008. Lecture Notes in Computer Science. Springer, 1994, pp. 196–211.

[SE94] Claus-Peter Schnorr and Martin Euchner. “Lattice basis reduction: Improved practical algorithms and solving subset sum problems”. In: *Mathematical programming* 66.1-3 (1994), pp. 181–199.

[Cop96a] Don Coppersmith. “Finding a small root of a bivariate integer equation; factoring with high bits known”. In: *International Conference on the Theory and Applications of Cryptographic Techniques*. Springer. 1996, pp. 178–189.

[Cop96b] Don Coppersmith. “Finding a small root of a univariate modular equation”. In: *International Conference on the Theory and Applications of Cryptographic Techniques*. Springer. 1996, pp. 155–165.

[HR96] Paul Hansen and Jennifer Ryan. “Testing integer knapsacks for feasibility”. In: *European journal of operational research* 88.3 (1996), pp. 578–582.

[Hor96] Akos G Horváth. “On the Dirichlet—Voronoi cell of unimodular lattices”. In: *Geometriae Dedicata* 63.2 (1996), pp. 183–191.

[Shp96] Igor Shparlinski. “On finding primitive roots in finite fields”. In: *Theoretical computer science* 157.2 (1996), pp. 273–275.

[Aro+97] Sanjeev Arora et al. “The hardness of approximate optima in lattices, codes, and systems of linear equations”. In: *Journal of Computer and System Sciences* 54.2 (1997), pp. 317–331.

[Ajt98] Miklós Ajtai. “The Shortest Vector Problem in L2 is NP-hard for Randomized Reductions (Extended Abstract)”. In: *STOC*. 1998.

[CN98] Jin-Yi Cai and Ajay Nerurkar. “Approximating the SVP to within a factor (1-1/dim/sup/spl epsiv//) is NP-hard under randomized conditions”. In: *Proceedings. Thirteenth Annual IEEE Conference on Computational Complexity (Formerly: Structure in Complexity Theory Conference)(Cat. No. 98CB36247)*. IEEE. 1998, pp. 46–55.

[DKS98] Irit Dinur, Guy Kindler, and Shmuel Safra. “Approximating-CVP to within almost-polynomial factors is NP-hard”. In: *Foundations of Computer Science, 1998. Proceedings. 39th Annual Symposium on.* IEEE. 1998, pp. 99–109.

[Haw98] Philip Hawkes. “Differential-Linear Weak Key Classes of IDEA”. In: *Advances in Cryptology - EUROCRYPT ’98, International Conference on the Theory and Application of Cryptographic Techniques, Espoo, Finland, May 31 - June 4, 1998, Proceeding.* Ed. by Kaisa Nyberg. Vol. 1403. Lecture Notes in Computer Science. Springer, 1998, pp. 112–126.

[KS98] Erich Kaltofen and Victor Shoup. “Subquadratic-time factoring of polynomials over finite fields”. In: *Mathematics of computation* 67.223 (1998), pp. 1179–1197.

[MT98] Alexis Maciel and Denis Therien. “Threshold circuits of small majority-depth”. In: *Information and Computation* 146.1 (1998), pp. 55–83.

[MSK98] Shiho Moriai, Takeshi Shimoyama, and Toshinobu Kaneko. “Higher Order Differential Attak of CAST Cipher”. In: *Fast Software Encryption, 5th International Workshop, FSE ’98, Paris, France, March 23-25, 1998, Proceedings.* Ed. by Serge Vaudenay. Vol. 1372. Lecture Notes in Computer Science. Springer, 1998, pp. 17–31.

[EM99] Pál Erdös and M Ram Murty. “On the order of a (mod p)”. In: *CRM Proceedings and Lecture Notes.* Vol. 19. 1999, pp. 87–97.

[Gol+99] Oded Goldreich et al. “Approximating shortest lattice vectors is not harder than approximating closest lattice vectors”. In: *Information Processing Letters* 71.2 (1999), pp. 55–61.

[Pis99] David Pisinger. “Linear time algorithms for knapsack problems with bounded weights”. In: *Journal of Algorithms* 33.1 (1999), pp. 1–14.

[Bür00] Peter Bürgisser. *Completeness and reduction in algebraic complexity theory.* Vol. 7. Springer Science & Business Media, 2000.

[AKS01] Miklós Ajtai, Ravi Kumar, and Dandapani Sivakumar. “A sieve algorithm for the shortest lattice vector problem”. In: *Proceedings of the thirty-third annual ACM symposium on Theory of computing.* ACM. 2001, pp. 601–610.

[FMS01] Scott R. Fluhrer, Itsik Mantin, and Adi Shamir. “Weaknesses in the Key Scheduling Algorithm of RC4”. In: *Selected Areas in Cryptography, 8th Annual International Workshop, SAC 2001 Toronto, Ontario, Canada, August 16-17, 2001, Revised Papers.* Ed. by Serge Vaudenay and Amr M. Youssef. Vol. 2259. Lecture Notes in Computer Science. Springer, 2001, pp. 1–24.

[KS01] Adam R Klivans and Daniel Spielman. “Randomness efficient identity testing of multivariate polynomials”. In: *Proceedings of the thirty-third annual ACM symposium on Theory of computing.* 2001, pp. 216–223.

[MS01] Itsik Mantin and Adi Shamir. "A Practical Attack on Broadcast RC4". In: *Fast Software Encryption, 8th International Workshop, FSE 2001 Yokohama, Japan, April 2-4, 2001, Revised Papers*. Ed. by Mitsuru Matsui. Vol. 2355. Lecture Notes in Computer Science. Springer, 2001, pp. 152–164.

[Mic01] Daniele Micciancio. "The shortest vector in a lattice is hard to approximate to within some constant". In: *SIAM journal on Computing* 30.6 (2001), pp. 2008–2035.

[AKS02] Miklós Ajtai, Ravi Kumar, and Dandapani Sivakumar. "Sampling short lattice vectors and the closest lattice vector problem". In: *Proceedings 17th IEEE Annual Conference on Computational Complexity*. IEEE. 2002, pp. 53–57.

[CV02] Anne Canteaut and Marion Videau. "Degree of Composition of Highly Nonlinear Functions and Applications to Higher Order Differential Cryptanalysis". In: *Advances in Cryptology - EUROCRYPT 2002, International Conference on the Theory and Applications of Cryptographic Techniques, Amsterdam, The Netherlands, April 28 - May 2, 2002, Proceedings*. Ed. by Lars R. Knudsen. Vol. 2332. Lecture Notes in Computer Science. Springer, 2002, pp. 518–533.

[KW02] Lars R. Knudsen and David A. Wagner. "Integral Cryptanalysis". In: *Fast Software Encryption, 9th International Workshop, FSE 2002, Leuven, Belgium, February 4-6, 2002, Revised Papers*. Ed. by Joan Daemen and Vincent Rijmen. Vol. 2365. Lecture Notes in Computer Science. Springer, 2002, pp. 112–127.

[Cou03] Nicolas T. Courtois. "Fast Algebraic Attacks on Stream Ciphers with Linear Feedback". In: *Advances in Cryptology - CRYPTO 2003, 23rd Annual International Cryptology Conference, Santa Barbara, California, USA, August 17-21, 2003, Proceedings*. Ed. by Dan Boneh. Vol. 2729. Lecture Notes in Computer Science. Springer, 2003, pp. 176–194.

[CM03] Nicolas T. Courtois and Willi Meier. "Algebraic Attacks on Stream Ciphers with Linear Feedback". In: *Advances in Cryptology - EUROCRYPT 2003, International Conference on the Theory and Applications of Cryptographic Techniques, Warsaw, Poland, May 4-8, 2003, Proceedings*. Ed. by Eli Biham. Vol. 2656. Lecture Notes in Computer Science. Springer, 2003, pp. 345–359.

[Mal03] Guillaume Malod. "Polynômes et coefficients". PhD thesis. Université Claude Bernard-Lyon I, 2003.

[AKS04] Manindra Agrawal, Neeraj Kayal, and Nitin Saxena. "PRIMES is in P". In: *Annals of mathematics* (2004), pp. 781–793.

[CM04] Koji Chinen and Leo Murata. "On a distribution property of the residual order of a (modp)". In: *Journal of Number Theory* 105.1 (2004), pp. 60–81.

[Kho05] Subhash Khot. "Hardness of approximating the shortest vector problem in lattices". In: *Journal of the ACM (JACM)* 52.5 (2005), pp. 789–808.

[RO05] Vincent Rijmen and Elisabeth Oswald. "Update on SHA-1". In: *Topics in Cryptology–CT-RSA 2005: The Cryptographers' Track at the RSA Conference 2005, San Francisco, CA, USA, February 14-18, 2005. Proceedings*. Springer. 2005, pp. 58–71.

[Val05] Leslie G. Valiant. "Completeness for Parity Problems". In: *Computing and Combinatorics, 11th Annual International Conference, COCOON 2005, Kunming, China, August 16-29, 2005, Proceedings*. Vol. 3595. Lecture Notes in Computer Science. 2005, pp. 1–8.

[WYY05] Xiaoyun Wang, Yiqun Lisa Yin, and Hongbo Yu. "Finding collisions in the full SHA-1". In: *Advances in Cryptology–CRYPTO 2005: 25th Annual International Cryptology Conference, Santa Barbara, California, USA, August 14-18, 2005. Proceedings 25*. Springer. 2005, pp. 17–36.

[CLS06] Scott Contini, Arjen K Lenstra, and Ron Steinfeld. "VSH, an efficient and provable collision-resistant hash function". In: *Annual International Conference on the Theory and Applications of Cryptographic Techniques*. Springer. 2006, pp. 165–182.

[Coc07] Martin Cochran. "Notes on the Wang et al. 2^{63} SHA-1 Differential Path". In: *Cryptology ePrint Archive* (2007).

[HS07] Guillaume Hanrot and Damien Stehlé. "Improved analysis of Kannan's shortest lattice vector algorithm". In: *Annual International Cryptology Conference*. Springer. 2007, pp. 170–186.

[HR07] Ishay Haviv and Oded Regev. "Tensor-based hardness of the shortest vector problem to within almost polynomial factors". In: *Proceedings of the thirty-ninth annual ACM symposium on Theory of computing*. ACM. 2007, pp. 469–477.

[KM07] Orhun Kara and Cevat Manap. "A New Class of Weak Keys for Blowfish". In: *Fast Software Encryption, 14th International Workshop, FSE 2007, Luxembourg, Luxembourg, March 26-28, 2007, Revised Selected Papers*. Ed. by Alex Biryukov. Vol. 4593. Lecture Notes in Computer Science. Springer, 2007, pp. 167–180.

[AJ08] Vikraman Arvind and Pushkar S Joglekar. "Some sieving algorithms for lattice problems". In: *IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science*. Schloss Dagstuhl-Leibniz-Zentrum für Informatik. 2008.

[CP08] Christophe De Cannière and Bart Preneel. "Trivium". In: *New Stream Cipher Designs - The eSTREAM Finalists*. Ed. by Matthew J. B. Robshaw and Olivier Billet. Vol. 4986. Lecture Notes in Computer Science. Springer, 2008, pp. 244–266.

[Mic08] Daniele Micciancio. "Efficient reductions among lattice problems". In: *Proceedings of the nineteenth annual ACM-SIAM symposium on Discrete algorithms*. Society for Industrial and Applied Mathematics. 2008, pp. 84–93.

[NV08] Phong Q Nguyen and Thomas Vidick. "Sieve algorithms for the shortest vector problem are practical". In: *Journal of Mathematical Cryptology* 2.2 (2008), pp. 181–207.

[AM09] Jean-Philippe Aumasson and Willi Meier. “Zero-sum distinguishers for reduced Keccak-f and for the core functions of Luffa and Hamsi”. In: *rump session of Cryptographic Hardware and Embedded Systems-CHES* 2009 (2009), p. 67.

[Aum+09] Jean-Philippe Aumasson et al. “Cube Testers and Key Recovery Attacks on Reduced-Round MD6 and Trivium”. In: *Fast Software Encryption, 16th International Workshop, FSE 2009, Leuven, Belgium, February 22-25, 2009, Revised Selected Papers*. Ed. by Orr Dunkelman. Vol. 5665. Lecture Notes in Computer Science. Springer, 2009, pp. 1–22.

[BHV09] Cristina Bazgan, Hadrien Hugot, and Daniel Vanderpooten. “Solving efficiently the 0–1 multi-objective knapsack problem”. In: *Computers & Operations Research* 36.1 (2009), pp. 260–279.

[Ber+09] Guido Bertoni et al. “Keccak specifications”. In: *Submission to nist (round 2)* (2009), pp. 320–337.

[BN09] Johannes Blömer and Stefanie Naewe. “Sampling methods for shortest vectors, closest vectors and successive minima”. In: *Theoretical Computer Science* 410.18 (2009), pp. 1648–1665.

[CDK09] Christophe De Cannière, Orr Dunkelman, and Miroslav Knezevic . “KATAN and KTANTAN - A Family of Small and Efficient Hardware-Oriented Block Ciphers”. In: *Cryptographic Hardware and Embedded Systems - CHES 2009, 11th International Workshop, Lausanne, Switzerland, September 6-9, 2009, Proceedings*. Ed. by Christophe Clavier and Kris Gaj. Vol. 5747. Lecture Notes in Computer Science. Springer, 2009, pp. 272–288.

[Cor+09] Thomas H Cormen et al. *Introduction to algorithms*. MIT press, 2009.

[DS09] Itai Dinur and Adi Shamir. “Cube Attacks on Tweakable Black Box Polynomials”. In: *Advances in Cryptology - EUROCRYPT 2009, 28th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Cologne, Germany, April 26-30, 2009. Proceedings*. Ed. by Antoine Joux. Vol. 5479. Lecture Notes in Computer Science. Springer, 2009, pp. 278–299.

[Gen09] Craig Gentry. “Fully homomorphic encryption using ideal lattices”. In: *Proceedings of the forty-first annual ACM symposium on Theory of computing*. 2009, pp. 169–178.

[PS09] Xavier Pujol and Damien Stehlé. “Solving the Shortest Lattice Vector Problem in Time 22.465 n.” In: *IACR Cryptology ePrint Archive* 2009 (2009), p. 605.

[Ber10] Daniel J Bernstein. “Second preimages for 6 (7?(8??)) rounds of keccak”. In: *NIST mailing list* (2010).

[EJT10] Michael Elberfeld, Andreas Jakoby, and Till Tantau. “Logspace versions of the theorems of Bodlaender and Courcelle”. In: *2010 IEEE 51st Annual Symposium on Foundations of Computer Science*. IEEE. 2010, pp. 143–152.

[Kan10] Daniel M Kane. “Unary subset-sum is in logspace”. In: *arXiv preprint arXiv:1012.1336* (2010).

[Kay10] Neeraj Kayal. “Algorithms for Arithmetic Circuits.” In: *Electron. Colloquium Comput. Complex.* Vol. 17. 2010, p. 73.

[KP10] Mikhail Y Kovalyov and Erwin Pesch. “A generic approach to proving NP-hardness of partition type problems”. In: *Discrete applied mathematics* 158.17 (2010), pp. 1908–1912.

[KM10] Hidenori Kuwakado and Masakatu Morii. “Quantum distinguisher between the 3-round Feistel cipher and the random permutation”. In: *2010 IEEE International Symposium on Information Theory.* IEEE. 2010, pp. 2682–2685.

[LN10] Daniel Lokshtanov and Jesper Nederlof. “Saving space by algebraization”. In: *Proceedings of the Forty-second ACM Symposium on Theory of Computing.* 2010, pp. 321–330.

[LPS10] Vadim Lyubashevsky, Adriana Palacio, and Gil Segev. “Public-key cryptographic primitives provably as secure as subset sum”. In: *Theory of Cryptography Conference.* Springer. 2010, pp. 382–400.

[MV10] Daniele Micciancio and Panagiotis Voulgaris. “Faster exponential time algorithms for the shortest vector problem”. In: *Proceedings of the twenty-first annual ACM-SIAM symposium on Discrete Algorithms.* Society for Industrial and Applied Mathematics. 2010, pp. 1468–1480.

[AFV11] Shweta Agrawal, David Mandell Freeman, and Vinod Vaikuntanathan. “Functional encryption for inner product predicates from learning with errors”. In: *International Conference on the Theory and Application of Cryptology and Information Security.* Springer. 2011, pp. 21–40.

[Ber+11a] G Bertoni et al. *The Keccak reference. online at http://keccak. noekeon. org/Keccak-reference-3.0. pdf.* 2011.

[Ber+11b] Guido Bertoni et al. *Cryptographic sponges.* 2011.

[BCD11] Christina Boura, Anne Canteaut, and Christophe De Canniere. “Higher-order differential properties of Keccak and Luffa”. In: *International Workshop on Fast Software Encryption.* Springer. 2011, pp. 252–269.

[HPS11] Guillaume Hanrot, Xavier Pujol, and Damien Stehlé. “Algorithms for the shortest and closest lattice vector problems”. In: *International Conference on Coding and Cryptology.* Springer. 2011, pp. 159–190.

[Lea+11] Gregor Leander et al. “A Cryptanalysis of PRINT cipher: The Invariant Subspace Attack”. In: *Advances in Cryptology - CRYPTO 2011 - 31st Annual Cryptology Conference, Santa Barbara, CA, USA, August 14-18, 2011. Proceedings.* Ed. by Phillip Rogaway. Vol. 6841. Lecture Notes in Computer Science. Springer, 2011, pp. 206–221.

[NRM11] María Naya-Plasencia, Andrea Röck, and Willi Meier. “Practical Analysis of Reduced-Round Keccak.” In: *INDOCRYPT.* Vol. 7107. Springer. 2011, pp. 236–254.

[Wan+11] Xiaoyun Wang et al. “Improved Nguyen-Vidick heuristic sieve algorithm for shortest vector problem”. In: *Proceedings of the 6th ACM Symposium on Information, Computer and Communications Security*. ACM. 2011, pp. 1–9.

[DDS12] Itai Dinur, Orr Dunkelman, and Adi Shamir. “New Attacks on Keccak-224 and Keccak-256.” In: *FSE*. Vol. 12. Springer. 2012, pp. 442–461.

[Duc+12] Alexandre Duc et al. “Unaligned rebound attack: application to Keccak”. In: *International Workshop on Fast Software Encryption*. Springer. 2012, pp. 402–421.

[MG12] Daniele Micciancio and Shafi Goldwasser. *Complexity of lattice problems: a cryptographic perspective*. Vol. 671. Springer Science & Business Media, 2012.

[Mor12] Pieter Moree. “Artin’s primitive root conjecture–a survey”. In: *Integers* 12.6 (2012), pp. 1305–1416.

[Sha12] Andrew Shallue. “Division algorithms for the fixed weight subset sum problem”. In: *arXiv preprint arXiv:1201.2739* (2012).

[Aum+13] Jean-Philippe Aumasson et al. “Quark: A Lightweight Hash”. In: *J. Cryptol.* 26.2 (2013), pp. 313–339.

[Ber+13a] Daniel J Bernstein et al. “Quantum algorithms for the subset-sum problem”. In: *International Workshop on Post-Quantum Cryptography*. Springer. 2013, pp. 16–33.

[Ber+13b] Guido Bertoni et al. “Keccak”. In: *Annual international conference on the theory and applications of cryptographic techniques*. Springer. 2013, pp. 313–314.

[BC13] Christina Boura and Anne Canteaut. “On the Influence of the Algebraic Degree of F^{-1} on the Algebraic Degree of $G \circ F$”. In: *IEEE Trans. Inf. Theory* 59.1 (2013), pp. 691–702.

[CS13] John Horton Conway and Neil James Alexander Sloane. *Sphere packings, lattices and groups*. Vol. 290. Springer Science & Business Media, 2013.

[DDS13] Itai Dinur, Orr Dunkelman, and Adi Shamir. “Collision attacks on up to 5 rounds of SHA-3 using generalized internal differentials”. In: *International Workshop on Fast Software Encryption*. Springer. 2013, pp. 219–240.

[Köl+13] Stefan Kölbl et al. “Differential cryptanalysis of Keccak variants”. In: *IMA International Conference on Cryptography and Coding*. Springer. 2013, pp. 141–157.

[MV13] Daniele Micciancio and Panagiotis Voulgaris. “A deterministic single exponential time algorithm for most lattice problems based on Voronoi cell computations”. In: *SIAM Journal on Computing* 42.3 (2013), 1364–1391.

[MPS13] Paweł Morawiecki, Josef Pieprzyk, and Marian Srebrny. “Rotational cryptanalysis of round-reduced Keccak”. In: *International Workshop on Fast Software Encryption*. Springer. 2013, pp. 241–262.

[MS13] Paweł Morawiecki and Marian Srebrny. “A SAT-based preimage analysis of reduced KECCAK hash functions”. In: *Information Processing Letters* 113.10-11 (2013), pp. 392–397.

[VG13] Joachim Von Zur Gathen and Jürgen Gerhard. *Modern computer algebra.* Cambridge university press, 2013.

[Agg+14] Divesh Aggarwal et al. “Solving the shortest vector problem in 2n time via discrete Gaussian sampling. arXiv preprint”. In: *arXiv* 1412 (2014).

[Cha+14] Donghoon Chang et al. “1st and 2nd Preimage Attacks on 7, 8 and 9 Rounds of Keccak-224,256,384,512”. In: *SHA-3 workshop (August 2014).* 2014.

[DDS14] Itai Dinur, Orr Dunkelman, and Adi Shamir. “Improved practical attacks on round-reduced Keccak”. In: *Journal of cryptology* 27.2 (2014), pp. 183–209.

[MW14] Daniele Micciancio and Michael Walter. “Fast lattice point enumeration with minimal overhead”. In: *Proceedings of the twenty-sixth annual ACM-SIAM symposium on Discrete algorithms.* SIAM. 2014, pp. 276–294.

[WH14] Hongjun Wu and Tao Huang. “JAMBU lightweight authenticated encryption mode and AES-JAMBU”. In: *CAESAR competition proposal* (2014).

[ADS15] Divesh Aggarwal, Daniel Dadush, and Noah Stephens-Davidowitz. “Solving the Closest Vector Problem in 2^ n Time–The Discrete Gaussian Strikes Again!” In: *2015 IEEE 56th Annual Symposium on Foundations of Computer Science.* IEEE. 2015, pp. 563–582.

[Agg+15] Divesh Aggarwal et al. “Solving the shortest vector problem in 2 n time using discrete Gaussian sampling”. In: *Proceedings of the forty-seventh annual ACM symposium on Theory of computing.* ACM. 2015, pp. 733–742.

[BKS15] Jean Bourgain, Sergei Konyagin, and Igor Shparlinski. “Character sums and deterministic polynomial root finding in finite fields”. In: *Mathematics of Computation* 84.296 (2015), pp. 2969–2977.

[Dwo15] Morris J Dworkin. *SHA-3 standard: Permutation-based hash and extendable-output functions.* Tech. rep. 2015.

[JN15] Jérémy Jean and Ivica Nikolić. “Internal Dierential Boomerangs: Practical Analysis o the Round-Reduced Keccak-*f* Permutation”. In: *International Workshop on Fast Software Encryption.* Springer. 2015, pp. 537–556.

[LMR15] Gregor Leander, Brice Minaud, and Sondre Rønjom. “A Generic Approach to Invariant Subspace Attacks: Cryptanalysis of Robin, iSCREAM and Zorro”. In: *Advances in Cryptology - EUROCRYPT 2015 - 34th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Sofia, Bulgaria, April 26-30, 2015, Proceedings, Part I.* Ed. by Elisabeth Oswald and Marc Fischlin. Vol. 9056. Lecture Notes in Computer Science. Springer, 2015, pp. 254–283.

[Tod15] Yosuke Todo. “Structural Evaluation by Generalized Integral Property”. In: *Advances in Cryptology - EUROCRYPT 2015 - 34th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Sofia, Bulgaria, April 26-30, 2015, Proceedings, Part I*. Ed. by Elisabeth Oswald and Marc Fischlin. Vol. 9056. Lecture Notes in Computer Science. Springer, 2015, pp. 287–314.

[Cyg+16] Marek Cygan et al. “On problems as hard as CNF-SAT”. In: *ACM Transactions on Algorithms (TALG)* 12.3 (2016), pp. 1–24.

[FMV16] Sebastian Faust, Daniel Masny, and Daniele Venturi. “Chosen-ciphertext security from subset sum”. In: *Public-Key Cryptography–PKC 2016*. Springer, 2016, pp. 35–46.

[GVL16] Bruno Grenet, Joris Van Der Hoeven, and Grégoire Lecerf. “Deterministic root finding over finite fields using Graeffe transforms”. In: *Applicable Algebra in Engineering, Communication and Computing* 27.3 (2016), pp. 237–257.

[GLS16] Jian Guo, Meicheng Liu, and Ling Song. “Linear structures: applications to cryptanalysis of round-reduced Keccak”. In: *International Conference on the Theory and Application of Cryptology and Information Security*. Springer. 2016, pp. 249–274.

[Kim16] Sungjin Kim. “Average results on the order of a modulo p”. In: *Journal of Number Theory* 169 (2016), pp. 353–368.

[Wu16] Hongjun Wu. “ACORN: a lightweight authenticated cipher (v3)”. In: *Candidate for the CAESAR Competition* (2016).

[Xia+16] Zejun Xiang et al. “Applying MILP Method to Searching Integral Distinguishers Based on Division Property for 6 Lightweight Block Ciphers”. In: *Advances in Cryptology - ASIACRYPT 2016 - 22nd International Conference on the Theory and Application of Cryptology and Information Security, Hanoi, Vietnam, December 4-8, 2016, Proceedings, Part I*. Ed. by Jung Hee Cheon and Tsuyoshi Takagi. Vol. 10031. Lecture Notes in Computer Science. 2016, pp. 648–678.

[BGS17] Huck Bennett, Alexander Golovnev, and Noah Stephens-Davidowitz. “On the quantitative hardness of CVP”. In: *FOCS*. 2017.

[Bri17] Karl Bringmann. “A near-linear pseudopolynomial time algorithm for subset sum”. In: *Proceedings of the Twenty-Eighth Annual ACM-SIAM Symposium on Discrete Algorithms*. SIAM. 2017, pp. 1073–1084.

[Jao+17] David Jao et al. *SIKE – Supersingular Isogeny Key Encapsulation*. Manuscript avaiable at https://sike.org/. 2017.

[Li+17] Ting Li et al. “Preimage Attacks on the Round-reduced Keccak with Cross-linear Structures”. In: *IACR Transactions on Symmetric Cryptology* (2017), pp. 39–57.

[Qia+17] Kexin Qiao et al. “New Collision Attacks on Round-Reduced Keccak”. In: *Annual International Conference on the Theory and Applications of Cryptographic Techniques*. Springer. 2017, pp. 216–243.

[SLG17] Ling Song, Guohong Liao, and Jian Guo. "Non-full sbox linearization: Applications to collision attacks on round-reduced keccak". In: *Annual International Cryptology Conference*. Springer. 2017, pp. 428–451.

[Ste+17] Marc Stevens et al. "The first collision for full SHA-1". In: *Advances in Cryptology–CRYPTO 2017: 37th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 20–24, 2017, Proceedings, Part I 37*. Springer. 2017, pp. 570–596.

[Tod+17] Yosuke Todo et al. "Cube Attacks on Non-Blackbox Polynomials Based on Division Property". In: *Advances in Cryptology - CRYPTO 2017 - 37th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 20-24, 2017, Proceedings, Part III*. Ed. by Jonathan Katz and Hovav Shacham. Vol. 10403. Lecture Notes in Computer Science. Springer, 2017, pp. 250–279.

[AS18] Divesh Aggarwal and Noah Stephens-Davidowitz. "(Gap/S)ETH Hardness of SVP". In: *STOC*. 2018.

[HM18] Alexander Helm and Alexander May. "Subset Sum Quantumly in 1.17^ n". In: *13th Conference on the Theory of Quantum Computation, Communication and Cryptography (TQC 2018)*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik. 2018.

[JW18] Ce Jin and Hongxun Wu. "A simple near-linear pseudopolynomial time randomized algorithm for subset sum". In: *arXiv preprint arXiv:1807.11597* (2018).

[KX18] Konstantinos Koiliaris and Chao Xu. "Subset sum made simple". In: *arXiv preprint arXiv:1807.08248* (2018).

[KMS18] Rajendra Kumar, Nikhil Mittal, and Shashank Singh. "Cryptanalysis of 2 round Keccak-384". In: *International Conference on Cryptology in India*. Springer. 2018, pp. 120–133.

[MW18] Dylan M McKay and Richard Ryan Williams. "Quadratic time-space lower bounds for computing natural functions with a random oracle". In: *10th Innovations in Theoretical Computer Science Conference (ITCS 2019)*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik. 2018.

[NIS18] NIST. *National Institute of Standards and Technology. Lightweight Cryptography (LWC) Standardization project*. 2018.

[Abb+19] Amir Abboud et al. "SETH-based lower bounds for subset sum and bicriteria path". In: *Proceedings of the Thirtieth Annual ACM-SIAM Symposium on Discrete Algorithms*. SIAM. 2019, pp. 41–57.

[KX19] Konstantinos Koiliaris and Chao Xu. "Faster pseudopolynomial time algorithms for subset sum". In: *ACM Transactions on Algorithms (TALG)* 15.3 (2019), pp. 1–20.

[LS19] Ting Li and Yao Sun. "Preimage Attacks on Round-Reduced Keccak-224/256 via an Allocating Approach". In: *Annual International Conference on the Theory and Applications of Cryptographic Techniques*. Springer. 2019, pp. 556–584.

[LL19] Yang Li and Hongbo Li. “Improved quantum algorithm for the random subset sum problem”. In: *arXiv preprint arXiv:1912.09264* (2019).

[WH19] Hongjun Wu and Tao Huang. “TinyJAMBU: A Family of Lightweight Authenticated Encryption Algorithms”. In: *Submission to the NIST Lightweight Cryptography Standardization Process (March 2019)* (2019).

[Dob+20] Christoph Dobraunig et al. *ASCON v1. 2. submission to NIST (2019).* 2020.

[DMT20] Konstantinos A Draziotis, V Martidis, and S Tiganourias. “Product Subset Problem: Applications to number theory and cryptography”. In: *arXiv preprint arXiv:2002.07095* (2020).

[EM20] Andre Esser and Alexander May. “Low Weight Discrete Logarithm and Subset Sum in 20. 65n with Polynomial Memory”. In: *memory* 1 (2020), p. 2.

[GGR20] Pascal Giorgi, Bruno Grenet, and Daniel S Roche. “Fast in-place algorithms for polynomial operations: division, evaluation, interpolation”. In: *Proceedings of the 45th International Symposium on Symbolic and Algebraic Computation.* 2020, pp. 210–217.

[Guo20] Zeyu Guo. “Deterministic polynomial factoring over finite fields: a uniform approach via P-schemes”. In: *Journal of Symbolic Computation* 96 (2020), pp. 22–61.

[Hu+20] Kai Hu et al. “An algebraic formulation of the division property: Revisiting degree evaluations, cube attacks, and key-independent sums”. In: *International Conference on the Theory and Application of Cryptology and Information Security.* Springer. 2020, pp. 446–476.

[Sah+20] Dhiman Saha et al. “On the security margin of TinyJAMBU with refined differential and linear cryptanalysis”. In: *IACR Transactions on Symmetric Cryptology* (2020), pp. 152–174.

[Agg+21a] Divesh Aggarwal et al. “Dimension-preserving reductions between SVP and CVP in different p-norms”. In: *SODA*. 2021.

[Agg+21b] Divesh Aggarwal et al. “Fine-grained hardness of CVP(P)— Everything that we can prove (and nothing else)”. In: *SODA*. 2021.

[Ant+21] Antonis Antonopoulos et al. *Faster Algorithms for k-Subset Sum and Variations.* 2021.

[BW21] Karl Bringmann and Philip Wellnitz. “On near-linear-time algorithms for dense subset sum”. In: *Proceedings of the 2021 ACM-SIAM Symposium on Discrete Algorithms (SODA).* SIAM. 2021, pp. 1777–1796.

[Hao+21] Yonglin Hao et al. “Modeling for Three-Subset Division Property without Unknown Subset”. In: *Journal of Cryptology* 34.3 (2021), pp. 1–69.

[JVW21] Ce Jin, Nikhil Vyas, and Ryan Williams. “Fast low-space algorithms for subset sum”. In: *Proceedings of the 2021 ACM-SIAM Symposium on Discrete Algorithms (SODA).* SIAM. 2021, pp. 1757–1776.

[Liu+21] Fukang Liu et al. “Weak Keys in Reduced AEGIS and Tiaoxin”. In: *IACR Trans. Symmetric Cryptol.* 2021.2 (2021), pp. 104–139.

[PST21] Ulrich Pferschy, Joachim Schauer, and Clemens Thielen. “Approximating the product knapsack problem”. In: *Optimization Letters* (2021), pp. 1–12.

[Roh+21] Raghvendra Rohit et al. “Misuse-Free Key-Recovery and Distinguishing Attacks on 7-Round Ascon”. In: *IACR Transactions on Symmetric Cryptology* (2021), pp. 130–155.

[Ten+21] Wil Liam Teng et al. “Cube Attacks on Round-Reduced TinyJAMBU”. In: *Nature Scientific Reports* (2021).

[WH21] Hongjun Wu and Tao Huang. “TinyJAMBU: A Family of Lightweight Authenticated Encryption Algorithms (Version 2)”. In: *Submission to the NIST Lightweight Cryptography Standardization Process (May 2021)* (2021).

[CD22] Wouter Castryck and Thomas Decru. “An efficient key recovery attack on SIDH (preliminary version)”. In: *Cryptology ePrint Archive* (2022).

[CAE] CAESAR. “CAESAR: Competition for authenticated encryption: Security, applicability, and robustness”. In.

[Tea] Keccak Team. *Figures of the components of Keccak*. https://www.keccak.team/figures.html. Accessed: 2022-06-01.

[Sag17] Sage Developers. *SageMath, the Sage Mathematics Software System (Version 7.6)*. https://www.sagemath.org. 2017.

Appendix A

Algorithms Related to Polynomials and Subset-Sum

A.1 Generalizing Jin and Wu's Technique [JW18] to Different Settings

A.1.1 Revisiting [JW18] with Weighted Coefficient

In [JW18, Lemma 5], Jin and Wu established the main lemma which shows that one can compute

$$A(x) :\equiv \prod_{i\in[n]} (1 + x^{a_i}) \mod \langle x^{t+1}, p\rangle \text{ , for any prime } p \in [t+1, (n+t)^3] ,$$

in $\tilde{O}(t)$ time. Further, choosing a *random* p, one can decide nonzeroness of $\mathrm{coef}_{x^t}(A(x))$, with high probability. In this paper, we will work with a more general polynomial

$$G(x) :\equiv \prod_{i\in[n]} (1 + W^b \cdot x^{a_i}) \mod \langle x^{t+1}, p\rangle ,$$

for some integer W, not necessarily 1 and $b \in \mathbb{Z}_{\geq 0}$. Therefore, the details slightly differ. For completeness, we give the details. However, before going into the details, we define some basics of power series and expansion of exp (respectively ln), which will be crucially used in the proof of Lemma A.2. In general, we will be working with primes p such that $\log(p) = O(\log(n+t))$, thus $\log(p)$ terms in the complexity can be subsumed in $\tilde{O}$ notation.

Basic Power series tools. We denote $\mathbb{F}[x]$ as the ring of polynomials over a field $\mathbb{F}$, and $\mathbb{F}[[x]]$ denote the ring of formal power series over $\mathbb{F}$ which has elements of the form $\sum_{i\geq 0} a_i x^i$, for $a_i \in \mathbb{F}$. Two important power series over $\mathbb{Q}[[x]]$ are:

$$\ln(1+x) \;=\; \sum_{k\geq 1} \frac{(-1)^{k-1}x^k}{k}\,, \quad \text{and} \quad \exp(x) \;=\; \sum_{k\geq 0} \frac{x^k}{k!}\,.$$

They are inverse to each other and satisfy the basic properties:

$$\exp\left(\ln\left(1+f(x)\right)\right) = 1 + f(x),$$

$$\ln\left((1+f(x))\cdot(1+g(x))\right) = \ln(1+f(x)) + \ln(1+g(x))\;,$$

for every $f(x), g(x) \in x\mathbb{Q}[x]$ (i.e., constant term is 0). Here is an important lemma to compute $\exp(f(x)) \bmod x^{t+1}$; for details see [Bre76]; for an alternative proof, see [JW18, Lemma 2].

Lemma A.1 ([Bre76])**.** *Given a polynomial $f(x) \in x\mathbb{F}[x]$ of degree at most $t(t < p)$, one can compute a polynomial $g(x) \in \mathbb{F}_p[x]$ in $\tilde{O}(t)$ time such that $g(x) \equiv \exp(f(x)) \mod \langle x^{t+1}, p\rangle$.*

Here is the most important lemma, which is an extension of [JW18, Lemma 4], where the authors considered the simplest form. In this paper, we need the extensions for the 'robust' usage of this lemma (in Section 7.2.1).

Lemma A.2 (Coefficient Extraction Lemma)**.** *Let $A(x) = \prod_{i\in[n]}(1 + W^b \cdot x^{a_i})$, for any non-negative integers a_i, b and $W \in \mathbb{Z}$. Then, for a prime $p > t$, one can compute $\mathrm{coef}_{x^r}(A(x)) \mod p$ for all $0 \leq r \leq t$, in time $\tilde{O}((n + t\log(Wb)))$.*

Proof. Let us define $B(x) := \ln(A(x)) \in \mathbb{Q}[[x]]$. By definition,

$$B(x) = \ln\left(\prod_{i\in[n]}\left(1+W^b\cdot x^{a_i}\right)\right) = \sum_{i\in[n]} \ln\left(1+W^b\cdot x^{a_i}\right)$$
$$= \sum_{i\in[n]}\sum_{j=1}^{\infty} \frac{(-1)^{j-1}}{j}\cdot W^{jb}\cdot x^{a_i j}\,.$$

Let $B_t(x) := B(x) \bmod \langle x^{t+1}, p\rangle$. Define $S_k := \{i \mid a_i = k\}$. Moreover, let us define

$$d_{k,j} := \begin{cases} \sum_{i\in S_k} W^{jb}, & \text{if } S_k \neq \phi\,,\\ 0, & \text{otherwise}\,. \end{cases}$$

Then, by rewriting the above expression, we get

$$B_t(x) \equiv \sum_{i\in[n]}\sum_{j=1}^{\lfloor t/a_i\rfloor} \frac{(-1)^{j-1}}{j}\cdot W^{jb}\cdot x^{a_i j} \equiv \sum_{k\in[t]}\sum_{j=1}^{\lfloor t/k\rfloor}\frac{(-1)^{j-1}\cdot d_{k,j}}{j}\cdot x^{jk} \mod p\,.$$

Since $p > t$, $j^{-1} \bmod p$ exists, for $j \in [t]$. So we pre-compute all $j^{-1} \bmod p$, which takes total $\tilde{O}(t)$ time. Further, we can pre-compute $|S_k|$, $\forall\, k \in [t]$ in $\tilde{O}((n+t))$ time, just by a linear scan.

Moreover, computing each $d_{k,j} = |S_k|\cdot W^{jb}$, takes $\tilde{O}(\log(Wbt))$ time, since $j \leq t$ (assuming we have computed $|S_k|$). Thus, the total time complexity to compute coefficients of $B_t(x)$ is

$$\tilde{O}((n+t)) + \tilde{O}(t) + \sum_{k\in[t]}\sum_{j\in\lfloor t/k\rfloor}\tilde{O}(\log(Wbt)\cdot) = \tilde{O}((n+t\log(Wb)))\,.$$

In the last, we use that $\sum_{k=1}^{\lfloor t/k\rfloor} 1 = O(t\log t)$, which gives the sum to be $\tilde{O}((n + t\log(Wb)))$ ($\log t$ is absorbed inside the $\tilde{O}$).

The last step is to compute $A(x) \equiv \exp(B_t(x)) \bmod \langle x^{t+1}, p\rangle$. Since one can compute $B_t(x)$ in time $\tilde{O}((n + t\log(Wb)))$, using Lemma A.1, one concludes to compute the coefficients of x^r of $A(x)$, $0 \leq r \leq t$, over $\mathbb{F}_p$ in similar time of $\tilde{O}((n +$

$t \log(Wb))$). □

▶ Remark. When $|W| = 1$, it is exactly [JW18, Lemma 5]. One can also work with $A(x) = \prod_{i\in[n]}(1 - W^b \cdot x^{a_i})$; the negative sign does not matter since we can use $\ln(1-x) = -\sum_{i\geq 1} -x^i/i$ and the proof goes through.

A.1.2 Fast Multivariate Polynomial Multiplication

In this section, we will study the time required to compute

$$A(x_1,\ldots,x_k) \;:\equiv\; \prod_{i=1}^{n}\left(1+\prod_{j=1}^{k} x_j^{a_{ij}}\right) \bmod \langle x_1^{t_1+1}, x_2^{t_2+1},\ldots,x_k^{t_k+1}, p\rangle$$

for some prime p. The case when $k = 1$ has been studied in [JW18, Lemma 5] where the authors gave an $\tilde{O}(n + t_1)$ time algorithm for $p \in [t_1 + 1, (n + t_1)^3]$. Here we will present the generalisation of this lemma which is used in Theorem 7.7 using multivariate FFT.

Lemma A.3 (Fast multivariate exponentiation). *Let $\boldsymbol{x} = (x_1,\ldots,x_k)$ and $f(\boldsymbol{x}) = \sum_{i=1}^{t_1} f_i(\boldsymbol{x})\cdot x_1^i \in \mathbb{F}_p[\boldsymbol{x}]$ where $f_i(\boldsymbol{x}) \in \mathbb{F}_p[x_2,\ldots,x_k]$ such that*

1. *$f(\boldsymbol{x}) \bmod \langle x_1,\ldots,x_k\rangle = 0$, i.e., the constant term of $f(\boldsymbol{x})$ is 0, and,*

2. *$\deg_{x_j}(f) = t_j$, for positive integers t_j.*

Then, there is an $\tilde{O}(\prod_{i=1}^{k}(2t_i + 1))$ time deterministic algorithm that computes a polynomial $g(\boldsymbol{x}) \in \mathbb{F}_p[\boldsymbol{x}]$ such that $g(\boldsymbol{x}) \equiv \exp(f(\boldsymbol{x})) \bmod \langle x_1^{t_1+1},\ldots,x_k^{t_k+1}\rangle$ over $\mathbb{F}_p$.

Proof. Let $g(\boldsymbol{x}) = \exp(f(\boldsymbol{x})) = \sum_{i=0}^{\infty} g_i(x_2,\ldots,x_k)\cdot x_1^i$, where $g_i \in \mathbb{F}_p[[x_2,\ldots,x_k]]$. Differentiate wrt x_1 to get:

$$g'(\boldsymbol{x}) \;:=\; \frac{\partial g(\boldsymbol{x})}{\partial x_1} \;=\; g(\boldsymbol{x})\cdot\frac{\partial f(\boldsymbol{x})}{\partial x_1}\;.$$

By comparing the coefficients of x_1^i on both sides, we get (over $\mathbb{F}_p$):

$$g_i \;\equiv\; i^{-1} \cdot \sum_{j=0}^{i-1} f_{i-j} \cdot g_j \mod \langle x_2^{t_2+1}, \ldots, x_k^{t_k+1} \rangle \,,$$

where $g_0 = 1$. By initializing $g_0 = 1$, the rest g_i to 0 and calling $Compute(0, t_1)$ procedure in Algorithm 10, we can compute all the coefficients up to $x_1^{t_1}$, in the polynomial $g(\boldsymbol{x}) \mod \langle x_2^{t_2+1}, \ldots, x_k^{t_k+1} \rangle$, over $\mathbb{F}_p$.

```
Input: integers $\ell, r$ and polynomials $f_i, g_i$
Output: Updated values of $g_i$
if $\ell < r$ then
    $m = \lfloor (\ell + r)/2 \rfloor$;
    $Compute(\ell, m)$;
    for $i \in \{m+1, \ldots, r\}$ do
        $g_i = g_i + i^{-1} \sum_{j=\ell}^{m} (i-j) f_{i-j} g_j \mod \langle x_2^{t_2+1}, x_3^{t_3+1}, \ldots, x_k^{t_k+1}, p \rangle$;
    end
    $Compute(m+1, r)$;
end
return $g_\ell, \ldots, g_r$;
```

Algorithm 10: Algorithm for $Compute(\ell, r)$

To speed up this algorithm, we can set $A(\boldsymbol{x}) = \sum_{i=0}^{r-\ell} i f_i x_1^i$ and $B(\boldsymbol{x}) = \sum_{i=0}^{m-\ell} g_{i+\ell} x_1^i$; here the f_i and $g_{i+\ell}$ have been computed modulo $\langle x_2^{t_2+1}, \ldots, x_k^{t_k+1} \rangle$ already. Use multidimensional FFT [Cor+09, Chapter 30] to compute $C(\boldsymbol{x}) = A(\boldsymbol{x})B(\boldsymbol{x})$ to speed up the for loop which takes $O(\prod_{i=1}^{k}(2t_i+1)\log(\prod_{i=1}^{k}(2t_i+1)))$ time.

Observe that $\sum_{j=\ell}^{m}(i-j)f_{i-j}g_j$ is the coefficient of $x_1^{i-\ell}$ in $C(\boldsymbol{x})$; importantly $\deg_{x_i}(C) \leq 2t_i$, for $i \geq 2$. The extraction of the coefficient of x_1^i in $C(\boldsymbol{x})$ for all i, mod $\langle x_2^{t_2+1}, x_3^{t_3+1}, \ldots, x_k^{t_k+1} \rangle$ can be performed in $O(\prod_{i=1}^{k}(2t_i+1))$ time. This is done by traversing through the polynomial and collecting coefficient along with monomials having the same x_1^i term (and there can be at most $\prod_{i=2}^{k}(2t_i+1)$ many terms). Thus, the total time complexity of computing $g(\boldsymbol{x}) \mod \langle x_2^{t_2+1}, x_3^{t_3+1}, \ldots, x_k^{t_k+1} \rangle$ is

$$T(t_1, t_2, \ldots, t_k) \;=\; 2T(t_1/2, t_2, \ldots, t_k) \;+\; \tilde{O}(\prod_{i=1}^{k}(2t_i+1)) \;=\; \tilde{O}(\prod_{i=1}^{k}(2t_i+1)) \,,$$

as desired. □

Lemma A.4 (Fast logarithm computation)**.** *Let* $A(\boldsymbol{x}) = \prod_{i=1}^{n}\left(1+\prod_{j=1}^{k} x_j^{a_{ij}}\right)$*. Then, there exists an* $\tilde{O}(kn + \prod_{i=1}^{k} t_i)$ *time deterministic algorithm that computes* $\mathrm{coef}_{\boldsymbol{x}^{\boldsymbol{e}}}(\ln(A(\boldsymbol{x}))) \mod p$ *for all* $\boldsymbol{e}$*, such that* $\boldsymbol{e} = (e_1, \ldots, e_k)$ *with* $e_i \leq t_i$*.*

Proof. Let us define $B(\boldsymbol{x}) := \ln(A(\boldsymbol{x}))$. Then,

$$\begin{aligned} B(\boldsymbol{x}) &= \ln\left(\prod_{i=1}^{n}(1+\prod_{j=1}^{k} x_j^{a_{ij}})\right) \\ &= \sum_{i=1}^{n} \ln\left(1+\prod_{j=1}^{k} x_j^{a_{ij}}\right) \\ &= \sum_{i=1}^{n}\sum_{\ell=1}^{\infty}\left(\frac{(-1)^{\ell-1}}{\ell}(\prod_{j=1}^{k} x_j^{a_{ij}})^{\ell}\right) . \end{aligned}$$

Without loss of generality, we can assume that $t_1 \leq t_i, \forall i > 1$. Let $C(\boldsymbol{x}) := B(\boldsymbol{x}) \mod \langle x_1^{t_1+1}, \ldots, x_k^{t_k+1}, p\rangle$. Since, we are interested where the individual degree of x_j can be at most t_j, the index ℓ in the above equation (for a fixed i) must satisfy $a_{ij} \cdot \ell \leq t_j$ for each $j \in [k]$. This implies $\ell \leq t_j/a_{ij}$, for $j \in [k]$. Therefore, define $M_i := min_{j=1}^{k}\lfloor t_j/a_{ij}\rfloor$. Now, one can express $C(\boldsymbol{x})$ using M_i since it suffices to look at the index ℓ till M_i (for a fixed i), as argued before.

Importantly, note that the above equation involves $\prod_{j=1}^{k} x_j^{a_{ij}\ell}$ which has individual degree > 0, since both $a_{ij}, \ell \geq 1$. Thus, define $T := \{\boldsymbol{e} = (e_1, \ldots, e_k) \in \mathbb{Z}^k \mid 1 \leq e_i \leq t_i, \forall i \in [k]\}$. Then,

$$\begin{aligned} C(\boldsymbol{x}) &= \sum_{i=1}^{n}\sum_{\ell=1}^{M_i}\left(\frac{(-1)^{\ell-1}}{\ell}(\prod_{j=1}^{k} x_j^{a_{ij}})^{\ell}\right) \\ &= \sum_{\overline{e}\in T}\sum_{\ell=1}^{t_1/e_1}\left(\frac{s_{\overline{e}} \times (-1)^{\ell-1}}{\ell}\prod_{j=1}^{k} x_j^{e_i\ell}\right) , \end{aligned}$$

where $s_{\overline{e}} = |\{i \in [n] \,|\, (a_{i1}, \ldots, a_{ik}) = \overline{e}\}|$. Essentially, for a given $s_{\boldsymbol{e}}$, the quantity computes how many times a_{ij} is equal to e_j, for all $j \in [k]$. Using $s_{\boldsymbol{e}}$, we can interchange the order of the summation as shown above. Moreover, we can precompute $s_{\boldsymbol{e}}$, for all $\boldsymbol{e} \in T$ in time $O\left(kn + \prod_{i=1}^{k} t_i\right)$.

Observe that $\mathrm{coef}_{\boldsymbol{x}^{\boldsymbol{e}}}(B(\boldsymbol{x})) = \mathrm{coef}_{\boldsymbol{x}^{\boldsymbol{e}}}(C(\boldsymbol{x}))$, for any $(e_1, \ldots, e_k) \in T$. Since, $\ell \le t_1 < p$, ℓ^{-1} exists and can be pre-computed in $\tilde{O}(t_1)$.

Time complexity. Observe that we have

$$\begin{aligned} C(\boldsymbol{x}) &= \sum_{\overline{e} \in T} \sum_{\ell=1}^{t_1/e_1} \left(\frac{s_{\overline{e}} \times (-1)^{\ell-1}}{\ell} \prod_{j=1}^{k} x_j^{e_i \ell} \right) \\ &= \sum_{e_2=1}^{t_2} \sum_{e_3=1}^{t_3} \cdots \sum_{e_k=1}^{t_k} \left(\sum_{e_1=1}^{t_1} \sum_{\ell=1}^{t_1/e_1} \left(\frac{s_{\overline{e}} \times (-1)^{\ell-1}}{\ell} \prod_{j=1}^{k} x_j^{e_i \ell} \right) \right) \end{aligned}$$

The time taken to compute all $\mathrm{coef}_{\boldsymbol{x}^{\boldsymbol{e}}}(C(\boldsymbol{x}))$, given $s_{\boldsymbol{e}}$, is the number of iterations over all $(e_2, \ldots, e_k)$, for $1 \le e_i \le t_i$, $i > 1$ and $\ell \in [t/e_1]$, which is atmost $\sum_{j=1}^{t_1} \lfloor t_1/j \rfloor \times \prod_{i=2}^{k} t_i = \tilde{O}(\prod_{i=1}^{k} t_i)$, since $\sum_{j=1}^{t_1} t_1/j = O(t_1 \log t_1)$. Thus, the total time is $\tilde{O}(kn + \prod_{i=1}^{k} t_i)$. $\square$

A.1.3 Solving Linear Recurrence: Tool for Section 7.2.1

In this section, we briefly sketch how to speed up the algorithm of computing E_i, for $i \in [m]$, using FFT, rather than just going through one by one. Equation (7.1) gives the following relation:

$$E_j \equiv j^{-1} \cdot \left(\sum_{i \in [j]} (-1)^{j-i-1} E_i \cdot P_{j-i} \right) \mod q \; .$$

Here, by E_j (respectively P_j), we mean $E_j(\mu^{w_1}, \ldots, \mu^{w_\ell})$ (respectively P_j). We can assume that P_j's are already pre-computed and hence contribute to the complexity only once. This calculation is very similar to [JW18, Lemma 2], with a similar relation. But we give the details, for completeness.

Eventually, once we have computed P_j's, we can use FFT (Algorithm 11) to find E_j's, which eventually gives $T(m) \le \tilde{O}(k(n+t))$.

To elaborate, in the for-loop 7-8 in Algorithm 11, we want to find $\sum_{i=\ell}^{s} (-1)^{j-i} E_i \cdot$

```
Input: $P_i$, for $i \in [m]$, $q$ and $E_0 = 1$
Output: $E_i$ for $i \in [m]$
Initialize $E_j \leftarrow 0$, for $j \in [m]$;
return Compute$(0, m)$;
Procedure Compute$(\ell, u)$ $\triangleright$ the values returned by Compute$(\ell, u)$ are the
  final values $E_\ell, \ldots, E_u$ are computed;
for $\ell < u$ do
    $s \leftarrow \lfloor \frac{\ell+u}{2} \rfloor$
    Compute$(\ell, s)$;
    for $j \leftarrow s+1, \ldots, u$ do
        $E_j \leftarrow E_j + j^{-1} \cdot (\sum_{i=\ell}^{s} (-1)^{j-i} E_i \cdot P_{j-i}) \mod q$;
    end
    Compute$(s+1, u)$
end
return $E_\ell, \ldots, E_u$;
```

Algorithm 11: Algorithm for computing E_i

P_{j-i} for all $j \in \{s+1, \ldots, u\}$. To achieve this, we define the polynomials:

$$F(x) := \sum_{k=0}^{u-\ell} (-1)^{k-1} P_k x^k, \quad \text{and} \quad G(x) := \sum_{j=0}^{s-\ell} E_{j+\ell} x^j \ .$$

Note that our $F(x)$ is *different* than used in [JW18], because of a slightly different recurrence relation. We can compute $H(x) = F(x) \cdot G(x)$, in time $\tilde{O}((u-\ell))$. Observe that $\sum_{i=\ell}^{u} (-1)^{j-i-1} P_{j-i} \cdot E_i = \text{coef}_{x^{j-\ell}}(H(x))$ because $(-1)^{j-i-1} P_{j-i} = \text{coef}_{x^{j-i}}(F(x))$ and $E_i = \text{coef}_{x^{i-\ell}}(G(x))$. Therefore, the inner for loop can be computed in $\tilde{O}((u-\ell))$ time.

Final time complexity. Let $T'(m)$ is the complexity of computing $E_1, \ldots, E_m$ assuming precomputations of P_j and j^{-1}. Then,

$$T'(m) \ \leq \ 2T'(m/2) + \tilde{O}(m) \ \implies \ T'(m) \leq \tilde{O}(m) \ .$$

Therefore, the total complexity of computing $E_1, \ldots, E_m$, is $T(m) = T'(m) + \tilde{O}(k(n+t))$, where $\tilde{O}(k(n+t))$ is for the time for computing P_j's (and j^{-1}). Since, $q = O(n+k+t)$ and $m \leq k$, we get $T(m) = \tilde{O}(k(n+t))$, as we wanted.

A.2 Algorithms

A.2.1 Trivial Solution for $k-\mathsf{SSSUM}$

Bellman's dynamic programming solution for the decision version of SSUM is based on the recurrence relation $S((a_1,\ldots,a_n),t) = S((a_1,\ldots,a_{n-1}),t) \oplus S((a_1,\ldots,a_{n-1}),t-a_n)$ where $S((a_1,\ldots,a_j),t') = 1 \iff t'$ is a realisable target of $(a_1,\ldots,a_j)$. Using this relation, the algorithm needs to store only the values of $S((a_1,\ldots,a_{j-1}),t')$ for all $1 \leq t' \leq t$ to compute $S((a_1,\ldots,a_j),t'')$ for all $1 \leq t'' \leq t$. So, the time complexity is $O(nt)$ whereas the space complexity is $\Omega(t)$.

To find all the solutions, we modify the above algorithm by adding a pointer from $S((a_1,\ldots,a_j),t')$ to $S((a_1,\ldots,a_{j-1}),t-a_j)$ when both are equal to 1. The same is done for $S((a_1,\ldots,a_j),t')$ and $S((a_1,\ldots,a_{j-1}),t)$. Apart from these, we also add a pointer from $S(a_i,a_i)$ to a new node $S(\{\},0)$ where $1 \leq i \leq n$. This gives a directed graph of size $O(nt)$ because the out-degree of each node is at most 2. To find all the solutions to the SSUM , we simply run a modified version of DFS algorithm[1] on this graph to finds all the paths from $S((a_1,\ldots,a_n),t)$ to $S(\{\},0)$.

It is evident that if the number of solutions to the SSUM instance is k, then the number of paths is also k. The modified DFS algorithm goes through all the neighbouring vertices of a given vertex, no matter if they are visited or not. Furthermore, any path that starts from $S((a_1,\ldots,a_n),t)$ will end at $S(\{\},0)$.

Clearly, this algorithm will terminate because the graph is directed acyclic. The running time and space of the modified DFS algorithm is $O(nk)$ because each path is of length at most n and the algorithm traverses through each path at most twice (the first traversal ends at $S(\{\},0)$ which finds the path and the second one is backtracking). Therefore, the total time and space complexity is $O(n(t+k))$.

[1]The graph is a directed acyclic one and we can use the algorithm mentioned in https://stackoverflow.com/questions/20262712/enumerating-all-paths-in-a-directed-acyclic-graph

A.2.2 Trivial Dynamic Algorithm for SimulSubsetSum

In this section, we sketch a dynamic pseudo-polynomial time algorithm which solves SimulSubsetSum, with targets $t_1, \ldots, t_k$, in $O(n(t_1+1)\ldots(t_k+1))$ time. This is a direct generalisation of Bellman's work [Bel57].

The algorithm considers an $n \times (t_1+1) \times \cdots \times (t_k+1)$ boolean matrix M and populates it with 0/1 entries. $M[i, j_1, j_2, \ldots, j_k]$ has 1 iff the SimulSubsetSum instance with ℓ^{th} SSUM instance $(a_{1\ell}, a_{2\ell}, \ldots, a_{i\ell}, j_i)$ has a solution. Here $i \in [n]$ and $j_i \in [0, t_i]$. Even though we have remarked that wlog $t_i \geq 1, \forall i \in [n]$, we cannot do the same for a_{ij}'s. This forces us to look at $j_i \in [0, t_i], \forall i \in [k]$. The algorithm starts by setting $M[1, a_{11}, a_{12}, \ldots, a_{1k}] = 1$ and $M[1, j_1, j_2, \ldots, j_k] = 0$ for the rest. Then, using the following recurrence relation, the algorithm populates the rest of the matrix.

$$M[i, j_1, j_2, \ldots, j_k] = M[i-1, j_1, j_2, \ldots, j_k] + M[i-1, j_1 - a_{i1}, j_2 - a_{i2}, \ldots, j_k - a_{ik}]$$

i.e., $M[i, j_1, j_2, \ldots, j_k]$ is set to 1 iff either $M[i-1, j_1, j_2, \ldots, j_k] = 1$ or $M[i-1, j_1 - a_{i1}, j_2 - a_{i2}, \ldots, j_k - a_{ik}] = 1$. Since, the size of the matrix is $n(t_1+1)\ldots(t_k+1)$, the running time of the algorithm is $O(n(t_1+1)\ldots(t_k+1))$.

A.2.3 Dynamic Programming Approach for Subset Product

In this section, we will briefly discuss the modification to Bellman's dynamic programming approach for SSUM to solve Subset Product in deterministic (expected) time $O(nt^{o(1)})$.

The algorithm starts by removing all a_i that does not divide t. Then using the factoring algorithm in [LP92], we can factor t into prime factor p_j, i.e., $t = \prod_{i\in[k]} p_j^{t_j} = t$, where $k = O(\log(t)/\log\log(t))$. We now compute the DP table T of size $n \times (t_1 + 1) \times \cdots \times (t_k + 1)$ such that

$$T[i, x_1, \ldots, x_k] = 1, \text{ if and only if there exists } S \in [i], \text{ such that } \prod_{j \in S} a_j = \prod_{j \in [k]} p_j^{x_j} .$$

Observe that the time complexity of the algorithm is the time taken to populate the DP table with either 1 or 0. Since the size of the DP table is $n \times \prod_{i \in [k]}(1 + t_i)$, using the similar analyse mentioned in Section 7.2.2, we can bound the term $\prod_{i \in [k]}(1 + t_i)$ by $t^{o(1)}$. Therefore, the total time complexity is $O(nt^{o(1)})$.