FlipFields—New Building Blocks for Cryptograpic Primitives?

Christopher Wolf
chris/at/Christopher-Wolf.de
Version 1.15, 2026-09-19

Cryptology ePrint Archive, Paper 2026/1088
https://eprint.iacr.org/2026/1088

Code: https://github.com/cbwolf-mq/flipFields

Abstract

$1 + 1 \equiv 0$—while looking strange at first glance, this is certainly true in GF(2). In this paper we propose the two field-like structures $\text{d}\text{i}\text{l}\text{Ⅎ}\mathit{Int}\dot{s}$ and $\text{d}\text{i}\text{l}\text{Ⅎ}\breve{P}\mathit{olys}$ as potential alternative building block for crypographic schemes—in particular in the post-quantum setting. Both structures have 2^d elements for some positive integer d and are derived from the natural numbers $\mathbb{N}$ in the first case and the univariate polynomial ring GF(2)[t] in the second case. We call the generalization of these two structures $\text{d}\text{i}\text{l}\text{Ⅎ}\curvearrowleft\mathit{Field}$s. In addition, we see examples how they can be used for post-quantum cryptography, in particular Unbalanced Oil and Vinegar, Learning with Errors and Saber. There is also a discussion of cryptographic primitives that are most likely not suitable for $\text{d}\text{i}\text{l}\text{Ⅎ}\curvearrowleft\mathit{Field}$s or at least not easy to tweak. As the structures are very new, this paper also includes a list of uəd_{O} P̆roblems.

Contents

1 Introduction 3

2 Basics of $\text{d}\text{i}\text{l}\text{Ⅎ}\curvearrowleft\mathit{Field}$s 4
- 2.1 $\text{d}\text{i}\text{l}\text{Ⅎ}\mathit{Int}\dot{s}$ . . . 4
- 2.2 $\text{d}\text{i}\text{l}\text{Ⅎ}\breve{P}\mathit{olys}$ . . . 5
- 2.3 Cryptographical Motivation . . . 5
- 2.4 Abstract $\text{d}\text{i}\text{l}\text{Ⅎ}\curvearrowleft\mathit{Field}$s and Definitions . . . 6

3 Applications to Cryptology 8
- 3.1 Unbalanced Oil and Vinegar . . . 8
- 3.2 Learning With Errors . . . 9
- 3.3 Saber . . . 9
- 3.4 Misceangelous Cryptographic Primitives . . . 11
  - 3.4.1 McEliece encryption scheme . . . 11
  - 3.4.2 NTRU . . . 12
  - 3.4.3 ML-KEM . . . 13
  - 3.4.4 S-Boxes . . . 13
  - 3.4.5 Trivium . . . 15
  - 3.4.6 Hidden Subgroup Problem . . . 17

4 Mathematical Algorithms 19
- 4.1 Hensel Lifting . . . 19
- 4.2 Gaussian Elimination Heuristic . . . 22
- 4.3 Number of Solutions for $Ax = b$ . . . 24
- 4.4 SubstSolver . . . 28
- 4.5 Trimmed LU form . . . 31

4.6 Linear Feedback Shift Registers (LFSR) . . . 33
4.7 Non-Linear Feedback Shift Registers (NLFSR) . . . 36
4.8 Ring mod $X^N + 1$ . . . 40
4.9 Polynomial Multiplication . . . 40
4.10 Miscellaneous . . . 44

5 Security Reductions 48
5.1 $\mathcal{M}$ultivariate $\mathcal{Q}$uadratic Equations . . . 48
5.2 Reducing $\mathcal{MQ}$-GF(2) to $\mathcal{MQ} - \tilde{\mathbb{N}}$ . . . 49

6 Generalizations 56
6.1 General Basis b . . . 56
6.2 Matrices . . . 56
6.3 Polynomials . . . 56
6.4 Extension Fields . . . 56
6.4.1 Inner$\odot$dịlƎ↶Fields . . . 57
6.5 Complex numbers and Quaternions . . . 57

7 Concrete Implementations 59
7.1 Motivation . . . 59
7.2 Python Implementation . . . 59
7.3 Symbolic Computation . . . 60

8 Discussion / Conclusions / **uədO P̆roblems** 62

References 64

A Matrix solving—Depreciated Methods 67
A.1 Generalized LU Decomposition . . . 67
A.2 Bespoke LU Decomposition . . . 69
A.3 Empirics on Solving Affine Equations $Ax = b$ . . . 70

B Matrix Operations—Empirical Findings 77
B.1 Gaussian Elimination Heuristic . . . 77
B.2 Generatlized LU Decomposition . . . 80
B.3 General Solving of Affine Equations $Ax = b$ . . . 83

C Implementing Linear Feedback Shift Registers (LFSR) 87
C.1 LFSR—Maximal cycle length over dịlƎ$\dot{Int}s$ . . . 87
C.2 LFSR—Maximal cycle length over dịlƎ$\breve{P}olys$ . . . 93
C.3 LFSR—States over dịlƎ$\dot{Int}s$ . . . 98
C.4 LFSR—States over dịlƎ$\breve{P}olys$ . . . 99

D Non-Linear Feedback Shift Registers (NLFSR) 102
D.1 Genetic algorithm for finding NLFSR with large cycle length . . . 102
D.2 NLFSR—Affine Polynomials . . . 102
D.3 NLFSR—HigherDegree Polynomials . . . 113

E How to write dịlƎ↶$Field$ 125

F Change Log 126

1 Introduction

In cryptology, it is usually prudent not to put all eggs into one basket. Following a proverb, we propose some new eggs for cryptographic primitives to put into different baskets. The overall structure called $\mathrm{d}\underset{\cdot}{\mathrm{I}}\underset{\cdot}{\mathrm{I}}\mathrm{F}\curvearrowleft Fields$ is somehow in between a finite integer ring mod 2^d and the finite field $\mathrm{GF}(2^d)$ for some positive integer $d\mathbb{N}$. With $\mathbb{N}$ we denote all non-negative integers, *i.e.* $\mathbb{N} := \{0, 1, \ldots\}$. In this news structure, we use that multiplication is closed for odd numbers. Therefore, we observe

$$(2a + 1)(2b + 1) = 2(2ab + a + b) + 1\,.$$

So while working in the corresponding multiplicative group is easy, it is difficult to keep all numbers in the additive group odd. We start with the observation

$$(2a + 1) + (2b + 1) + (2c + 1) = 2(a + b + c + 1) + 1 \;\forall a, b, c \in \mathbb{N}$$

To make sure all addition does not produce even numbers, we replace the regular addition by a trinary-tally.

All in all, the aim of this paper is to identify potential cryptographic primitives that can be used in the post-quantum context. In particular, this includes the $\mathcal{M}$ultivariate $\mathcal{Q}$uadratic signature scheme Unbalanced Oil and Vinegar (UOV) in Sect. 3.1, Learning with Errors in Sect. 3.2 and also the key-exchange scheme Saber in Sect. 3.3. All three are elaborated in in Sect. 4 when diving deeper into related algorithms.

This paper is organized as follows: We start by considering $\mathrm{d}\underset{\cdot}{\mathrm{I}}\underset{\cdot}{\mathrm{I}}\mathrm{F}\curvearrowleft Fields$, *i.e.* a field-like structure. However, all results are also true for $\mathrm{d}\underset{\cdot}{\mathrm{I}}\underset{\cdot}{\mathrm{I}}\breve{P}olys$, *i.e.*a similar structure over polynomial rings. We mainly concentrate on $\mathrm{d}\underset{\cdot}{\mathrm{I}}\underset{\cdot}{\mathrm{I}}\dot{Int}s$ for easier explanation, cf Sect. 1 for an example. Hence, a formal treatment of $\mathrm{d}\underset{\cdot}{\mathrm{I}}\underset{\cdot}{\mathrm{I}}\breve{P}olys$ is postponed to Sect. 2.2. An axiomatic treatment of $\mathrm{d}\underset{\cdot}{\mathrm{I}}\underset{\cdot}{\mathrm{I}}\mathrm{F}\curvearrowleft Fields$ is given in Sect. 2.4. Following a cryptological motivation in Sect. 2.3, we outline several cryptographic schemes, in particular the $\mathcal{M}$ultivariate $\mathcal{Q}$uadratic signature scheme Unbalanced Oil and Vinegar (UOV) Learning with Errors in Sect. 3.2 and also Saber in Sect. 3.3. After that, we discuss some miscellaneous cryptographic primitives in Sect. 3.4. Based in these cryptologic insights, we concentrate on concrete algorithms, in particular Gaussian Elimination (Sect. 4.2). Only the last can be used to solve affine equations. On a more practical level, we have the Hensel Lifting to compute multiplicative inverses (Sect. 4.1) and Gaussian Elimination (Sect. 4.2). Moving to symmetric cryptographic schemes, we have a treatment of Linear Feedback Shift Registers in Sect. 4.6. The non-linear case is treated in Sect. 4.7. This is followed by the very important ring $X^N + 1$ in Sect. 4.8 and polynomial multiplication in Sect. 4.9. Possible generalizations of $\mathrm{d}\underset{\cdot}{\mathrm{I}}\underset{\cdot}{\mathrm{I}}\mathrm{F}\curvearrowleft Fields$ are given in Sect. 6. Insights of the corresponding implementation are given in Sect. 7. The paper concludes with Sect. 8, which also includes a list of $\mathfrak{O}$pen $\breve{\mathrm{P}}$roblems. The Appendix gives more details on concrete empirical findings, in particular on matrix operations in App. A, linear and non-linear feedback shift registers (app. B and C, respectively). A change log is provided in App F.

2 Basics of $\mathsf{d}_{\mathrm{i}}\mathsf{I}\!\!\mathsf{F}\curvearrowleft Fields$

While both $\mathsf{d}_{\mathrm{i}}\mathsf{I}\!\!\mathsf{F}\mathit{Int\dot{s}}$ and $\mathsf{d}_{\mathrm{i}}\mathsf{I}\!\!\mathsf{F}\breve{P}olys$ are specializations of the overall idea of $\mathsf{d}_{\mathrm{i}}\mathsf{I}\!\!\mathsf{F}\curvearrowleft Fields$, we concentrate on the integer case first for easier explanation. Where it makes sense, we we abstract to $\mathsf{d}_{\mathrm{i}}\mathsf{I}\!\!\mathsf{F}\curvearrowleft Fields$ or use $\mathsf{d}_{\mathrm{i}}\mathsf{I}\!\!\mathsf{F}\breve{P}olys$ for illustration purposes.

2.1 $\mathsf{d}_{\mathrm{i}}\mathsf{I}\!\!\mathsf{F}\mathit{Int\dot{s}}$

We first define the set of elements on which we will define our new structure.

Definition 1. *Let* $\tilde{\mathbb{N}}^d := \{k \in \mathbb{N} : 0 < k < 2^{d+1}, 2 \nmid k\}$ *the* ***integer carrier*** *of dimension* d.

Note $|\tilde{\mathbb{N}}^d| = 2^d$. For $\mathsf{d}_{\mathrm{i}}\mathsf{I}\!\!\mathsf{F}\mathit{Int\dot{s}}$ over the integers, we will carry over multiplication, but define addition different to integer rings mod 2^{d+1}. A full list of properties of the $\mathsf{d}_{\mathrm{i}}\mathsf{I}\!\!\mathsf{F}\mathit{Int\dot{s}}$ is given below. For shorthand, we write $\tilde{\mathbb{N}} := \tilde{\mathbb{N}}^d$ and use it for the $\mathsf{d}_{\mathrm{i}}\mathsf{I}\!\!\mathsf{F}\mathit{Int\dot{s}}$ of dimension d, if this is clear from the context.

Definition 2. *Let* $a_i \in \tilde{\mathbb{N}}$ *be for some odd* $k \in \mathbb{N}$ *and* $i \in \mathbb{N} : 1 \leq i \leq k$. *Then we define the* ***tally operator*** *as* $[a_1, \ldots, a_k]_k := []_{i=1}^k a_i := \sum_{i=1}^k a_i \pmod{2^{d+1}}$.
Shorthand, we write $[a, b, c] := [a, b, c]_3$ *for some* $a, b, c \in \tilde{\mathbb{N}}$.

Lemma 1. *The tally operator is closed,* i.e. *we have*

$$[a_1, \ldots, a_k]_k \in \tilde{\mathbb{N}} \ \forall k \ : \ a_1, \ldots, a_k \in \tilde{\mathbb{N}}.$$

Proof. We use an inductive proof. The cases $k = 1, 3$ are clear, in particular see the example in Sect. 1 for $k = 3$. For $k > 3$ we have $[a_1, \ldots, a_k]_k = [[a_1, a_2, a_3], a_4, \ldots, a_k]_{k-2}$. Using the induction hypothesis completes the proof. □

Multiplicative group. We just saw that the multiplicative group is closed mod 2^{d+1} for elements from the integer carrier $\tilde{\mathbb{N}}^d$. Associativity, commutativity, and the multiplicative neutral are inherited from the corresponding integer ring.

In addition, as $\gcd(a, 2) = 1 \ \forall a \in \tilde{\mathbb{N}}^d$, we also have a multiplicative inverse for all elements of the integer carrier of dimension d. Hence, define a^{-1} as this multiplicative inverse such that $a.a^{-1} = 1 \bmod 2^{d+1}$. For example, it is possible to compute a^{-1} for given $a, 2^{d+1}$ using the Extended Euclidean Algorithm or the Hensel Lifting, cf. Sect. 4.1 . As the multiplicative group over the integer carrier coincides with the multiplicative group over the integers mod 2^{d+1}, such a $b \in \tilde{\mathbb{N}}$ always exists and are unique.

Tally operator. In the case of the tally operator, we have to adapt our definition a bit, as there is no additive neutral, *i.e.* 0 over the integer ring. However, associativity and commutativity are inherited from the integer ring. Alas, there is no direct additive inverse as the tally operator is trinary, not binary. We capture this with the following definition.

$\mathsf{d}_{\mathrm{i}}\mathsf{I}\!\!\mathsf{F}\mathit{Int\dot{s}}$. So all in all, the structure we are looking for is very similar to a finite field over the integers. In particular, our construction ensures that all elements have both a multiplicative and additive inverse—on cost of an additive neutral. We call this an $\mathsf{d}_{\mathrm{i}}\mathsf{I}\!\!\mathsf{F}\mathit{Int\dot{s}}$ with dimension d. To properly build up an $\mathsf{d}_{\mathrm{i}}\mathsf{I}\!\!\mathsf{F}\mathit{Int\dot{s}}$ we start with the properties of the trinary-additive group **FlipTrix**, see Table 1. The abstract definition for $\mathsf{d}_{\mathrm{i}}\mathsf{I}\!\!\mathsf{F}\curvearrowleft Fields$ is given in Sect. 2.4.
The properties of **FlipTrix** are also required for general $\mathsf{d}_{\mathrm{i}}\mathsf{I}\!\!\mathsf{F}\curvearrowleft Fields$ and the special case $\mathsf{d}_{\mathrm{i}}\mathsf{I}\!\!\mathsf{F}\breve{P}olys$.

Table 1: Properties of the **FlipTrix** for $\mathrm{d}\underset{\cdot}{\mathrm{I}}\underset{\cdot}{\mathrm{I}}\mathit{Int\dot{s}}$

Associativity. This is inherited from the integer ring, *e.g.* we have $[a, b, [c, d, e]] = [a, [b, c, d], e] = [[a, b, c], d, e]$.

Closure. As we saw in Lem. 1, the tally operator is closed. The definition of a tally and also the proof and closure carry through to $\mathrm{d}\underset{\cdot}{\mathrm{I}}\underset{\cdot}{\mathrm{I}}\breve{P}olys$ and also general $\mathrm{d}\underset{\cdot}{\mathrm{I}}\underset{\cdot}{\mathrm{I}}\curvearrowleft Field$s. By construction, the tally operator ensures closure within the corresponding structure.

Additive Neutral and Inverse. As zero does not exist in $\mathrm{d}\underset{\cdot}{\mathrm{I}}\underset{\cdot}{\mathrm{I}}\mathit{Int\dot{s}}$, we use $[a, b, -b] = a$ instead, $\forall a, b \in \tilde{\mathbb{N}}^d$.

Commutativity. This is also inherited from the integer ring *e.g.* $[a, b, c] = [a, c, b]$.

Distributive Law. In addition to the tri-add group laws, we also have $a \cdot [b, c, d] = [a \cdot b, a \cdot c, a \cdot d]$ $\forall a, b, c, d \in \tilde{\mathbb{N}}^d$.

2.2 $\mathrm{d}\underset{\cdot}{\mathrm{I}}\underset{\cdot}{\mathrm{I}}\breve{P}olys$

In a similar fashon to $\mathrm{d}\underset{\cdot}{\mathrm{I}}\underset{\cdot}{\mathrm{I}}\mathit{Int\dot{s}}$, we can consider polynomials over GF(2) that all have the constant term "+". Let $\tilde{\mathbb{P}}^d$ be defined over the ring $\mathrm{GF}(2)[x]$ with polynomials with a constant fixed term "+1", *i.e.*

$$\tilde{\mathbb{P}}^n := \{p \in \mathrm{GF}(2)[x] : \deg(p) \leq n, x \nmid p\}$$

and denote this as $\mathrm{d}\underset{\cdot}{\mathrm{I}}\underset{\cdot}{\mathrm{I}}\breve{P}olys$ of dimension d. The idea is similar to the $\mathrm{d}\underset{\cdot}{\mathrm{I}}\underset{\cdot}{\mathrm{I}}\mathit{Int\dot{s}}$ $\tilde{\mathbb{N}}^d$ while addition is adjusted accordingly. For the remainder of this paper, we mainly concentrate on $\mathrm{d}\underset{\cdot}{\mathrm{I}}\underset{\cdot}{\mathrm{I}}\mathit{Int\dot{s}}$ but state when we deal with $\mathrm{d}\underset{\cdot}{\mathrm{I}}\underset{\cdot}{\mathrm{I}}\breve{P}olys$ instead.

The abstraction of $\mathrm{d}\underset{\cdot}{\mathrm{I}}\underset{\cdot}{\mathrm{I}}\mathit{Int\dot{s}}$ and $\mathrm{d}\underset{\cdot}{\mathrm{I}}\underset{\cdot}{\mathrm{I}}\breve{P}olys$ is called an $\mathbf{d}\underset{\cdot}{\mathbf{I}}\underset{\cdot}{\mathbf{I}}\curvearrowleft Field$. Further generalizations can be found in Sect. 6.

2.3 Cryptographical Motivation

In a nutshell, we replace the finite field structure or the finite ring mod 2^{d+1} in cryptographic schemes with an $\mathrm{d}\underset{\cdot}{\mathrm{I}}\underset{\cdot}{\mathrm{I}}\curvearrowleft Field$, *i.e.* an $\mathrm{d}\underset{\cdot}{\mathrm{I}}\underset{\cdot}{\mathrm{I}}\mathit{Int\dot{s}}$ or $\mathrm{d}\underset{\cdot}{\mathrm{I}}\underset{\cdot}{\mathrm{I}}\breve{P}olys$, accordingly. While this looks easy at first glance, we actually need a carefully inspect each step of the corresponding algorithm, such as key and signature generation, signature verifiction, de- and encryption, respectively. See Sect. 3 for further details on concrete schemes. In particular, the missing zero and the trinary tally—rather than a binary addition—cause ripple effects that need to be taken into account carefully. To capture those, all key components for the following schemes have been implemented in Python, see Sect. 7.2 for a quick overview or throughout this paper.

When discussion the effect of $\mathrm{d}\underset{\cdot}{\mathrm{I}}\underset{\cdot}{\mathrm{I}}\curvearrowleft Field$s on the efficiency of a cryptographic scheme, we heavily rely on the underlying system. We choose two special points for comparision, denoted HW and SW. First, we assume a hardware implementation (HW) that mainly counts the number of gates and the latency. Second, we assume a software implementation (SW) where we have plenty of build-in commands and are mainly concerned about timings. For ease of explanation, we neither allow threading nor multi-core architecture. Neither do we take energy efficiency into account. Using our observations so far, we can order the integer ring $\mathcal{I}$ mod 2^d, the finite field $\mathbb{F} := \mathrm{GF}(2^d)$, the $\mathrm{d}\underset{\cdot}{\mathrm{I}}\underset{\cdot}{\mathrm{I}}\mathit{Int\dot{s}}$ $\tilde{\mathbb{N}}^d$ and the $\mathrm{d}\underset{\cdot}{\mathrm{I}}\underset{\cdot}{\mathrm{I}}\breve{P}olys$ $\tilde{\mathbb{P}}^d$ the following way both for SW and HW. Our first odering is done via multiplication, the second one regarding addition. In all cases, we

use some integer $d \in \mathbb{N} : 1 \leq d$. The ordering regarding multiplication can be found in Table 2. The corresponding information regarding addition is given in Table 3.

Table 2: Ordering regarding multiplication

↑ efficiency	$\tilde{\mathbb{P}}^d$	$\mathrm{d}\underset{\cdot}{\mathrm{I}}\underset{\cdot}{\mathrm{I}}\breve{P}olys$ of dimension d
	$\mathcal{I}$, $\tilde{\mathbb{N}}^d$	integer ring mod 2^d, $\mathrm{d}\underset{\cdot}{\mathrm{I}}\underset{\cdot}{\mathrm{I}}\dot{Int}s$ of dimension d
↓ information diffusion	$\mathbb{F}$	finite field $\mathrm{GF}(2^d)$

T-functions. To better understand the point of *information diffusion*, we consider T-functions [KS02] over integer rings mod 2^k. Let i be some integer with $0 \leq i < k$. We call a function a T-function, if each bit b_i only changes as a function of all bits that are from the same level or lower, *i.e.* the bits $b_0, \ldots, b_i$. For this reason, these functions have been called *triangular* or T-functions for short. Following this definition, addition is a T-function for $\mathrm{d}\underset{\cdot}{\mathrm{I}}\underset{\cdot}{\mathrm{I}}\dot{Int}s$ $\tilde{\mathbb{N}}^d$, $\mathrm{d}\underset{\cdot}{\mathrm{I}}\underset{\cdot}{\mathrm{I}}\breve{P}olys$ $\tilde{\mathbb{P}}^d$, integer rings $\mathcal{I}$ and finte fields $\mathbb{F}$. The implications are shown in Table 3. However, multiplication is a T-function only in the structures $\tilde{\mathbb{N}}^d, \tilde{\mathbb{P}}^d$ and $\mathcal{I}$. The ramifications can be found in Table 2. In a nutshell, an $\mathrm{d}\underset{\cdot}{\mathrm{I}}\underset{\cdot}{\mathrm{I}}\dot{Int}s$ of dimension d is somehow in the middle between the integer ring $\mathcal{I}$ mod 2^d and the finite field $\mathbb{F} := \mathrm{GF}(2^d)$, also when it comes to information diffusion. In the ring I, multiplication is a T-function, in the field $\mathbb{F}$ it is not, which allows for faster diffusion in the latter. As we know from symmetric cryptography, high diffusion can lead to smaller, but still secure parameters. Here, division / inverse elements could enhance diffusion. Secondly, the scheme might use finite field operations over $\mathbb{F}$. In this case, $\mathrm{d}\underset{\cdot}{\mathrm{I}}\underset{\cdot}{\mathrm{I}}\dot{Int}s$ might be beneficial as their modulo operation is 2^k rather than a primitive polynomial of degree k. Hence, we expect them to yield faster implementations of the corresponding cryptographic primitive while still maintaning the same level of security.

Table 3: Ordering regarding addition

↑ efficiency	$\mathcal{I}$, $\tilde{\mathbb{N}}^d$	integer ring mod 2^d, $\mathrm{d}\underset{\cdot}{\mathrm{I}}\underset{\cdot}{\mathrm{I}}\dot{Int}s$ of dimension d
↓ information diffusion	$\tilde{\mathbb{P}}^d$, $\mathbb{F}$	finite field of order 2^d, $\mathrm{d}\underset{\cdot}{\mathrm{I}}\underset{\cdot}{\mathrm{I}}\breve{P}olys$ of dimension d

Easier computation. From another point of view, an $\mathrm{d}\underset{\cdot}{\mathrm{I}}\underset{\cdot}{\mathrm{I}}\dot{Int}s$ might replace a finite fields as it is easier to compute both in the HW as well as the SW model. In this case, we need to carefully balance the information diffusion in an finite field with the speed of an $\mathrm{d}\underset{\cdot}{\mathrm{I}}\underset{\cdot}{\mathrm{I}}\dot{Int}s$. In particular, `xoring` two binary vectors is computationally much easier than adding two numbers. Alas, information diffusion is much worse in the former case. However, if there are some inversions in the cryptographic primitive, this might balance out information diffusion in comparison to finite fields.

To get a full picture, we also need to consider $\mathrm{d}\underset{\cdot}{\mathrm{I}}\underset{\cdot}{\mathrm{I}}\breve{P}olys$, see Sect. 2.2 for details. In addition, see Sect. 7 for considerations regarding concrete implementations.

2.4 Abstract $\mathrm{d}\underset{\cdot}{\mathrm{I}}\underset{\cdot}{\mathrm{I}}\curvearrowleft Field$s and Definitions

A $\mathrm{d}\underset{\cdot}{\mathrm{I}}\underset{\cdot}{\mathrm{I}}\curvearrowleft Field$ is an abstraction of $\mathrm{d}\underset{\cdot}{\mathrm{I}}\underset{\cdot}{\mathrm{I}}\dot{Int}s$ and $\mathrm{d}\underset{\cdot}{\mathrm{I}}\underset{\cdot}{\mathrm{I}}\breve{P}olys$. We start by revisiting the **FlipTrix** from the beginning of Sect. 2.1. Consider some non-empty set $\tilde{\mathfrak{C}}$. We call it the a **FlipCarrier**. In addition consider the tally operator $[a, b, c] \to d$ with $a, b, c \in \tilde{\mathfrak{C}}$ and some element d. We call this function the **tally**. We say the **FlipCarrier** and the corresponding tally form a **FlipTrix** if the axioms of Table 4 hold. Note that there is no additive neutral in an $\mathrm{d}\underset{\cdot}{\mathrm{I}}\underset{\cdot}{\mathrm{I}}\curvearrowleft Field$. Hence,

Table 4: Axioms of the tally and the **FlipTrix** for $\mathrm{d}\underset{\cdot}{\mathrm{I}}\underset{\cdot}{\mathrm{I}}\curvearrowleft Field$s

Closure. $\forall a, b, c \in \tilde{\mathfrak{C}}$ we have $[a, b, c] \in \tilde{\mathfrak{C}}$.

Associativity. $\forall a, b, c, d, e \in \tilde{\mathfrak{C}}$ we have $[[a, b, c], d, e] = [a, [b, c, d], e] = [a, b, [c, d, e]]$.

Additive (Neutral) and Inverse. $\forall a, b \in \tilde{\mathfrak{C}}$ we have $[a, b, -b] = a$ and call $-b$ the **additive inverse** of b.

Commutativity. $\forall a, b, c \in \tilde{\mathfrak{C}}$ we have
$[a, b, c] = [a, c, b] = [b, a, c] = [b, c, a] = [c, a, b] = [c, b, a]$.

this notion was captured in one commen axiom on *additive neutral and inverse*. In addition, Lem. 1 carries through, so we can use the tally operator with any odd number of operands.

For an $\mathrm{d}\underset{\cdot}{\mathrm{I}}\underset{\cdot}{\mathrm{I}}\curvearrowleft Field$, we also need a distributive law and a multiplicative group. While we refer to [LN02] for the group laws, we explicitly state then distributive law $a \cdot [b, c, d] = [a \cdot b, a \cdot c, a \cdot d]$ for $a, b, c, d \in \tilde{\mathbb{N}}^d$.

If all lf the above axioms are satisfied, we call such a structure an $\mathrm{d}\underset{\cdot}{\mathrm{I}}\underset{\cdot}{\mathrm{I}}\curvearrowleft Field$. By construction, $\mathrm{d}\underset{\cdot}{\mathrm{I}}\underset{\cdot}{\mathrm{I}}\dot{Int}s$ and $\mathrm{d}\underset{\cdot}{\mathrm{I}}\underset{\cdot}{\mathrm{I}}\breve{P}olys$ fulfull the above axioms.

Definitions. We start with the definition of **characteristic**. Let $a, b \in \tilde{\mathfrak{C}}$. If we have

$$a + \sum_{i=0}^{t} b = a$$

for every a, b and some $t \in \mathbb{N}, t > 1$ we call the minimal such t the **characteristic** of this particular $\mathrm{d}\underset{\cdot}{\mathrm{I}}\underset{\cdot}{\mathrm{I}}\curvearrowleft Field\ \tilde{\mathfrak{C}}$, also see [LN02, Sect. 1.2], in particular Def. 1.43. If no such t exists, we say that the $\mathrm{d}\underset{\cdot}{\mathrm{I}}\underset{\cdot}{\mathrm{I}}\curvearrowleft Field$ has characteristic 0. In the remainder, we are particularity interested in $\mathrm{d}\underset{\cdot}{\mathrm{I}}\underset{\cdot}{\mathrm{I}}\curvearrowleft Field$s of characteristics larger than zero.

Fact. For all $\mathrm{d}\underset{\cdot}{\mathrm{I}}\underset{\cdot}{\mathrm{I}}\breve{P}olys$ we have characteristic 2 while for the $\mathrm{d}\underset{\cdot}{\mathrm{I}}\underset{\cdot}{\mathrm{I}}\dot{Int}s$ of dimension d, the characteristic is 2^d.

In the multiplicative group, we recall that the **order** of an element $a \in \tilde{\mathfrak{C}}$ of the multiplicative group is the smallest $k \in \mathbb{N}, k \geq 1$ such that $a^k = 1$.

In addition, the number of elements in a finite $\mathrm{d}\underset{\cdot}{\mathrm{I}}\underset{\cdot}{\mathrm{I}}\curvearrowleft Field$ is called its **order**. Both in $\mathrm{d}\underset{\cdot}{\mathrm{I}}\underset{\cdot}{\mathrm{I}}\dot{Int}s$ as in $\mathrm{d}\underset{\cdot}{\mathrm{I}}\underset{\cdot}{\mathrm{I}}\breve{P}olys$ of dimension $d \in \mathbb{N}$ the order of the multiplicative group is 2^d. The coincides with the number of elements in the *FlipTrix*. Note that this is different from finite fields where the number of elements in the additive and multiplicative group is different. The reason for this is that $\mathrm{d}\underset{\cdot}{\mathrm{I}}\underset{\cdot}{\mathrm{I}}\curvearrowleft Field$s do not contain zero as an element.

uədO P̆roblem 1.

i. Which other structurs but $\mathrm{d}\underset{\cdot}{\mathrm{I}}\underset{\cdot}{\mathrm{I}}\dot{Int}s$ and $\mathrm{d}\underset{\cdot}{\mathrm{I}}\underset{\cdot}{\mathrm{I}}\breve{P}olys$ satisfy the axioms from Tab. 4?

ii. Can we drop commutativity in the multiplicative group but still preserve cryptographically relevant properties?

iii. Is the order of $\mathrm{d}\underset{\cdot}{\mathrm{I}}\underset{\cdot}{\mathrm{I}}\curvearrowleft Field$ always of the form 2^d for some $d \in \mathbb{N}, d \geq 1$?

iv. Can we find $\mathrm{d}\underset{\cdot}{\mathrm{I}}\underset{\cdot}{\mathrm{I}}\curvearrowleft Field$s that have a characteristic not divisible by 2?

Possible candidates for this problem could be matrix rings over the integers, finite fields, $\mathrm{d}\underset{\cdot}{\mathrm{I}}\underset{\cdot}{\mathrm{I}}\dot{Int}s$ or $\mathrm{d}\underset{\cdot}{\mathrm{I}}\underset{\cdot}{\mathrm{I}}\breve{P}olys$ although these possibilities have not been investigated deeper in this paper.

3 Applications to Cryptology

In this section, we take a deeper dive into cryptographic schemes, in particular from the Post-Quantum world. Our aim is to investigate, if these schemes could benefit from using dịlƎ↶*Field*s rather than finite fields as their basis. For Unbalanced Oil and Vinegar (Sect. 3.1, Learning with Errors (Sect. 3.2) and Saber (Sect. 3.3, the answer is affirmative. For other schemes, it is not so clear, cf Sect. 3.4 .

to see how dịlƎ*Inṫs* are used in practice. In a second step, we take a bird's perspective on various cryptographic algorithm to see how they might benefit from dịlƎ*Inṫs*.

3.1 Unbalanced Oil and Vinegar

Unbalanced Oil and Vinegar [KPG99] is a $\mathcal{M}$ultivariate $\mathcal{Q}$uadratic public key scheme that was submitted to the NIST Post-Quantum Cryptography Standardization [N-PQC]. For example, see [Wo05] for this and other $\mathcal{M}$ultivariate $\mathcal{Q}$uadratic constructions. A state-of-the-art overview of UOV can be found in [UOV25].

We give a brief sketch of UOV here, adapting it already to dịlƎ↶*Field*s. The secrete key consists of a linear map $T : \tilde{\mathfrak{C}}^n \to \tilde{\mathfrak{C}}^n$. We express this map via a bespoke LU decomposition, cf. App. A.2. Note that n needs to be an odd number. Next, we have the internal map $\mathcal{F}$ which consists of m $\mathcal{M}$ultivariate $\mathcal{Q}$uadratic polynomials $p_0, \ldots, p_{m-1}$. Note that m needs to be an odd number, too. In addition, we have special variables called **vinegar variables**. We have $n = v + m$, so v is an even number.

$$p^{(k)}(x_0, \ldots, x_{n-1}) := \sum_{i=0}^{v-1} \sum_{j=0}^{n-1} a_{i,j}^{(k)} \cdot x_i x_j \text{ for } 0 \leq k < m.$$

The number of terms for each polynomial p_k of the internal map $\mathcal{F}$ is given by vn. Hence, the overall number of terms will be even, not allowing a tally over a dịlƎ↶*Field*. However, there is an easy tweak to fix this problem: We remove a (small) number of terms from each polynomial. From a practical point of view, we have $a_{i,j}^{(k)} \in \tilde{\mathfrak{C}} \cup 0$. All terms with $a_{i,j}^{(k)} = 0$ will vanish in the above sum. Call the number of vanishing terms $\nu \in \mathbb{N}$. We suggest to keep the number ν as small as possible, preferable 1 and choose the individual vanishing terms at random inside its polynomial p_i for $0 \leq i \leq m$.

We now generate the public key as $\mathcal{P} := \mathcal{F} \circ T$. To stay inside the dịlƎ↶*Field* structure, we need the number of terms to be odd. We obtain

$$\tilde{p}^{(k)}(x_0, \ldots, x_{n-1}) := \sum_{i=0}^{n-1} \sum_{j=i}^{n-1} \tilde{a}_{i,j}^{(k)} \cdot x_i x_j \text{ for } 0 \leq k < m.$$

The terms per polynomial are $n(n+1)/2$. This number needs to be odd to allow to form a tally. Following [UOV25], we suggest the following set of parameters, see also `paramsUOV`:

NIST Security Level	Name	n	v	m	ν	d	D
1	flipUOV-1a	113	68	45	1	8	512
1	flipUOV-1b	161	96	65	1	4	32
3	flipUOV-3	185	112	73	1	8	512
5	flipOUV-5	245	148	97	1	8	512

uədO P̆roblem 2

i. Is the above construction secure? How do attacks against UOV scale against flipUOV? Do they work at all?

ii. How fast does an optimized implementation of flipUOV run against UOV?

iii. Are the known attacks against UOV applicable? May we safely reduce the parameters given above?

3.2 Learning With Errors

Learning with errors (LWE) has been introduced in [Re05]. In Sect. 3.3, we will see that the special case "learning with rounding" can be expressed over $\mathrm{d}\dot{\mathrm{i}}\mathrm{l}\mathrm{\exists}\mathit{Int}\dot{s}$. In this section, we have a closer look at the original LWE, in particular the encryption function.

We start with some dimension $d \in \mathbb{N}$, *e.g.* $d > 10$ and some modulus $q = 2^{d+1}$. We now define $\tilde{\mathbb{N}}^d := \{1, 3, \ldots, 2^{d+1} - 1\}$. We have some security parameters $m, n \in \mathbb{N}$. Here, m is the sample size and n is the dimension. In addition, we have a maximal error offset $\varepsilon \in \mathbb{N}$ with $\varepsilon \ll |\tilde{\mathbb{N}}^d|$. The concrete value ε is also a security parameter. On this basis, we define

i. Choose a secret vector $s \in_R \left(\tilde{\mathbb{N}}^d\right)^n$

ii. Choose a random matrix $A \in_R \left(\tilde{\mathbb{N}}^d\right)^{m \times n}$ and a random error vector $e \in_R \left(\{i \in \tilde{\mathbb{N}}^d : 1 \leq i \leq \varepsilon\}\right)^m$.

iii. Now compute $b := As + e$. Note that we have $b \in \left(\tilde{\mathbb{N}}^d\right)^m$ by construction.

iv. Publish the pair (A, b) as public key. The vector s is the secret key.

Using the public key pair (A, b), we now look into encryption. Therefore we use a one-bit-ciphertext $c \in \{0, 1\}$. On this basis Alice computes

i. Choose a random vector $r \in_R \{0, 1\}^m$.

ii. Output as ciphertext $(u, v) := (A^T r, b^T r c 2^d)$.

For decryption, we use the following formula:

$$v = b^T r + c2^d = (As + e)^T r + c2^d = s^T A^T r + e^T r + c2^d = s^T u + e^T r + c2^d .$$

By construction, $e^T r$ is still "small' in comparison to 2^d, so Bob can distinguish between the two cases 0 and 2^d and hence decode the bit c with high probability.

For m, n odd numbers, the above scheme naturally works over $\mathrm{d}\dot{\mathrm{i}}\mathrm{l}\mathrm{\exists}\mathit{Int}\dot{s}$. However, we can only encode 1 bit at a time. In the next section, we come to a scheme with a higher bandwidth for key encapsulation.

3.3 Saber

Saber is a 2-party key exchange protocol [DKRV18]. It is based on Mod-LWR (Module Learning With Rounding problem) and is very close to the original LWE [Re05]. We introduce an adapted variant of Saber, already expressing it over $\mathrm{d}\dot{\mathrm{i}}\mathrm{l}\mathrm{\exists}\mathit{Int}\dot{s}$. We certainly do not recommend to use it in practice, but more as a study objects as many details are still unclear. Our treatment follows [DKRV18, Sect. 3].

We have the security parameters p, q which are powers of 2 and some parameter t. We have $q > p > (t + 1)$. We have $d = q$ and hence $|\tilde{\mathbb{N}}^d| = 2^q = 2^d$. We start with a random seed *seedA*

Table 5: Key Exchange Protocol Saber over $\mathrm{d}\underset{\cdot}{\mathrm{i}}\underset{\cdot}{\mathrm{l}}\underset{\cdot}{\mathrm{I}}\mathit{Int}\dot{s}$

1	$seedA \in_R \{0,1\}^{256}$		
2	$A :=$ gen(seedA) with $A \in \tilde{\mathbb{N}}^{l\times l}$		
3	$s \in_R \tilde{\mathbb{N}}^l$		$s' \in_R \tilde{\mathbb{N}}^l$
4	b := emb($[As, h, h]$) $\in (\tilde{\mathbb{N}}^p)^l$	$\xrightarrow{b,\, seedA}$	$A :=$ gen(seedA) with $A \in \tilde{\mathbb{N}}^{l\times l}$
5			$b' :=$ emb($[A^T s', h, h]$) $\in (\tilde{\mathbb{N}}^p)^l$
6			$v' := [b^T$ bits(s') $, h_1, h_1] \in \tilde{\mathbb{N}}^p$
7	$v := [b'^T$ bits(s), $h_1, h_1] \in \tilde{\mathbb{N}}^p$	$\xleftarrow{b',\, c}$	c = emb(v')$\in R^t$
8	$k = [$emb$(v - 2^{p-t-1}c, h_2, h_2])$		k' := emb(v')

of at least 256 bits. Let $A \in \tilde{\mathbb{N}}^{l\times l}$ for $l \in \mathbb{N}, l$ odd. In addition, we have two vectors $s, s' \in \tilde{\mathbb{N}}^l$ which are sampled uniformly for some binomial β_μ within $[-\mu, \mu]$. In particular, we do not need the inverse of the matrix A but only need its transpose A^T. Hence, we can randomly choose the matrix A over $\tilde{\mathbb{N}}$. Similarly, there are no further restrictions on the vectors s, s' but being uniformly sampled. For our purpose, the following constants are interesting. First, we have the vector $h \in (\tilde{\mathbb{N}}[x])^l$ where all polynomials have their coefficients set to $2^{q-p-2} + 1$. Second, there is the constant polynomial h_1 with all coefficients set to $2^{q-p-2} + 1$, too. Finally, there is the constant polynomial h_2 where all coefficients are set to $(2^{p-2} - 2^{p-t-2}) + 1$. Computations are performed in the ring $\tilde{\mathbb{N}}[x]$. We have the method bits that outputs a vector from $\tilde{\mathbb{N}}^l$ and truncates elements, for example from $\tilde{\mathbb{N}}^q$ to $\tilde{\mathbb{N}}^p$ by only using the least significant bits. In this case, the output of bits is an element from $\tilde{\mathbb{N}}^p$. Hence, the output of bits strongly depends on the input. Following the discussion in Sect. 4.8, we need two consecutive reduction functions. We choose them to be $X^{257} + 1$ and $X^{256} + 3$, respectively. Both parties agree on a common secret if the errors in the computation are not too big using a key derivation function on $k = k'$ after Step 8 is successfully completed, see Tab. 5 for the full protocol of FlipSaber. Inspecting Line 8, we see the operator "emb". It is just a shortcut for considering the input mod 2^q as element mod 2^p, *i.e.* "embed" them into the smaller $\mathrm{d}\underset{\cdot}{\mathrm{i}}\underset{\cdot}{\mathrm{l}}\underset{\cdot}{\mathrm{I}}\mathit{Int}\dot{s}$. This is well-defined as $p \mid q$.

Following the parameters of [BDK+21], we obtain:

NIST Security Level	Name	l	p	q	t
3	FlipSaber-III	3	10	13	3

It is out of the scope of this paper to compute the actual error probability. Note that [BDK+21] also gives sets of parameters for security levels $1, 5$. Alas, the corresponding parameters for l are $2, 4$—and hence even number. Therefore, the matrix A is not defined over $\mathrm{d}\underset{\cdot}{\mathrm{i}}\underset{\cdot}{\mathrm{l}}\underset{\cdot}{\mathrm{I}}\mathit{Int}\dot{s}$. However, it is for security level 3 it is and can be fully implemented over $\mathrm{d}\underset{\cdot}{\mathrm{i}}\underset{\cdot}{\mathrm{l}}\underset{\cdot}{\mathrm{I}}\mathit{Int}\dot{s}$, in particular as no matrix-vector-equation needs to be solved, *i.e.* $b = As$ for given matrix A, given vector b and unknown vector s. This can be a problem over $\mathrm{d}\underset{\cdot}{\mathrm{i}}\underset{\cdot}{\mathrm{l}}\underset{\cdot}{\mathrm{I}}\mathit{Int}\dot{s}$, cf. 4.2.

We leave it as **uədO P̆roblem 3**

i. Verify if the above set of parameters is secure and also give the failure probabilty and workload for a successful attack.

ii. How could NTT (Number Theoretic Transformation) be used over $\mathrm{d}\underset{\cdot}{\mathrm{i}}\underset{\cdot}{\mathrm{l}}\underset{\cdot}{\mathrm{I}}\curvearrowleft\mathit{Field}$s to speed up the above Saber instantiation?

3.4 Misceangelous Cryptographic Primitives

In this section, we take a bird's view on several cryptographic primitives and see if they can be implemented over $\mathrm{d}\underset{\cdot}{\mathrm{l}}\underset{\cdot}{\mathrm{l}}\mathrm{\Lsh}\curvearrowleft Field$s. In particular, we observe that multiplication and division are performed in a group and are hence well-defined for $\mathrm{d}\underset{\cdot}{\mathrm{l}}\underset{\cdot}{\mathrm{l}}\mathrm{\Lsh}\dot{Int}s$. The same goes for addition—if the number of terms is odd and hence can be implemented by a tally. Armed with this this knowledge, we do a quick inspection on some cryptographic primitives.

3.4.1 McEliece encryption scheme

We start with a quick recap of the McEliece encryption scheme as an example for a flexible, coding based scheme, cf. [GAA+25, WGR24]. We use an (n, k)-linear code $\mathcal{C}$ over GF(2) and have $n > k$ for $n, k \in \mathbb{N}$. In addition, there is a maximal weight $t \in \mathbb{N}$. We start with the key generation algorithm:

1. Select some generator matrix $G \in \mathbb{F}_2^{k\times n}$ for this code $\mathcal{C}$. The latter has an efficient decryption algorithm $\mathcal{A}$ for given (n, k, t).
2. Select a random, invertible matrix $S \in_R \mathbb{F}_2^{k\times k}$.
3. Select a random permutation matrix $P \in \mathbb{F}_2^{n\times n}$.
4. Set $H := SGP$ as the public key
5. Note that the private key consists of $(S, P, \mathcal{A})$

We inspect if we can use $\mathrm{d}\underset{\cdot}{\mathrm{l}}\underset{\cdot}{\mathrm{l}}\mathrm{\Lsh}\curvearrowleft Field$s in this context. While we have a definition of an invertible matrix over $\mathrm{d}\underset{\cdot}{\mathrm{l}}\underset{\cdot}{\mathrm{l}}\mathrm{\Lsh}\curvearrowleft Field$s (cf. App. B.2), we have no way to actually check for a given random matrix if it is invertible. However, we can construct invertible matrices via the Bespoke LU Decomposition, cf. App. A.2. Second, a permutation matrix does not exist over $\mathrm{d}\underset{\cdot}{\mathrm{l}}\underset{\cdot}{\mathrm{l}}\mathrm{\Lsh}\curvearrowleft Field$s as 0 is not part of the *FlipCarrier*. We can remedy this by not using a permutation *matrix* but simply a *permutation* of columns in SG. That is, we choose a random permutation π over n elements and permute the result of SG accordingly. This will lead to an $(k \times n)$-matrix as in the original scheme. Unfortunately, the final problem is the greatest—we do not know a linear code over $\mathrm{d}\underset{\cdot}{\mathrm{l}}\underset{\cdot}{\mathrm{l}}\mathrm{\Lsh}\curvearrowleft Field$s. In addition, it is very unlikely that such an algorithm will be different from the corresponding algorithm over GF(2). Hence it is open problem if an encryption algorithm based on coding schemes based on GF(2) will actually be useful in practice.

However, assuming that such an algorithm actually exists, we would need to take care of the encoding of the message. In particular, we suggest to use a $\mathrm{d}\underset{\cdot}{\mathrm{l}}\underset{\cdot}{\mathrm{l}}\mathrm{\Lsh}\curvearrowleft Field$ of small dimension, *e.g.* $d = 2, 4$ for this purpose. Now, we can split the input in bitstrings of size 2 (or 4, respectively), and embed it into a message space for the corresponding *FlipCarrier*.

For the sake of completeness, we also inspect encryption and decryption, starting with encryption when Bob wants to send a message to Alice.

1. Bob's message is $m \in \tilde{\mathbb{N}}^{4\times k}$.
2. He randomly chooses an error vector $z\tilde{\mathbb{N}}^{4\times n}$ with exactly t non-zero elements.
3. He computes $c := mH + z$ and transmits c as his message.

For decryption, Alice performs the following steps:

1. Alice computes $c' := cP^{-1}$.
2. She uses the decoding algorithm for $\mathcal{C}$ to decode c' to m'.

3. She computes $m = m'S^{-1}$.

Using the ideas for key generation, we can replace the permutation matrix P with an actual permutation π. Similarly, we replace the inverse matrix S^{-1} by solving the corresponding equation using Bespoke LU decomposition. There is a slight problem with the error vector $z \in \tilde{\mathbb{N}}^{4\times n}$. Obviously, it is not expressible over $\mathrm{d}_{\dot{\mathbb{I}}\mathbb{L}\mathbb{H}}\curvearrowleft Fields$ as we do not have a 0-element. Secondly, we cannot add a single 1 to an element of a $\mathrm{d}_{\dot{\mathbb{I}}\mathbb{L}\mathbb{H}}\curvearrowleft Field$. However, we can also remedy this. First, we select t coefficients in the original message $m \in \tilde{\mathfrak{C}}^n$ but leave the other coefficients unchanged. Second, we form a tally by randomly selecting two elements from $\tilde{\mathfrak{C}}$ and hence form a valid tally for each of these t coefficients. This way, we have in practice applied an error vector of weight t without adding two elements or using a zero-element in this vector. All operations are within the corresponding $\mathrm{d}_{\dot{\mathbb{I}}\mathbb{L}\mathbb{H}}\curvearrowleft Field$. To keep a large distance to codes over GF(2), it might be prudent to concentrate on $\mathrm{d}_{\dot{\mathbb{I}}\mathbb{L}\mathbb{H}}\dot{Int}s$ instead of $\mathrm{d}_{\dot{\mathbb{I}}\mathbb{L}\mathbb{H}}\check{P}olys$ here.

At first glance, this result looks good—it seems we have shown that the McEliece encryption scheme in all of its three steps key generation, encryption and decryption can be used over $\mathrm{d}_{\dot{\mathbb{I}}\mathbb{L}\mathbb{H}}\curvearrowleft Fields$, as nearly all steps can be transferred from GF(2) to a $\mathrm{d}_{\dot{\mathbb{I}}\mathbb{L}\mathbb{H}}\curvearrowleft Field$. Alas, this is not true as we are lacking the most crucial step, namely the decoding algorithm $\mathcal{A}$. We simply do not have such an algorithm. Therefore, we have to leave the existence of a secure coding based encryption scheme over $\mathrm{d}_{\dot{\mathbb{I}}\mathbb{L}\mathbb{H}}\curvearrowleft Fields$ as a **uəd$_{\mathbf{O}}$ Ŏ Problem 4**.

However, aimed with this knowledge, we can look at linear coding schemes over $\tilde{\mathfrak{C}}^2$ and $\tilde{\mathfrak{C}}^4$, respectively. Potential candidates might be Reed-Salomon, Goppa and Turbo Codes for this purpose. Unfortunately, it is out of the scope of this paper to investigate this question further, hence see Open Problem 4 above.

3.4.2 NTRU

The primitive from [NTRU] was submitted to Round 3 of [N-PQC]. Since then, the parameters have been updated. Current parameters and an up-to-date version of NTRU are given in [FPC+25]. In particular, for NTRU-HPS we have $q = 2048 = 2^{11}$ or $q = 4096 = 2^{12}$, while these parameters for NTRU-HPPS are $q = 8192 = 2^{13}$ or $q = 16384 = 2^{14}$, respectively. The size of the polynomial are given with $N = 677, 821, 1229, 701, 1373$.

At the heart of NTRU, we have a cyclic convoluted product of two polynomials f, g over the integer ring modulo q of degree N as $h := f \circledast g$ with coefficients h_k defined as

$$h_k = \sum i = 0^k f_i g_{k-i} + \sum_{i=k+1}^{N+1} f_i g_{N+k-i} = \sum_{i+j\equiv k \pmod N} f_i g_j \pmod q .$$

As we will see in Sect. 4.9, we need to ensure that both polynomials are chosen carefully s/t the result is meaningful over an $\mathrm{d}_{\dot{\mathbb{I}}\mathbb{L}\mathbb{H}}\curvearrowleft Field$. On the lowest level, NTRUEncrypt uses addition and multiplication of elements in the ring modulo 2^k for some $k \in \mathbb{N}$ and the mixing of information is done via randomness in one of the polynomials or its inverse. To express NTRUEncrypt over $\mathrm{d}_{\dot{\mathbb{I}}\mathbb{L}\mathbb{H}}\dot{Int}s$, we need to make sure that all intermediate sums actually have an odd number of terms. In particular, this is true for the polynomials f, g, h. Alas, there is a catch: For efficiency, some coefficients are defined over $\{-1, 0, 1\}$ where 0 is clearly no element of $\mathrm{d}_{\dot{\mathbb{I}}\mathbb{L}\mathbb{H}}\dot{Int}s$. It is out of the scope of this paper to examine if this set might be replaced by $\{-1, 1, 3\}$ or something similar. Main problem to express NTRU over $\mathrm{d}_{\dot{\mathbb{I}}\mathbb{L}\mathbb{H}}\curvearrowleft Fields$ is polynomial multiplication, cf. Sect. 4.9. Hence, the construction of a full NTRU scheme over an $\mathrm{d}_{\dot{\mathbb{I}}\mathbb{L}\mathbb{H}}\curvearrowleft Field$ is therefore left as **uəd$_{\mathbf{O}}$ Ŏ Problem 5**.

3.4.3 ML-KEM

Learning with errors (LWE) has been introduced in [Re05]. In Sect. 3.3, we have already discussed the special case "learning with rounding" and concluded that $\mathrm{d}_{\mathrm{I}\mathbb{F}}\dot{Int}s$ can be used to define a flip-variant of it. We briefly discuss ML-LWE or [FIPS203] which is one of the finalists of the NIST Post-Quantum Cryptography competition,

Different to Saber, it is not possible to express ML-KEM directly over $\mathrm{d}_{\mathrm{I}\mathbb{F}}\curvearrowleft Fields$. Let $q = 2^{d+1}$ be some prime power. Then we have $\tilde{\mathbb{N}}^d$ as the basis of the ML-KEM problem. Now, all operations are performed in the polynomial ring $\tilde{\mathbb{N}}^d[X]/(X^n + 1)$ for some $n \in \mathbb{N}$. This is clearly not possible in a $\mathrm{d}_{\mathrm{I}\mathbb{F}}\curvearrowleft Field$ as we will get even coefficients after each reduction. While we will see a trick to deal with this problem in Sect. 4.8, this is not our only problem. Second, multiplying polynomials leads to intermediate results with an even number of terms—and hence an even output. A more detailed discussion can be found in Sect. 4.9. A possible solution might be use "corrections" (adding or subtracting 1 for the individual term). While is looks like a possible solution at first glance, it changes the probability distribution of ML-KEM and hence has to be handled with care. Third, ML-KEM uses a lot of optimizations—each of them poses a challenge for the adaptation.

So all in all, we have to leave the adaptation of ML-KEM/MLWE to $\mathrm{d}_{\mathrm{I}\mathbb{F}}\curvearrowleft Fields$ as an **uəd$_{\mathbf{O}}$ Problem 6**.

3.4.4 S-Boxes

For the AES, the S-boxes are designed as x^{-1} over GF(2^8). It is way out of scope of this paper to put in as much effort into researching an alternative S-box using $\mathrm{d}_{\mathrm{I}\mathbb{F}}\curvearrowleft Fields$. However, we can at least check if we can build a bijective S-box using $\mathrm{d}_{\mathrm{I}\mathbb{F}}\dot{Int}s$. An alternative way to create S-boxes might lie in $\mathrm{d}_{\mathrm{I}\mathbb{F}}\breve{P}olys$.

An obvious hurdle to bijectivity is the least-significant bit, which is always 1. However, when embedding an 8-bit value into a $\mathrm{d}_{\mathrm{I}\mathbb{F}}\curvearrowleft Field$ with $d = 8$ and projecting the highest 8 bits from the result, we might be able to construct a bijective S-box. Over $\mathrm{d}_{\mathrm{I}\mathbb{F}}\dot{Int}s$, we will use the function $S(x) := \lfloor (2(x+1)^{-1})/2 \rfloor - 1$. We walk through this function step-by-step: First, we need to embed the numbers $0, \ldots, 255$ into the $\mathrm{d}_{\mathrm{I}\mathbb{F}}\dot{Int}s$ of dimension $d = 8$. Therefore, we use 2(x+1). The actual S-box operation is computing the inverse element via $^{-1}$. Finally, we have to transfer the elements from $\tilde{\mathbb{N}}^8$ back to the set $\{0, \ldots, 255\}$. We see the results for both $\mathrm{d}_{\mathrm{I}\mathbb{F}}\dot{Int}s$ and $\mathrm{d}_{\mathrm{I}\mathbb{F}}\breve{P}olys$ in Tab. 6. Both have been generated with the Python function `SboxBijective`. We follow a similar route for $\mathrm{d}_{\mathrm{I}\mathbb{F}}\breve{P}olys$. Here, we have to transfer the integers to the vector space GF(2^{d+1}). We do this using the function $\tau(x) := \sum_{i=0}^{d} 2^i x_i$ and its inverse. The result is shown in Tab. 6(b). We see that both S-boxes are bijections. Hence, they fulfil the first requirement for a good S-box. However, we see that they have a very high degree of regularity—both for $\mathrm{d}_{\mathrm{I}\mathbb{F}}\dot{Int}s$ and $\mathrm{d}_{\mathrm{I}\mathbb{F}}\breve{P}olys$. A similar result was obtained for $d = 4, 8, 12$ and 16, respectively. We also tried other exponential functions like x^3, x^7, x^{17} or x^{127} and dimensions $d = 4, 12, 16$—to no avail. However, it is interesting to see that the corresponding S-boxes have such a high degree of regularity. The reason for this is not known at the moment.

uəd$_{\mathbf{O}}$ Problem 7.

i. Why is there such a high regularity in the S-boxes using $\mathrm{d}_{\mathrm{I}\mathbb{F}}\curvearrowleft Fields$?

ii. Can we construct secure S-boxes based on $\mathrm{d}_{\mathrm{I}\mathbb{F}}\curvearrowleft Fields$? How do they look like?

Table 6: S-box x^{-1} for $d = 8$

(a) For $\mathrm{d}_{\mathrm{ILI}}$*Int*̇*s*

	00	01	02	03	04	05	06	07	08	09	0a	0b	0c	0d	0e	0f
00	00	55	66	db	1c	d1	62	77	78	0d	9e	d3	14	09	1a	ef
10	f0	c5	d6	cb	0c	41	d2	67	68	7d	0e	c3	04	79	8a	df
20	e0	35	46	bb	fc	b1	42	57	58	ed	7e	b3	f4	e9	fa	cf
30	d0	a5	b6	ab	ec	21	b2	47	48	5d	ee	a3	e4	59	6a	bf
40	c0	15	26	9b	dc	91	22	37	38	cd	5e	93	d4	c9	da	af
50	b0	85	96	8b	cc	01	92	27	28	3d	ce	83	c4	39	4a	9f
60	a0	f5	06	7b	bc	71	02	17	18	ad	3e	73	b4	a9	ba	8f
70	90	65	76	6b	ac	e1	72	07	08	1d	ae	63	a4	19	2a	7f
80	80	d5	e6	5b	9c	51	e2	f7	f8	8d	1e	53	94	89	9a	6f
90	70	45	56	4b	8c	c1	52	e7	e8	fd	8e	43	84	f9	0a	5f
a0	60	b5	c6	3b	7c	31	c2	d7	d8	6d	fe	33	74	69	7a	4f
b0	50	25	36	2b	6c	a1	32	c7	c8	dd	6e	23	64	d9	ea	3f
c0	40	95	a6	1b	5c	11	a2	b7	b8	4d	de	13	54	49	5a	2f
d0	30	05	16	0b	4c	81	12	a7	a8	bd	4e	03	44	b9	ca	1f
e0	20	75	86	fb	3c	f1	82	97	98	2d	be	f3	34	29	3a	0f
f0	10	e5	f6	eb	2c	61	f2	87	88	9d	2e	e3	24	99	aa	ff

(b) For $\mathrm{d}_{\mathrm{ILI}}$*P*̌*olys*

	00	01	02	03	04	05	06	07	08	09	0a	0b	0c	0d	0e	0f
00	00	ff	aa	6d	24	cb	4e	99	88	d7	a2	c5	ac	e3	46	31
10	10	af	ba	3d	34	9b	5e	c9	98	87	b2	95	bc	b3	56	61
20	20	5f	8a	cd	04	6b	6e	39	a8	77	82	65	8c	43	66	91
30	30	0f	9a	9d	14	3b	7e	69	b8	27	92	35	9c	13	76	c1
40	40	bf	ea	2d	64	8b	0e	d9	c8	97	e2	85	ec	a3	06	71
50	50	ef	fa	7d	74	db	1e	89	d8	c7	f2	d5	fc	f3	16	21
60	60	1f	ca	8d	44	2b	2e	79	e8	37	c2	25	cc	03	26	d1
70	70	4f	da	dd	54	7b	3e	29	f8	67	d2	75	dc	53	36	81
80	80	7f	2a	ed	a4	4b	ce	19	08	57	22	45	2c	63	c6	b1
90	90	2f	3a	bd	b4	1b	de	49	18	07	32	15	3c	33	d6	e1
a0	a0	df	0a	4d	84	eb	ee	b9	28	f7	02	e5	0c	c3	e6	11
b0	b0	8f	1a	1d	94	bb	fe	e9	38	a7	12	b5	1c	93	f6	41
c0	c0	3f	6a	ad	e4	0b	8e	59	48	17	62	05	6c	23	86	f1
d0	d0	6f	7a	fd	f4	5b	9e	09	58	47	72	55	7c	73	96	a1
e0	e0	9f	4a	0d	c4	ab	ae	f9	68	b7	42	a5	4c	83	a6	51
f0	f0	cf	5a	5d	d4	fb	be	a9	78	e7	52	f5	5c	d3	b6	01

3.4.5 Trivium

Trivium [DCP09] was chosen in the European Ecrypt project as a stream cipher. It is both fast and stood the test of time. In particular, it is now an the international standard ISO/IEC 29192-3:2012.

Table 7: Update function for flipTrivium.

(a) flipTrivium-6

$$\begin{cases} x_{0,1} := x_{0,1} \oplus a_{i-65} \\ x_{0,3} := x_{0,3} \oplus b_{i-68} \\ x_{0,5} := x_{0,5} \oplus c_{i-65} \end{cases}$$

(b) Trivium-State

$$\begin{cases} a_{i-66} := a_{i-66} \oplus x_{16,2} \\ b_{i-69} := b_{i-69} \oplus x_{16,4} \\ c_{i-66} := c_{i-66} \oplus x_{16,6} \end{cases}$$

(c) flipTrivium-9

$$\begin{cases} x_{0,1} := x_{0,1} \oplus a_{i-65} \\ x_{0,4} := x_{0,4} \oplus b_{i-68} \\ x_{0,7} := x_{0,7} \oplus c_{i-65} \end{cases}$$

We give a brief summary of Trivium, following the treatment in [QW15]: The state of Trivium consists of the three shift registers

$$\begin{cases} A := (a_i, \ldots, a_{i-92}) & \text{of size 93 bits} \\ B := (b_i, \ldots, g_{i-83}) & \text{of size 84 bits} \\ C := (c_i, \ldots, c_{i-83}) & \text{of size 111 bits} \end{cases}$$

Let $K := (k_0, \ldots, k_{79})$ be the 80-bit symmetric key and $V := (v_0, \ldots, v_{79})$ the also 80-bit, publicly known initialization vector. It is assumed that the attacker knows and even might choose the initialization vector V, but not the key K. Based on this, the three registers are initialized as follows:

$$\begin{cases} A := (k_0, \ldots, k_{79}, 0, \ldots, 0) \\ B := (v_0, \ldots, v_{79}, 0, \ldots, 0) \\ C := (0, \ldots, 0, 1, 1, 1) \end{cases}$$

To update the state, we use the following three equations to update the triple A, B, C:

$$\begin{cases} a_i := c_{i-65} + c_{i-110} + c_{i-108}c_{i-109} + a_{i-68} \\ b_i := a_{i-65} + a_{i-92} + a_{i-90}a_{i-91} + b_{i-77} \\ c_i := b_{i-68} + b_{i-83} + b_{i-81}b_{i-82} + c_{i-86} \end{cases}$$

Finally, from the state (A, B, C), we extract a key stream via

$$z_i := c_{i-65} + c_{i-110} + a_{i-65} + a_{i-92} + b_{i-68} + b_{i-83}$$

Before the first bit of the key stream is produced, Trivium is clocked $R = 1152$ times without output.

bits	d	D	N	Name
102	6	128	17	flipTrivium-6
99	9	1024	11	flipTrivium-9

There are several properties we have to consider before adding $\mathrm{flip}\dot{I}nits$ components to Trivium. First, we note that there are at least 64 bits difference between some new bit a_i, b_i, c_i, z_i. This allows for fast implementation on 64-bit systems. Second, we see that there is a constant clocking. This helps for software implementations of Trivium, as there is no branch prediction. Third, there are only three non-linear terms in the update functions of (A, B, C). This keeps the

Table 8: Reduction polynomials for different versions of flipTrivium.

Trivium variant	degree	Polynomial $p(x)$	cycle length	$\log_2$
flipTrivium^1-6a	affine	$7x_3 + 11x_0 + 5$	15,728,160	16.571
flipTrivium^1-6b	affine	$13x_3 + 5x_0 + 1$	15,728,160	16.571
flipTrivium^1-6c	affine	$17x_3 + 11x_0 + 11$	15,728,160	16.571
flipTrivium^2-6a	quadratic	$x_9x_{10} + 3x_8 + 7x_0$	16,251,936	16.604
flipTrivium^2-6b	quadratic	$x_9x_{10} + 3x_8 + 31x_0$	16,251,936	16.604
flipTrivium^2-6c	quadratic	$17x_9x_{10} + 31x_8 + 7x_0$	16,251,936	16.604
flipTrivium^1-9a	affine	$7x_3 + 11x_0 + 5$	499,968	13.122
flipTrivium^1-9b	affine	$13x_3 + 5x_0 + 1$	499,968	13.122
flipTrivium^1-9c	affine	$17x_3 + 11x_0 + 11$	499,968	13.122
flipTrivium^2-9a	quadratic	$x_5x_6 + 3x_4 + 7x_0$	524,032	13.169
flipTrivium^2-9b	quadratic	$x_5x_6 + 3x_4 + 31x_0$	524,032	13.169
flipTrivium^2-9c	quadratic	$17x_5x_6 + 31x_4 + 7x_0$	524,032	13.169
flipTrivium^1-Na	affine	$7x_3 + 11x_0 + 5$	n/a	
flipTrivium^1-Nb	affine	$13x_3 + 5x_0 + 1$	n/a	
flipTrivium^1-Nc	affine	$17x_3 + 11x_0 + 11$	n/a	
flipTrivium^2-Na	quadratic	$x_{\lfloor N/2\rfloor}x_{\lfloor N/2\rfloor+1} + 3x_{\lfloor N/2\rfloor-1} + 7x_0$	n/a	
flipTrivium^2-Nb	quadratic	$x_{\lfloor N/2\rfloor}x_{\lfloor N/2\rfloor+1} + 3x_{\lfloor N/2\rfloor-1} + 31x_0$	n/a	
flipTrivium^2-Nc	quadratic	$17x_{\lfloor N/2\rfloor}x_{\lfloor N/2\rfloor+1} + 31x_{\lfloor N/2\rfloor-1} + 7x_0$	n/a	

implementation of Trivium lean. Last but not least, the bits $a_{i-65}, b_{i-68}, c_{i-65}$ are "recycled" both in the state-update as in producing the key stream.

Aimed with these observations, we want to use Trivium as a testbed how dịlƎ*Inṫs* may be used to enhance the security of symmetric cryptographic primitives. The task is a bit difficult, as dịlƎ*Inṫs* are a rather new concept. Hence, the corresponding construction is for cryptanalytic purposes only—not for uses in real-world systems. After careful cryptanalysis, it might be possible to reduce the number R, *i.e.* reduce the time for the initialization phase. It might even be possible to extract two except of one bit per cycle—although this is out of the scope of this paper.

Details on NLFSR are given in 4.7. We now explain the design rationals. We will add one additional affine or high-degree NLFSR to Trivium. As a full cryptanalysis is not possible, we explain our rationals between the choices. First, we want the additional register to have similar length as the already existing three registers—giving a trade-off between security and efficiency. Second, we want to introduce an additional non-linear component to Trivium. Using dịlƎ*Inṫs*-NLFSR, there are two choices: Third, we use an affine NLFSR as multiplication with a constant is already a non-linear opperation when viewed from GF(2). Second, we want to introduce a multiplication between two registers. Finally, we will not add new bits to Trivium, but toggle existing ones. This way, we make it harder for the cryptanalyst to recover the internal state of Trivium.

Based on these assumptions, we choose two sets of parameters:

Before outlining the addition of Trivium, we first need to fix notation. We denote the state of the additional polynomial by $(x_0, \ldots, x_{N-1})$ and $x_i \in \tilde{\mathbb{N}}$. To address individual bits, we write $x_{i,j}$ for the jth bit of the ith element of the state of the NLFSR-polynomial. Note that we have

$1 \leq j \leq d$ and $x_{i,0} = 1$ by definition. We initialize this state with $(1, \ldots, 1)$.

We start with flipTrivium-6. First, we update the state variable x_0 via, all over the finite field GF(2), see Tab. 7(a). In addition, we flip three bits in Trivium, based on the following formulas over GF(2), see Tab. 7(b). Finally, for flipTrivium-9, we keep the update function of the state (A, B, C) of Trivium but adapt the update function for x_0 to flip bits, see 7(c) to update the state of the NLFSR over $\text{d}\underset{\cdot}{\text{i}}\underset{\cdot}{\text{l}}\text{Ǝ}\mathit{Int}\dot{s}$.

We propose two different variations of flipTrivium, namely an affine and a degree-2 polynomials over the corresponding $\text{d}\underset{\cdot}{\text{i}}\underset{\cdot}{\text{l}}\text{Ǝ}\mathit{Int}\dot{s}$. We have the following set of proposed polynomials, see Tab. 8. The six polynomials were chosen as they constantly give a high cycle length for different values of N. This allows to define a generic flipTrivium^1-N and flipTrivium^2-N, respectively. Note that flipTrivium is only defined for $D = 128, 1024$ due to the update functions given in Tab. 7 also small-scale variant of flipTrivium are certainly possible. All cycle lengths were computed from the start state $x = (1, \ldots, 1)$. They are obviously way too small to enhance Trivium and can be brute forced at ease. Hence, we flip variable x_0 in our proposal. This way, the $\text{d}\underset{\cdot}{\text{i}}\underset{\cdot}{\text{l}}\text{Ǝ}\mathit{Int}\dot{s}$-NLFSR will enter a new cycle frequently, increasing the cycle length.

To test this hypothesis, we were running small scale experiments, see the Python function `TriviumTestFlip`. Due to a lack of memory, we could not increase the parameter N further. Hence, we did not use Trivium in our test but simulated it using random flips on the two update functions from Tab. 7(a+c). The results are given in Tab. 9. Assuming that a Gaussian Random Walk is the best model to approximate the behaviour of the NLFSR-feedback function, we get a cycle length of $\sqrt{2^{(}dN)}$ and hence in the range of $2^{20} \approx 10^6$ for both $N = 5, D = 1024$ and $N = 7, D = 128$.

However, there are several as questions associated with this section.

uədO Ǐroblem 8

i. Is the above construction secure? Does it decrease or increase the security of Trivium?

ii. While keeping the security level of 2^{80}, can we formulate a flipTrivium with smaller register size, *i.e.* does the forth register actually give extra strength against cryptanalysis?

iii. Can $\text{d}\underset{\cdot}{\text{i}}\underset{\cdot}{\text{l}}\text{Ǝ}\mathit{Int}\dot{s}$ help with the security of other stream ciphers?

iv. Why are the six polynomials from Tab 8 so stable over different sizes of N and yield long cycle length? Does this behaviour also hold for higher values of N?

v. Is the model Gaussian Random Walk actually sufficient to explain the behaviour of the forth register in flipTrivium-N? If not—which model to use?

3.4.6 Hidden Subgroup Problem

For example, see [BDF20] to a introduction to the problem. Consider some group G and a subgroup $H \subseteq G$. In addition, we have a set X and a function $f : G \to X$. We say that f **hides** the subset H for all $a, b \in G$ iff $f(a) = f(b)$ for $aH = bH$, *i.e.* the function f is constant on each coset of the subgroup H while it is different for different cosets of H. We call this the **Hidden Subgroup Problem** (HSP).

If G is Abelian, Shor's algorithm can be used to solve the above problem. Examples are the order of an element or the discrete logarithm. In our case, both $\text{d}\underset{\cdot}{\text{i}}\underset{\cdot}{\text{l}}\text{Ǝ}\mathit{Int}\dot{s}$ as well as $\text{d}\underset{\cdot}{\text{i}}\underset{\cdot}{\text{l}}\text{Ǝ}\breve{P}\mathit{olys}$ are Abelian. Hence, we cannot formulate a quantum resilient version of the above problem.

There are several possibilities to construct non-commutative groups, one are matrix groups, in particular matrices with an odd number of rows and columns, respectively. Alas—we do not have a good theory of matrix groups for $\text{d}\underset{\cdot}{\text{i}}\underset{\cdot}{\text{l}}\text{Ǝ}\curvearrowleft\mathit{Field}$ s. Hence, we have to leave the construction of a quantum resistant version of the HSP as an **uədO Ǐroblem 9**.

Table 9: Empirival results for small-scale variante of flipTrivium on the cycle length for the $\mathrm{d}_{\mathrm{flip}}$*Intis*, additional feedback register.

Trivium variant	D	cycle length (empirically, random)				
flipTrivium^1-5a	1024	4,144,642	9,701,858	1,165,733	9,516,033	4,790,910
		8,240,843	5,085,359	1,544,343	2,924,850	3,198,106
flipTrivium^1-5b	1024	10,415,930	4,198,679	2,129,999	574,205	4,614,801
		7,544,803	3,311,930	5,358,554	7,173,632	8,094,079
flipTrivium^1-5c	1024	4,098,635	814,171	2,676,437	1,307,209	9,133,149
		4,399,240	1,918,301	261,533	275,668	2,293,057
flipTrivium^2-5a	1024	1,835,626	274,061	3,535,475	5,169,350	9,383,970
		8,029,002	6,689,532	1,620,776	2,275,871	2,523,650
flipTrivium^2-5b	1024	8,630,448	1,125,418	2,216,831	18,841,166	5,121,287
		9,500,167	4,041,119	1,821,558	2,306,187	2,430,590
flipTrivium^2-5c	1024	9,775,376	5,407,553	1,868,878	8,299,376	2,262,725
		1,564,603	540,409	766,243	601,695	871,102
flipTrivium^1-7a	128	1,390,964	2,160,538	588,344	2,181,534	2,838,657
		327,124	844,717	50,977	1,818,215	2,122,011
flipTrivium^1-7b	128	693,930	3,551,574	3,051,549	2,598,448	711,483
		1,644,204	718,348	2,145,818	532,499	2,023,250
flipTrivium^1-7c	128	2,882,464	1,700,157	1,651,605	80,036	1,503,789
		871,265	1,556,970	1,169,810	784,857	1,023,767
flipTrivium^2-7a	128	944,457	3,006,734	440,949	5,364,033	1,166,845
		599,254	3,762,563	193,534	60,172	1,832,327
flipTrivium^2-7b	128	1,371,784	979,226	3,199,842	2,200,522	418,699
		2,809,721	259,489	608,118	1,085,331	805,917
flipTrivium^2-7c	128	827,128	56,480	611,855	1,367,205	2,958,742
		945,055	1,134,094	2,142,660	1,700,628	627,671

4 Mathematical Algorithms

We start this section with Hensel lifting in Sect. 4.1 as it allows for efficient computation of the multiplicative inverse in $\mathrm{d}\underset{\cdot}{\mathrm{l}}\underset{\cdot}{\mathrm{l}}\mathrm{\Xi}\curvearrowleft Fields$. As solving linear equations is a basic building block for cryptographic schemes, we investigate them deeper in the sequel. In particular, we consider the matrix-vector equation $Mx = b$ for some matrix $M \in \tilde{\mathbb{N}}^{s\times s}$ and vectors $b, x \in \tilde{\mathbb{N}}^s$ with some odd integer $s \in \mathbb{N}$. In Sect. 4.2 we use a heuristic adaptation of Gauss elimination. While working empirically for $s < 10$, the algorithm completely fails for larger choices of s. In essence, this leads to a special form of matrices, called a trimmed LU form to allow invertability of the equation $Ax = b$ for some special matrix A. More details are given in Sect. 4.5. To get a better understanding of $\mathrm{d}\underset{\cdot}{\mathrm{l}}\underset{\cdot}{\mathrm{l}}\mathrm{\Xi}\curvearrowleft Fields$, we look at Linear (Sect. 4.6) and Non-Linear Feedback Shift Registers (Sect. 4.7). In addition, we investigate the polynomial ring mod $X^N + 1$ for some $N \in \mathbb{N}, N > 0$ in Sect. 4.8 and polynomial multiplication (Sect. 4.9).

4.1 Hensel Lifting

We use the Hensel lifting for computing the multiplicative inverse in a $\mathrm{d}\underset{\cdot}{\mathrm{l}}\underset{\cdot}{\mathrm{l}}\mathrm{\Xi}\dot{Int}s$, *i.e.* $a^{-1} \pmod{2^d}$ for some $d \in \mathbb{N}, d > 0$, cf. [BS96]. We start with a simple observation that is the core of the overall idea of Hensel lifting.

Lemma 2. *For some $k, n \in \mathbb{N}$ with $1 \le k \le n$, some function $f(X) : \mathbb{N} \to \mathbb{N}$ and some natural number $a \in \mathbb{N}$, if we have the solution $X \in \mathbb{N}$ for the equation*

$$f(X) \equiv a \pmod{2^n}.$$

This implied that

$$f(X) \equiv a \pmod{2^k}.$$

is also a solution.

Proof. Consider

$$f(X) \equiv a \pmod{2^n} \Leftrightarrow 2^n \mid (f(X) - a) \Leftrightarrow 2^{n-k}2^k \mid (f(X) - a)$$
$$\Rightarrow 2^k \mid (f(X) - a) \Leftrightarrow f(X) \equiv a \pmod{2^k}.$$

□

Note that we did not make any assumption on the function $f(X)$ in this lemma. Now, based on this observation, we use Lem. 2 for the polynomial $f(x) := aX - 1$ to iteratively compute the multiplicative inverse of $a \in \mathbb{N}$ modulo 2^n. We will improve on this result later on, but we use Lem. 3 as an intermediate step to illustrate the overall principle.

Lemma 3. *Let $a \in \mathbb{N}$ be an odd number. Moreover, we have $k, n \in \mathbb{N}$ with $1 \le k \le n$. Then we can compute $a^{-1} \pmod{2^n}$ by refining the result bit-wise, starting with $a^{-1} \pmod{2^k}$.*

Proof. Let $S^k := \{X \in \mathbb{N}, 0 \le X < 2^k : f(X) \equiv 0 \pmod{2^k}\}$ the set of all solutions for this equation modulo 2^k. We will show that $|S^k| = 1$.

We start with $k = 1$ and hence S^1 is modulo 2. As a is odd, so is a^{-1} as all computations are performed in the multiplicative group modulo 2. Hence we have $|S^1| = 1$.

For $k \to k + 1$ we start with $|S^k| = 1$. Each new element of S^{k+1} must also satisfy the equation modulo 2^k. Hence, it has the form X or $X + 2^{k+1}$, respectively. As computations are performed in the integer ring modulo 2^{k+1}, the number a^{-1} is unique, so we either have

$X \in S^{k+1}$ or $X + 2^{k+1} \in S^{k+1}$, but not both. Hence we have $|S^{k+1}| = 1$ throughout the induction.

This way, we gain one more bit of information through each iteration. Hence, starting with k, we need to repeat the above procedure $(n-k)$ times to gain all n bits of the inverse a^{-1} modulo 2^n over the natural numbers. □

While this proof shows the overall principle, it yields a rather slow variant of computing the inverse a^{-1} modulo 2^n. In particular, we only gain one extra bit of accuracy for each iteration. We can improve over this result using Hensel lifting.

Lemma 4. *Let $f(X)$ be a polynomial over the natural numbers. In addition, we have $k, n \in \mathbb{N} : 1 \leq k \leq n$. With $f'(X)$, we denote the derivative of $f(X)$. Consider some $a \in \mathbb{N}$ such that*

$$f(a) \equiv 0 \pmod 2 \text{ but } f'(a) \not\equiv 0 \pmod 2, \text{ i.e. } f'(a) \equiv 1 \pmod 2$$

Or to rephrase this, $f'(a) \in \mathbb{N}$ is an odd number. Now, this lemma states that there exists some natural number $b \in \mathbb{N} : b \equiv a \pmod{2^n}$ such that

$$f(b) \equiv 0 \pmod{2^{k+n}}.$$

We can construct this number b using some natrual number $c \in \mathbb{N}$ with $c \equiv (f'(a))^{-1} \pmod 2^k$ and $b := a - f(r)c$.

Proof. We start with $f(a_0) \equiv 0 \pmod 2$ as our induction hypothis. We want a_{k+1} such that $a_{k+1} := a_k + t2^k$ for some integer $t \in \mathbb{N}$. As $f(X)$ is a polynomial, we can use $f(a_k) + t2^k f'(a_k) + 2^{2k}$(higher terms). Note that modulo (2^{k+1}), the higher terms vanish since $(2^k \geq k+1)$ for $(k > 0)$. Hence we have the equation $f(a_k + tp^k) \equiv f(a_k) + t2^k f'(a_k) \pmod{2^{k+1}}$.

Using the induction hypothesis $f(a_k) \equiv 0 \pmod{p^k}$, we can write $f(a_k) := 2^n s$ for some integer $s \in \mathbb{N}$. Substituting, we obtain $f(a_{k+1}) \equiv 2^k s + t2^k f'(a_k) \pmod{2^{k+1}}$. Now, we factor 2^k using $f(a_{k+1}) \equiv 2^k(s + tf'(a_k)) \pmod{2^{k+1}}$.

To compute some $t \in \mathbb{N}$, we need $f(a_{k+1}) \equiv 0 \pmod{p^{k+1}}$. Therefore we have $s + tf'(a_n) \equiv 0 \pmod 2$ and hence $f'(a_k) \equiv f'(a_0) \not\equiv 0 \pmod 2$. We note that $f'(a_0)$ has a multiplicative inverse $\pmod 2$—which means we have $f'(a_0) = 1 \pmod 2$ and $f'(a_0)$ is odd. Thus we can solve $t \equiv -s(f'(a_k))^{-1} \pmod 2$ and we obtain a_{k+1} via $a_{k+1} := t \equiv -s(f'(a_k))^{-1} \pmod 2$.

We now "lift" the solution a_k to a_{k+1}. We start with $a_1 := a_0$ and extend this to $f(a_k) \equiv 0 \pmod{2^k}$.

Note that this solution is unique. Consider $v, w \in \mathbb{N}$, both being solutions modulo p^{k+1} with $v \equiv w \pmod{2^k}$. Now we can write $v = w + t2^n$. We show that this is a contradiction. Using the Taylor expansion we obtain $f(v) - f(w) \equiv t2^k f'(w) \equiv 0 \pmod{2^{k+1}}$. Dividing by 2^k yields $tf'(w) \equiv 0 \pmod 2$. Since $f'(w) \not\equiv 0 \pmod 2$ we get $t \equiv 0 \pmod 2$. Thus we have $v \equiv w \pmod{p^{k+1}}$, so the solution is unique. □

Using Lem. 4, we compute the inverse element $a^{-1} \pmod{2^n}$ more efficiently that using the Extended Euclidean algorithm.

Lemma 5. *Let $a \in \mathbb{N}$ be an odd number and $n \in \mathbb{N}, n > 0$ the exponant for the multiplicative group modulo 2^n. Given some partial solution $r \in \mathbb{N} : 0 < r < 2^k$ for $k \in \mathbb{N}, k > 0$, we can extend it to r' with $0 < r' < 2^{2k}$.*

Proof. Consider the polynomial $f(X) := aX - 1$ over the natural numbers and let $q := 2^k$ be some power of 2. The derivative $f'(X)$ is given by $f'(X) = a$. As a is an odd number, we have $f(a) \not\equiv 0 \pmod q$, so the prerequisite of Lem. 4 apply.

We now search for some natural number $s \in \mathbb{N}$ with $f(s) \equiv 0 \pmod{q^2}$. Using the above lemma, we have $c \equiv (f'(r))^{-1} \equiv a^{-1} \pmod q$ and can compute s via $s := r - f(r)c$. Substituting, we obtain $s \equiv r - f(r)c \equiv r - (ar - 1)r \equiv (2 - ar)r \pmod{q^2}$. □

Using this lemma repeatedly, we have a very efficient algorithm for computing the multiplicative inverse over a given $\tilde{\mathbb{N}}^d$.

The corresponding algorithm has been implemented as the function `invHensel`. To illustrate the above lemma, we look at the following example for $a = 105,423$ and $D = 21^{20}$, *i.e.* $d = 19$ and output the intermediate state of the Python function `invHensel`. Here we obtain for Hensel lifting

r=1	k=2
r=3	k=3
r=7	k=5
r=15	k=10
r=815	k=20

The algorithm yields the correct result $a^{-1} = 363,311$. In comparison, we use the Extended Euclidean Algorithm (EEA) to obtain, see function `invMul` for the algorithm in Python. Plotting the intermediate steps we get

q=9	r=105,423
q=1	r=99,769
q=17	r=5,654
q=1	r=3,651
q=1	r=2,003
q=1	r=1,648
q=4	r=355
q=1	r=228
q=1	r=127
q=1	r=101
q=3	r=26
q=1	r=23
q=7	r=3
q=1	r=2
q=2	r=1

In a nutshell, we can expect one extra bit for each iteration of the EEA, while we have a quadratic increase on our solution using Hensel lifting. Therefore, $\mathrm{d}\dot{\mathrm{i}}\mathrm{l}\mathrm{F}\dot{I}n\dot{t}s$ are expected to outperform Finite Fields when computing the multiplicative inverse in the corresponding structure.

Leaving the univariate case, we can also utilize Lem. 2 in the multivariate case, *i.e.* we can build our solution "bit-by-bit" for a system of equations. Note that there is a construction lifting solutions by squaring, using specific systems of equations [AG24]. However, our construction does not assume any structure for the overall system of equations $\mathcal{S}$ and might therefore be advantageous in practice.

Corollary 1. *Let $k, m, n \in \mathbb{N}$ be natural numbers with $1 \leq k \leq n$. In addition, consider partial solutions $x_1, \ldots, x_m \in \mathbb{N}$ modulo 2^k with $0 \leq x_j \leq 2^k$ for $0 \leq j < m$. Finally, consider function over the natural numbers $f_i(x_i) : \mathbb{N} \to \mathbb{N}$. The overall system of equations is now*

$$\mathcal{S}^k := \begin{cases} f_0(x_0) & \equiv 0 \pmod{2^k} \\ & \vdots \\ f_{m-1}(x_{m-1}) & \equiv 0 \pmod{2^k} \end{cases}$$

and

$$\mathcal{S}^n := \begin{cases} f_0(x_0) & \equiv 0 \pmod{2^n} \\ & \vdots \\ f_{m-1}(x_{m-1}) & \equiv 0 \pmod{2^n}. \end{cases}$$

Using time $O((n-k)2^m)$, we can construct a solution for $\mathcal{S}^n$, given a solution for $\mathcal{S}^k$.

Proof. We note that a solution $\pmod{2^{i+1}}$ is also a solution $\pmod{2^i}$ by virtue of Lem. 2. Hence, for $\mathcal{S}^i$, we can extend each initial solutions x_j to two potential solutions x_j and $x_j + 2^i$, respectively, for $0 \leq j < m$. The overall workload of this operation is $O(2^m)$ as we have a total of m equations. As we have to perform this operation a total of $(n-k)$ times, the overall workload becomes $O((n-k)2^m)$, which yields the result. □

Note that one initial solution for $\mathcal{S}^k$ can lead up to 2^{n-k} solutions for the system $\mathcal{S}^n$. If we are able to limit the number of potential solutions to at most one per polynomial equation $f_j(x_j) \equiv 0 \pmod{2^i}$ in each step, we can improve the result of Cor. 1 to a time complexity of $O(n-k)$.

In addition, we can also apply this lemma to the univariate setting, *i.e.* we have $x_1 = \ldots = x_m$. In this case, the workload drops to $O(n-k)$. Obviously, there are also intermediate cases when several $x_i = x_j$ for $0 \leq i < j < m$. The overall workload changes accordingly. In practice, we can evaluate the different functions f_i for $0 \leq i < m$ independently, which will yield a notably speed-up when using multiple processors.

All of the above statements are also valid for $\mathrm{d}_{\text{IH}}\check{P}olys$ rather than $\mathrm{d}_{\text{IH}}\dot{Int}s$, the corresponding algorithms and proofs can be adjusted accordingly. In particular, *invHensel* has been implemented and tested over $\mathrm{d}_{\text{IH}}\check{P}olys$, too.

uəd$_{\mathbf{O}}$ P̆roblem 10.

i. How efficient is the computation of the multiplicative inverse for $\mathrm{d}_{\text{IH}}\dot{Int}s$ and $\mathrm{d}_{\text{IH}}\check{P}olys$on different plattforms, when being compared with the Extended Eucledian algorithm?

ii. Are there benefits or penalties regarding side channel attacks?

4.2 Gaussian Elimination Heuristic

To see how arithmetic over $\mathrm{d}_{\text{IH}}\dot{Int}s$ works in practice, we consider a toy example. Let $\tilde{\mathbb{N}} := \tilde{\mathbb{N}}^3$ solving $Mx = b$ for some matrix $M \in \tilde{\mathbb{N}}^{3\times 3}$ and given vector $b \in \tilde{\mathbb{N}}^3$. We aim to solve for an unknown vector $x \in \tilde{\mathbb{N}}^s$ and write $(M|b)$ for the augmented matrix. As we need to generate zero elements in the intermediate maxtrix, all computations are performed in the ring over $\mathbb{N}$. We start with $D = 16$ and the followng augmended matrix:

$$\left(\begin{array}{ccc|c} 3 & 1 & 1 & 1 \\ 3 & 3 & 7 & 5 \\ 5 & 5 & 5 & 7 \end{array}\right) \rightsquigarrow \left(\begin{array}{ccc|c} 1 & 11 & 11 & 11 \\ 0 & 2 & 6 & 4 \\ 0 & 14 & 14 & 0 \end{array}\right) \rightsquigarrow \left(\begin{array}{ccc|c} 1 & 11 & 11 & 11 \\ 0 & 1 & 3 & 2 \\ 0 & 1 & 1 & 0 \end{array}\right)$$

We begin by multiply the first row with 11 as $3.11 \equiv 1 \pmod{16}$ to obtain a leading one. Second, we subtract the first row from the second and third row, respectively. We observe that the gcd of the second row is 2 and the gcd of the third row is 14. We divide accordingly to keep the number of zero-divisors in the ring $\pmod{16}$ small. We continue this example with

$$\rightsquigarrow \left(\begin{array}{ccc|c} 1 & 11 & 11 & 11 \\ 0 & 1 & 3 & 2 \\ 0 & 0 & 14 & 14 \end{array}\right) \rightsquigarrow \left(\begin{array}{ccc|c} 1 & 11 & 11 & 11 \\ 0 & 1 & 3 & 2 \\ 0 & 0 & 1 & 1 \end{array}\right)$$

after subtracting the second row from the third row, we again obtain a gcd of 14. Dividing the third row by 14 yields an upper rectangular matrix. Using backwards substitution leads to:

$$\begin{cases} x_3 = 1, & \\ x_2 = 15 & \equiv 2 - 3.1 \\ x_1 = 11 & \equiv 11 - 11.1 - 11.15 \end{cases}$$

which yields the vector $x^T = (11, 15, 1)$.

For details on a Python see App. B.1 for a quick glance. The corresponding code can be found there under the name `solveMatVec`. From a bird's eyes view, the algorithm consists of the following steps:

i. Check if the leading term of the current row is odd. If not, swap rows until this condition is met

ii. Multiply the leading term t of the current row by its multiplicative inverse t^{-1} so the leading term becomes 1

iii. Subtract the current from all rows with higher index.

iv. For each upper rows: reduce all elements by the gcd of this row

v. Using backwards substitution, derive the result vector x

While this algorithm looks good at first glance, it does not cover all possible cases. Considering a different example, we see that

$$\left(\begin{array}{ccc|c} 3 & 1 & 1 & 1 \\ 1 & 3 & 1 & 5 \\ 1 & 1 & 3 & 7 \end{array}\right) \rightsquigarrow \left(\begin{array}{ccc|c} 1 & 11 & 11 & 11 \\ 0 & 8 & 6 & 10 \\ 0 & 6 & 8 & 12 \end{array}\right) \rightsquigarrow \left(\begin{array}{ccc|c} 1 & 11 & 11 & 11 \\ 0 & 3 & 4 & 6 \\ 0 & 4 & 3 & 5 \end{array}\right)$$

Starting with our input, we first multiply the first row with 11 as $3.11 \equiv 1 \pmod{16}$. Second, we subtract the first row from the second and the third, respectively. For the last step we observe that both the second and the third row have a gcd of 2, so we can safely divide each row by this number. This way, we reduce the number of zero divisors in the result. In addition, we need to swap the second and the third row as 4 is an even number. We continue as follows:

$$\rightsquigarrow \left(\begin{array}{ccc|c} 1 & 11 & 11 & 11 \\ 0 & 1 & 12 & 2 \\ 0 & 0 & 3 & 13 \end{array}\right) \rightsquigarrow \left(\begin{array}{ccc|c} 1 & 11 & 11 & 11 \\ 0 & 1 & 12 & 2 \\ 0 & 0 & 1 & 15 \end{array}\right)$$

We multiply the second row by 11 and obtain 1 on the diagonal. After subtracting the second from the third row, we also multiply the third row with 11. Using back substitution, we obtain

$$\begin{cases} x_3 = 15, & \\ x_2 = 14 & \equiv 2 - 15.12 \\ x_1 = 12 & \equiv 11 - 15.11 - 14.11 \end{cases}$$

which yields the vector $x^T = (12, 14, 15)$. Obviously, the first two components are even numbers and hence are no elements of $\tilde{\mathbb{N}}^3$ and are hence parasiteric solutions. So in a sense, we have to say that our second example is a non-invertible matrix over the $\mathrm{d}\mathrm{i}\mathrm{l}\mathrm{F}\mathit{Int}\dot{s}\ \tilde{\mathbb{N}}^3$ although we obtain a unique solution over the integer ring.

All in all, this heuristic clearly fails for cryptographically relevant parameters, see App. B.1 for a more detailed analysis using empirical data.

4.3 Number of Solutions for $Ax = b$

In this section, we look at the number of solutions that we obtain for the affine equation $Ax = b$ for a matrix $A \in \tilde{\mathbb{N}}^{s\times s}$ and $b \in \tilde{\mathbb{N}}^s$. The equation is defined over the dịlƎ*Inṫs*with dimension $d \in \mathbb{N}, d \geq 2$.

Lemma 6. *Consider a dịlƎInṫs of dimension d and a matrix A of dimension s. Both $d, s \geq 1$. Let $Ax = b \pmod{2^{d+1}}$ have at least one solution $x^0 \in \tilde{\mathbb{N}}^s$. Then the minimal number of solutions is exactly*

$$2^{s-1}\,.$$

Proof. Let $x^{(0)} \in \tilde{\mathbb{N}}^s$ be one solution. Consider another solution $x \in \tilde{\mathbb{N}}^s$. As both $x^{(0)}$ and x are componentwise odd, their difference is componentwise even. Hence we can write

$$x = x^{(0)} + 2y \text{ for some unique } y \in \mathbb{Z}_{2^d}^s\,.$$

Substituting into $Ax = b$ yields

$$A(x^{(0)} + 2y) = b \pmod{2^{d+1}}\,.$$

Since $Ax^{(0)} = b \pmod{2^{d+1}}$, this is equivalent to

$$2Ay = 0 \pmod{2^{d+1}}$$

and also to

$$Ay = 0 \pmod{2^d}\,.$$

Thus the solutions of the original dịlƎ*Inṫs* system are in one-to-one correspondence with the elements of

$$\ker\left(A : \mathbb{Z}_{2^d}^s \to \mathbb{Z}_{2^d}^s\right).$$

It remains to determine a lower bound on the size of this kernel.

Let $a_1, \ldots, a_s \in \mathbb{Z}_{2^d}^s$ denote the columns of A, after the modular reduction via 2^d. Since every entry of A is odd, all columns are componentwise congruent modulo 2. Consequently, for every $j = 2, \ldots, s$ there exists a vector $w_j \in \mathbb{Z}_{2^d}^s$ such that

$$a_j = a_1 + 2w_j \pmod{2^d}\,.$$

For an arbitrary vector

$$y = (y_1, \ldots, y_s)^T \in \mathbb{Z}_{2^d}^s$$

we therefore have

$$Ay = \left(\sum_{j=1}^{s} y_j\right) a_1 + 2\sum_{j=2}^{s} y_j w_j \pmod{2^d}.$$

Hence the image of A is contained in the subgroup generated by

$$a_1, \quad 2w_2, \ldots, 2w_s.$$

The element a_1 has order at most 2^d, while each element $2w_j$ has order at most 2^{d-1}. Therefore

$$|\operatorname{im}(A)| \leq 2^d \left(2^{d-1}\right)^{s-1} = 2^{ds-s+1}.$$

Since $\mathbb{Z}_{2^d}^s$ has 2^{ds} elements, the kernel-image relation for finite abelian groups gives

$$|\ker(A)| = \frac{2^{ds}}{|\operatorname{im}(A)|}.$$

Consequently,

$$|\ker(A)| \geq \frac{2^{ds}}{2^{ds-s+1}} = 2^{s-1}.$$

Thus every solvable dịlƎ*Inṫs* system has at least

$$2^{s-1}$$

solutions.

It remains to show that this bound can be attained by giving a specific matrix A that yields this number of solutions. Consider

$$A := J + \operatorname{diag}(2, \ldots, 2, 4).$$

All entries of A are odd. Moreover,

$$\det(A) = 2^{s+1}\left(1 + \frac{2s-1}{4}\right) = 2^{s+1}\frac{2s+3}{4} = 2^{s-1}(2s+3),$$

where $2s+3$ is odd. Hence the kernel of A modulo 2^d has exactly 2^{s-1} elements. Taking

$$x^{(0)} = 1^s, \qquad b = A1^s \pmod{2^{d+1}},$$

gives a solvable dịlƎ*Inṫs* system with exactly 2^{s-1} solutions.

Therefore the minimal number of solutions is 2^{s-1}. □

Lemma 7. *Consider a dịlƎ*Inṫs *of dimension d and a matrix A of dimension s. So we have $A \in \tilde{\mathbb{N}}^{s\times s}$ and $b\tilde{\mathbb{N}}^s$. Both $d, s \geq 1$ and s is odd. Then the maximal number of solutions is exactly*

$$2^{d(s-1)}.$$

Proof. Recall that each $x \in \tilde{\mathbb{N}}^s$ can be written as

$$x = 1^s + 2z \text{ with } z \in \mathbb{Z}_q^s$$

and note that this actually a bijection. Thus

$$A(1^s + 2z) = b \pmod{2q}$$

is equivalent to

$$Az = c \pmod{q} \text{ for some } c = \frac{b - A1^s}{2} \pmod{q}$$

Note that all coefficients of A, b are odd, hence their difference is even and we can factor our 2. So the number of dịlƎ*Inṫs* solutions x is exactly the number of solutions z of this linear system modulo q.

No consider the linear map

$$\phi : R^s \to R^s, \qquad z \mapsto Az \pmod{q}.$$

As the coefficient $a_{0,0}$ of the matrix A is invertible in R, this map has an image containing at least q elements.

Indeed, consider the q vectors

$$z(t) := \begin{pmatrix} t \\ 0 \\ \vdots \\ 0 \end{pmatrix}, \qquad t \in R\,.$$

Their images are

$$\phi(z(t)) = \begin{pmatrix} a_{0,0}t \\ a_{1,0}t \\ \vdots \\ a_{s-1,0}t \end{pmatrix}.$$

Consider two different scalars $t_1, t_2 \in \mathbb{Z}_q^s$. Now looking only at their first component we have

$$\phi(z(t_1)) = \phi(z(t_2)) \Rightarrow a_{0,0}t_1 = a_{0,0}t_2 \pmod{q}\,.$$

As $a_{0,0}$ is odd, it is invertible modulo $q = 2^d$. Hence we have

$$t_1 = t_2$$

Thus all of the vectors $\phi(z(t))$ are distinct and consequently

$$|\text{im } \phi| \geq q\,.$$

As R^s has exactly q^s elements, we know that

$$|\ker \phi| \cdot |\text{im}\phi| = q^s\,.$$

Therefore

$$|\ker \phi| = \frac{q^s}{|\text{im } \phi|} \leq \frac{q^s}{q} = q^{s-1}\,.$$

Assume $Az = c$ is solvable, then its solution set is a coset of $\ker \phi$. Hence it contains exactly

$$|\ker \phi|$$

solutions, and therefore at most q^{s-1} solutions.

As $x \mapsto z$ is the bijection. this means that the number of solutions of the original $\text{dlI}\mathbb{F}\dot{Int}s$ equation is at most

$$q^{s-1} = 2^{d(s-1)}\,.$$

We now show that this bound can actually be attained. Consider the all-one matrix J and set the matrix A to J. This means

$$A := J = \begin{pmatrix} 1 & 1 & \dots & 1 \\ 1 & 1 & \dots & 1 \\ \vdots & \vdots & \ddots & \vdots \\ 1 & 1 & \dots & 1 \end{pmatrix}$$

This means we have

$$Az = \begin{pmatrix} z_0 + \ldots + z_{s-1} \\ \vdots \\ z_0 + \ldots + z_{s-1} \end{pmatrix}$$

If we inspect this equation more closely, we see that it has indeed exactly q^{s-1} solutions: Choose $z_0, \ldots, z_{s-2}$ arbitrarily and therefore uniquely determine

$$z_{s-1} = -\sum_{i=0}^{s-2} z_i \pmod{q}\,.$$

As s is odd, taking

$$b = A1^s$$

give a valid dịlƎ*Inṫs* vector. Thus the original equation has exactly q^{s-1} solutions and the bound $2^{d(s-1)}$ for the equation $Ax = b$ is attained. □

We summarize the results of the previous two lemmata in the following

Corollary 2. *Let $d \geq 1$ be the dimension of some dịlƎ*Inṫs*. Let $s \in \mathbb{N}, s \geq 1$ and s be odd. Consider the matrix $A \in \tilde{\mathbb{N}}^{s\times s}$, the vector $b \in \tilde{\mathbb{N}}^s$ and the matrix-vector-equation*

$$Ax = b\,.$$

Now consider the number of solutions $S_{A,b}$. We have

$$2^{s-1} \leq S_{A,b} \leq 2^{d(s-1)}\,.$$

In Sect 4.4 we will see that $S_{a,b}$ actually always is a power of two, so we can amend the result as

$$2^{s-1} \leq S_{A,b} \leq 2^{d(s-1)}\,.$$

and there is some $k \in \mathbb{N}$ such that $2^k = S_{A,b}$ in Cor. 2.

Lemma 8. *Consider a dịlƎ*Inṫs *of dimension d and a matrix A of dimension s. Choose randomly uniform $A \in_R \tilde{\mathbb{N}}^{s\times s}$ and $b \in \tilde{\mathbb{N}}^s$. Both $d, s \geq 1$ and s is odd. For $d \geq 2$ the solvability probability is*

$$\begin{aligned} Pr[Ax = b \text{ has no solution}] = 1 - 2^{-(d-1)(s-1)} \\ - 2^{-(s-1)} \sum_{r=1}^{s-1} P_{s-1,r}(2^r - 1) \\ \cdot 2^{-(d-2)(s-1-r)} \frac{1 - 2^{-r(d-1)}}{1 - 2^{-r}}. \end{aligned}$$

where for $m \geq 1$ we have

$$P_{m,r} := \frac{1}{2^{m^2}} \frac{\left(\prod_{i=0}^{r-1}(2^m - 2^i)\right)^2}{\prod_{i=0}^{r-1}(2^r - 2^i)} \text{ for } 0 \leq r \leq m\,.$$

For $d = 1$ and $s > 1$, this probability is

$$1 - 2^{-(s-1)}.$$

For $s = 1$, the probability is 0.

Proof. Put $m = s-1, k = d-1$. For $0 \le r \le m$, define

$$P_{m,r} := \frac{1}{2^{m^2}} \frac{\left(\prod_{i=0}^{r-1}(2^m - 2^i)\right)^2}{\prod_{i=0}^{r-1}(2^r - 2^i)} \text{ for } 0 \le r \le m\,.$$

Here $P_{m,r}$ is the probability that a uniformly random $(m \times m)$ binary matrix has rank r.

For $d \ge 2$, the solvability probability is

$$f_{s,d} := 2^{-km} + 2^{-m} \sum_{r=1}^{m} P_{m,r}(2^r - 1) \cdot 2^{-(k-1)(m-r)} \frac{1 - 2^{-rk}}{1 - 2^{-r}}.$$

We observe that

$$\Pr[Ax = b \text{ has no solution}] = 1 - f_{s,d}$$

which completes the proof for $d \ge 2$ and $s \ge 2$.

We now deal with the special cases.

Let $s = 1$. In this case, we only have one linear equation $a_{0,0}x = b_0$. As $a_{0,0}$ is invertible, this is equivalent to $a_{0,0}x = b_0 a_{0,0}^{-1}$ so there is always exactly 1 solution.

Now let $d = 1$. Here we are actually working over a binary matrix so

$$\Pr[Ax = b \text{ has no solution}] = 1 - 2^{-(s-1)}\,.$$

□

Plotting for specific values we obain

s	$d = 2$	$d = 4$	$d = 8$
3	0.328125	0.553711	0.620602
5	0.375524	0.618526	0.687870
11	0.389463	0.636721	0.706410
15	0.389665	0.636982	0.706674
21	0.389678	0.636999	0.706691
41	0.389678	0.636999	0.706692

Note that the assumption that the matrix/vector dimension s is odd is essential. If s were even, Ax would be even componentwise for every $A, x \in \tilde{\mathbb{N}}$, whereas b is componentwise odd, so the probability of a solution would simply be 0.

4.4 SubstSolver

Taking a different angle, we observe that each element of $\mathrm{d}\dot{\mathrm{i}}\mathrm{l}\mathrm{\exists}\mathit{Int}\dot{s}$ $\alpha \in \tilde{\mathbb{N}}^d$ can be written as $\alpha = 2\beta + 1$ for some integer $\beta \in \mathbb{Z}_{2^d}$. As this substitution is central for the algorithm, we call it **SubstSolver**. There are two main points here: First, this representation is unique for a given $\alpha \in \tilde{\mathbb{N}}^d$ and second, there is no further restriction on β. As usual, we denote $D = 2^{d+1}$ and $q = 2^d$.

Let $M \in (\tilde{\mathbb{N}}^d)^{sxs}$ with $s \in \mathbb{N}$, s odd be a matrix and $b \in (\tilde{\mathbb{N}}^d)^s$ a vector. Consider the matrix-vector equation $Mx = b$ over $\mathrm{d}\dot{\mathrm{i}}\mathrm{l}\mathrm{\exists}\mathit{Int}\dot{s}$. We solve for x. Denote with 1^s the all-one-vector of dimension s and let $x = 2z + 1^s$ for $z \in \mathbb{Z}_{2^d}^s$. Substituting $x = 2z + 1^s$ into the matrix-vector equation yields

$$M(1^s + 2z) = b \pmod{D}\ .$$

Hence

$$M1^s + 2Mz = b \pmod{D} \ ,$$

or

$$2Mz = b - M1^s \pmod{D} \ .$$

Since M has odd dimension s and all entries in M are odd, all entries in the vector $M1^s$ will be odd. Similarly, all entries of b are odd. Therefore, the vector $b - M1^s$ contains only even entries. We can therefore define a new vector c as

$$c := \frac{b - M1^s}{2} \pmod{q} \ .$$

Hence, the original matrix-vector equation becomes

$$Mz = c \pmod{q} \ .$$

Define $A := M \pmod{q}$, *i.e.*reduce M by q instead of D. This leads to the central equation

$$Az = c \pmod{q} .$$

over $\mathbb{Z}_q$. For each entry of A, c we have $0 \leq a_{i,j}, c_i < q$ for $0 \leq i, j < s$. Our goal is now to directly solve $Az = c$ over the integers modulo 2^d.

Solving $Az = c$ over Z_q. To keep things simple, we will solve this matrix-vector-equation using 2-adic Gaussian elimination, pivoting odd entries from the matrix A. Denote this element with $a_{i,j} \in \mathbb{Z}_q$. As $a_{i,j}$ is odd, we can compute $a_{i,j}^{-1}$ modulo 2^d. After that, we can perform ordinary Gaussian-elimination operations using this pivot $a_{i,j}$. We walk to the process step by step.

Suppose that we found some $a_{i,j} \in \mathbb{Z}_q$ at elimination step k in the remaining submatrix. After exchanging rows i and k and columns j and k, we have obtained an odd pivot

$$p_k := a_{k,k} .$$

As p_k is odd, we can compute its inverse p_k^{-1}, for example using the Extended Euclidean Algorithm or the Hensel Lifting modulo 2^d. Note that we also need to exchange elements in the solution vector z if we perform column changes.

Denote with R_k the kth row of the matrix A. Using p_k, we can normalize the pivot row by

$$R_k \leftarrow p_k^{-1} R_k \pmod{q} \ .$$

Note that this yields $a_{k,k} = 1$. For every remaining row $i > k$, we now eliminate the kth entry using

$$R_i \leftarrow R_i - a_{i,k} R_k \pmod{q} \ .$$

As $a_{k,k} = 1$ the above expression eliminates $a_{i,k}$ in R_k.

Repeating this process produces an upper-triangular matrix U where every pivot of U is 1. Our main equation now becomes

$$Uz = c' \pmod{q}$$

where U is the upper-triangular matrix after pivoting and exchanging rows while $c' \in \mathbb{Z}_q^s$ is the original vector c after exchanging rows.

With these definitions, we can easily backtrack the solution iterating from $z_(s-1), \ldots, z_0$ and computing along the way

$$z_k = c'_k - \sum_{j=k+1}^{s-1} u_{k,j} z_j \pmod{\text{q}} .$$

No odd pivot. The above algorithm works as long as an odd pivot is available. However, since neither $\mathbb{Z}_q$ nor $\mathbb{Z}_q$ is a field, we may encounter a remaining submatrix in which all entries are even.

This does not necessarily imply that the system has no solution. Instead, we can exploit the 2-adic structure of $\mathbb{Z}_q$.

Assume that elimination has been performed up to step $(k-1)$. We consider the remaining system as

$$A_k z' = b' \pmod{2^d},$$

where the submatrix A_k contains the remaining rows and columns, z' is the corresponding subvector of unknowns and b' are our remaining knowns.

Suppose that all entries of A_k and b' are divisible by 2^r for some $r \in \mathbb{N}, r \geq 1$. We write

$$A_k = 2^r \widetilde{A}_k, \qquad b' = 2^r \widetilde{b}'.$$

Then

$$2^r \widetilde{A}_k z' = 2^r \widetilde{b}' \pmod{2^d}$$

is equivalent to

$$\widetilde{A}_k z' = \widetilde{b}' \pmod{2^{d-r}}.$$

If r is chosen maximally, at least one of the entries of $\widetilde{A}_k, \widetilde{b}'$ is odd.

We first assume that both A_k, b allow a factorization by 2^r. the reduced system contains at least one odd coefficient in $\widetilde{A}_k$, unless the remaining matrix is zero. We can therefore continue the elimination over the smaller ring $\mathbb{Z}_{2^{d-r}}$.

If, on the other hand, some component of b' is not divisible by 2^r while all other coefficients in the corresponding equation are divisible by 2^r, then that equation is inconsistent and the original system has no solution.

Since the reduced system is defined modulo 2^{d-r}, a solution z' may have multiple lifts to $\mathbb{Z}_{2^d}$. Such lifts have the form

$$z_i = z'_i + t_i 2^{d-r}, \qquad t_i \in \{0, \ldots, 2^r - 1\}.$$

These lifts are not necessarily all solutions of the original system and must be checked against the remaining equations. Note Def. 5 on conjugates in this context.

Lifting strategy. From a complexity point of view, a naive enumeration of all possible lifts can become exponential in the worst case. Consequently, an efficient implementation requires a suitable lifting strategy rather than exhaustive enumeration of all lifts. In a nutshell, we must compute the solution starting at $\mathbb{Z}_2$ and then reconsider the equation for $\mathbb{Z}_{2^d}$ for each power of 2. This way, the algorithm uses equations modulo $\mathbb{Z}_{2^{f+1}}$. as a filter for solutions coming from $\mathbb{Z}_{2^f}$. Storing the solutions in form of a kernel rather than enumerating all solution further helps against a blow-up of the number of intermediate solutions.

We discuss this step-by-step. Suppose after factoring out 2^r we have

$$\widetilde{A} z = \widetilde{B} \pmod{2^{d-r}}.$$

Let $m := d - r$. We have a solution

$$z^{(m)} \pmod{2^m}.$$

Instead of considering all possible solutions

$$z^{(m)} + 2^m t \text{ for } t \in \{0, \ldots, 2^r - 1\}^s$$

we lift one bit at a time as follows. Assume

$$Az^{(j)} = b \pmod{2^j}$$

We now write a candidate lift modulo 2^{j+1} as

$$z^{(j+1)} = z^{(j)} + 2^j\delta \text{ with } \delta \in \{0,1\}^s\,.$$

Substituting yields

$$A(z^{(j)} + 2^j\delta) = b \pmod{2^j+1}\,.$$

Note that this way, we only consider the least significant bit from δ. More formally and as $Az^{(j)} = b \pmod{2^j}$, we can define the residual

$$r^{(j)} := \frac{b - Az(j)}{2^j} \pmod 2\,.$$

Now we can rewrite the lift condition modulo 2 as

$$A\delta = r(j) \pmod 2\,.$$

Note that this is a linear system. Assuming that either $z^{(m')}$ or $z^{(m')} + \delta$ are a solution. In this case, the lifting is straightforward as we have the following r steps $z^{(m)} \to z^{(d)}$.

However, if there are multiple solutions, we use the fact that all liftings come from linear equations. Hence, we can keep these intermediate solutions symbolically and filter them at a later state if needed. This gives an incremental system starting over GF(2) and finally reaching the corresponding system modulo 2^d.

4.5 Trimmed LU form

While not a decomposition for any given matrix, we can trim both the lower and upper matrix in LU to our needs, *i.e.* we construct an LU form that will yield a solution in any case. In a nutshell, our goal is to fulfill *ideally solvable* from Def. 13 by construction.

Let L be a lower triangular matrix of dimension s. We request that each row constains only an odd number of non-zero elements while the diagonal is zero-free. For example, the following matrix fulfills this property:

$$L := \begin{pmatrix} \star & & & & \\ & \star & & & \\ \star & \star & \star & & \\ \star & \star & & \star & \\ \star & \star & \star & \star & \star \end{pmatrix}$$

We see that each row affects exactly one variable, so for any input $b \in \tilde{\mathbb{N}}^s$, we obtain a solution $x \in \tilde{\mathbb{N}}^s$ for $Lx = b$.

Similarly, we can choose an upper triangular matrix U such that each row constains only an odd number of non-zero elements while the diagonal is zero-free. For example, the following matrix fulfills this property:

$$U := \begin{pmatrix} \star & \star & \star & \star & \star \\ & \star & & \star & \star \\ & & \star & \star & \star \\ & & & \star & \\ & & & & \star \end{pmatrix}$$

We see that each row affects exactly one variable, so for any input $b \in \tilde{\mathbb{N}}^s$, we obtain a solution $x \in \tilde{\mathbb{N}}^s$ for $Ux = b$.

Definition 3. *A matrix $A \in \tilde{\mathbb{N}}^{s\times s}$ of Trimmed LU form is called* ***maximal*** *if there is no other matrix A' with a higher number of non-zero entries.*

We see that both of the above examples are maximal under Defintion3. Alas, Trimmed LU is not a decomposition for any given matrix, as we see with a simple counting argument. Hence, there are matrices for which no decomposition into L, U and L, U trimmed can be found. This is well in line with the findings in App A.3 and App. B as an overwhelming number of matrix-vector eqtions does not have a solution.

Alas, we have a similar problem as with the multiplication of polynomials, see Sect. 4.9: It is not closed over $\dot{\mathbb{N}}$. Consider the following example, using two maximal matrices L, U:

$$L := \begin{pmatrix} l_{0,0} & 0 & 0 & 0 & 0 \\ 0 & l_{1,1} & 0 & 0 & 0 \\ l_{2,0} & l_{2,1} & l_{2,2} & 0 & 0 \\ l_{3,0} & l_{3,1} & 0 & l_{3,3} & 0 \\ l_{4,0} & l_{4,1} & l_{4,2} & l_{4,3} & l_{4,4} \end{pmatrix} U := \begin{pmatrix} u_{0,0} & u_{0,1} & u_{0,2} & u_{0,3} & u_{0,4} \\ 0 & u_{1,1} & 0 & u_{1,3} & u_{1,4} \\ 0 & 0 & u_{2,2} & u_{2,3} & u_{2,4} \\ 0 & 0 & 0 & u_{3,3} & 0 \\ 0 & 0 & 0 & 0 & u_{4,4} \end{pmatrix}$$

$$LU = \begin{pmatrix} l_{0,0}u_{0,0} & l_{0,0}u_{0_1} & l_{0,0}u_{0,2} & . \\ 0 & l_{1,1}u_{1,1} & 0 & . \\ l_{2,0}u_{0,0} & l_{2,0}u_{0,1} + l_{2,1}u_{1,1} & l_{2,0}u_{0,2} + l_{2,2}u_{2,2} & . \\ l_{3,0}u_{0,0} & l_{3,0}u_{0,1} + l_{3,1}u_{1,1} & l_{3,0}u_{0,2} & . \\ l_{4,0}u_{0,0} & l_{4,0}u_{0,1} + l_{4,1}u_{1,1} & l_{4,0}u_{0,2} + l_{4,2}u_{2,2} & . \end{pmatrix}$$

$$\begin{pmatrix} . & l_{0,0}u_{0,3} & l_{0,0}u_{0,4} \\ . & l_{1,1}u_{1,3} & l_{1,1}u_{1,4} \\ . & l_{2,0}u_{0,3} + l_{2,1}u_{1,3} + l_{2,2}u_{2,3} & l_{2,0}u_{0,4} + l_{2,1}u_{1,4} + l_{2,2}u_{2_4} \\ . & l_{3,0}u_{0,3} + l_{3,1}u_{1,3} + l_{3,3}u_{3,3} & l_{3,0}u_{0,4} + l_{3,1}u_{1,4} \\ . & l_{4,0}u_{0,3} + l_{4,1}u_{1,3} + l_{4,2}u_{2,3} + l_{4,3}u_{3,3} & l_{4,0}u_{0,4} + l_{4,1}u_{1,4} + l_{4,2}u_{2,4} + l_{4,4}u_{4,4} \end{pmatrix}$$

First we note that in Row 1 there are two zero-entries. This is not surprising as we have a total of 4 zero-entries in L, U, respectively. However, this shows a limit to our concept of maximal trimmed LU formes—they do not guarantee that their product is a full matrix over the $\dot{\mathbb{N}}$. In addition, there are sums with an even number of terms, *e.g.* in Row 2 we have $l_{2,0}u_{0,1} + l_{2,1}u_{1,1}$. In contrast to the polynomial case, we have way more degrees of freedom and can basically "sparse out" the matrices L, U. One such example is given here:

$$L := \begin{pmatrix} l_{0,0} & 0 & 0 & 0 & 0 \\ 0 & l_{1,1} & 0 & 0 & 0 \\ 0 & 0 & l_{2,2} & 0 & 0 \\ 0 & 0 & 0 & l_{3,3} & 0 \\ 0 & 0 & l_{4,2} & l_{4,3} & l_{4,4} \end{pmatrix} U := \begin{pmatrix} u_{0,0} & 0 & 0 & u_{0,3} & u_{0,4} \\ 0 & u_{1,1} & 0 & u_{1,3} & u_{1,4} \\ 0 & 0 & u_{2,2} & 0 & 0 \\ 0 & 0 & 0 & u_{3,3} & 0 \\ 0 & 0 & 0 & 0 & u_{4,4} \end{pmatrix}$$

$$LU = \begin{pmatrix} l_{0,0}u_{0,0} & 0 & 0 & l_{0,0}u_{0,3} & l_{0,0}u_{0,4} \\ 0 & l_{1,1}u_{1,1} & 0 & l_{1,1}u_{1,3} & l_{1,1}u_{1,4} \\ 0 & 0 & l_{2,2}u_{2,2} & 0 & 0 \\ 0 & 0 & 0 & l_{3,3}u_{3,3} & 0 \\ 0 & 0 & l_{4,2}u_{2,2} & l_{4,3}u_{3,3} & l_{4,4}u_{4,4} \end{pmatrix}$$

We see that all sums with an even number of terms have been eliminated while the two matrices L, U are not fully zero. This is a much better result than we could achieve for polynomials.

However, we have two conflicting goals now. First, we want to add zeros to the matrices L, U, respectively, to eliminate sums with an even number of terms in their product. Secondly,

we want to eliminate the zeros in the product, too, as they are not elements of $\mathrm{d}\underset{\cdot}{\mathrm{l}}\underset{\cdot}{\mathrm{l}}\mathrm{F}\mathit{Int}\dot{s}$ either. To achieve our goal, we introduce a *central matrix* $C \in \tilde{\mathbb{N}}^{s\times s}$ and compute our result R as

$$R := LCU\,.$$

Our goal is to ensure that all coefficients from the result R come from a $\mathrm{d}\underset{\cdot}{\mathrm{l}}\underset{\cdot}{\mathrm{l}}\mathrm{F}\mathit{Int}\dot{s}$ while the construction of L, C, U ensures solvability for $Rx = b$ for all elements $b \in \tilde{\mathbb{N}}^s$, *i.e.* all three matrices L, C, U are solvable for any vector $b \in \tilde{\mathbb{N}}^s$.

The above idea of constructing sparse matrices L, C, U that yield a matrix over $\mathrm{d}\underset{\cdot}{\mathrm{l}}\underset{\cdot}{\mathrm{l}}\mathrm{F}\mathit{Int}\dot{s}$ has been implemented in the Python method `trmRun`.

4.6 Linear Feedback Shift Registers (LFSR)

Linear Feedback Shift Registers (LFSR) are important building blocks in cryptology, *e.g.* as pseudorandom generators in stream ciphers, see [LN02, MOV18]. In general, we have the Galois configuration, where the highest bit is fed back into lower bits. In contrast, the Fibonacci configuration feeds several bits back into the highest bit. Both Galois and Fibonacci LFSR are equivalent. Alas, when adding two bits as in the Galois configuration, we cannot use a tally. Hence, we follow the idea of a Fibonacci LFSR in the remainder of this section and generalize it from bits to arbitrary elements. For more information, see [LN02, Chap. 6].

For the ease of explanation, we start our definitions over $\mathrm{d}\underset{\cdot}{\mathrm{l}}\underset{\cdot}{\mathrm{l}}\mathrm{F}\mathit{Int}\dot{s}$. For some $N \in \mathbb{N}, N > 1$, and coefficients $a_i : 0 \le i \le N$ with $a_i \in \tilde{\mathfrak{C}} \cup \{0\}$, *i.e.* the coefficient either come from the flipCarrier $\tilde{\mathfrak{C}}$ or are zero, we consider the sequence

$$R(X) := X^N + \sum_{i=0}^{N-1} r_i X^i = X^N + r_{N-1}X^{N-1} + \ldots + r_1 X + r_0\,.$$

The non-zero coefficients $r_i : 0 \le i < N$ are called **tabs**. If the number of tabs in a sequence is odd, we call this sequence a **flipSequence**. We note that $R(X)$ is monic. For some given state polynomial

$$A(X) := \sum_{i=0}^{N-1} a_i X^i\,,$$

we can generate a full sequence of elements by defining $a_{N+1} \in \tilde{\mathfrak{C}}$, using both the state $A(X)$ and the reduction polynomial $R(X)$. Let T be the indices of the tabs. Then our update-function can be written as

$$a_N := \sum_{i\in T} a_i r_i \mod D\,.$$

We replace the state polynomial $A(X)$ by $A'(X)$ which is defined as

$$A'(X) := \sum_{i=0}^{N-1} a_{i+1} X^i\,.$$

So for each step, we replace the old state $A(X)$ by a new state $A'(X)$. In addition, we extract the element a_0 as output of the LFSR to form the corresponding sequence.

Such sequences are particularily useful if they generate a long stream of elements without repetition. In particular, the sequence repeats iff the state polynomial $A(X)$ repeats after a certain number of steps. Note that LFSR over GF(2) and $\mathrm{d}\underset{\cdot}{\mathrm{l}}\underset{\cdot}{\mathrm{l}}\mathrm{F}\mathit{Int}\dot{s}$for $D = 2$ share a lot of similarities as in both cases, the state polynomial $A(X)$ toggles between two values, namely 0/1 over GF(2) and 1/3 over $\tilde{\mathbb{N}}^1$, respectively.

We start with a small example over $\mathrm{d}_{\mathrm{I}}\mathrm{F}_{\mathrm{I}}\dot{Int}s$ with $R(X) = X^2 + 3$ and D:

$$X + 1 \to 3X + 1 \to 3X + 3 \to X + 3 \rightsquigarrow (X + 1) \textit{ repetition}$$

While is gives us the full $2^2 = 4$ possible states, the pictures looks different for D=8, although we use the same reduction polynomial $R(X) = X^2 + 3$.

$$X + 1 \to 3X + 1 \to 3X + 3 \to X + 3 \rightsquigarrow (X + 1) \textit{ repetition}$$

We note that $3.3 \equiv 1 \mod D$, so with the given reduction polynomial, we cannot reach the other two values 5, 7. Starting with the all-7-state, we obtain:

$$7X + 7 \to 5X + 7 \to 5X + 5 \to 7X + 5 \rightsquigarrow (7X + 7) \textit{ repetition}$$

Similarly as before, we have $3.5 \equiv 7$ and $3.7 \equiv 5$ for $D = 8$, so the two cycles have no matching element with $D = 8$.

Finally, we look at the case $D = 4, N = 3$ and $R(X) = X^3 + 3$. Here we get

$$X^2 + X + 1 \to 3X^2 + X + 1 \to 3X^2 + 3X + 1 \to 3X^2 + 3X + 3 \to X^2 + 3X + 3 \to X^2 + X + 3$$

$$\rightsquigarrow (X^2 + X + 1) \textit{ repetition}$$

We note that this cycle includes all possible states $A(X)$ except $X^2 + 3X + 1, 3X^2 + X + 3$. Some more examples for specific values of D, N are given in app. C.3 and C.4, respectively.

After looking at some individual choices, we take more of a bird's eye view and look at aggregated data. In particular see App. C. The results are summarized in Tabs 10 and 11. As we can see there, the corresponding LFSR does not achieve full cycle length when compared to a binary LFRR. However, there are notably similarities but also differences for LFSR over $\mathrm{d}_{\mathrm{I}}\mathrm{F}_{\mathrm{I}}\dot{Int}s$ and $\mathrm{d}_{\mathrm{I}}\mathrm{F}_{\mathrm{I}}\breve{P}olys$, cf. tab 10 and 11.

Table 10: Summary of the results on $\mathrm{d}_{\mathrm{I}}\mathrm{F}_{\mathrm{I}}\dot{Int}s$ LFSR from App. C.1. Given are $D = 2^{d+1}$ for dimension d of the $\mathrm{d}_{\mathrm{I}}\mathrm{F}_{\mathrm{I}}\dot{Int}s$, the degree N of the polynomial and the longest cycle length for given D, N.

	$N=2$	$N=3$	$N=4$	$N=5$	$N=6$	$N=7$	$N=8$
$D=4$	4	6	14	30	62	126	254
$D=8$	4	8	14	30	62	126	254
$D=16$	8	16	28	60	124	252	

Table 11: Summary of the results on $\mathrm{d}_{\mathrm{I}}\mathrm{F}_{\mathrm{I}}\breve{P}olys$ LFSR from App. C.2. Given are $D = 2^{d+1}$ for dimension d of the $\mathrm{d}_{\mathrm{I}}\mathrm{F}_{\mathrm{I}}\breve{P}olys$, the degree N of the polynomial and the longest cycle length for given D, N.

	$N=2$	$N=3$	$N=4$	$N=5$	$N=6$	$N=7$	$N=8$
$D=4$	4	6	14	30	62	126	254
$D=8$	8	12	28	60	124	252	508
$D=16$	8	16	28	60	124	252	

We can draw some initial conclusions from the above tables. First, if we exclude $N = 2$, the cycle length for $D = 4$ always has the length $2^N - 2$, independent over which of the two $\text{d}\underset{\cdot}{\text{l}}\underset{\cdot}{\text{l}}\text{Ⅎ}\curvearrowleft Fields$ the corresponding LFSR is defined. For $D = 8$, the picture is different—the LFSR over $\text{d}\underset{\cdot}{\text{l}}\underset{\cdot}{\text{l}}\text{Ⅎ}\dot{Int}s$ follows the pattern of $D = 4$, while the LFSR over $\text{d}\underset{\cdot}{\text{l}}\underset{\cdot}{\text{l}}\text{Ⅎ}\breve{P}olys$ exactly follows the numbers for $D = 16$. For $D = 16$, the numbers are the same again for both $\text{d}\underset{\cdot}{\text{l}}\underset{\cdot}{\text{l}}\text{Ⅎ}\curvearrowleft Fields$. In addition, still, for $D = 16$ while excluding $N = 2, 3$, we can give a closed formula for the length of the largest cycle, namely $2^{N+1} - 4$. Alas, as we see in the next part, this simple law does not carry forwarsd if we increase the number of values N over $\text{d}\underset{\cdot}{\text{l}}\underset{\cdot}{\text{l}}\text{Ⅎ}\curvearrowleft Fields$.

Maximal cycle length for $D = 16$ and selected reduction polynomials $R(X)$. Based on the above observations, we plotted cycle lengths for concrete LFSR and fixed $D = 16$. In all cases, we use the all-1-polynomial $\sum_{i=0}^{N-1} X^i$ as initial state. The results can be found in Tab. 12. In particular the four reduction polynomials used for these experiments are also shown there. These four polynomials gave a maximal cycle and also had minimal weight according to the results from App. C. We see that these four polynomials have a lot of similarities, in particular we have the form $X^N + X^{N-2} + X + c$ for $c = 01/03$ for odd degrees N while the corresponding polynomials for even degrees N are $X^N + X^2 + X + c$. On the other hand, $c = 1$ holds for the reduction polynomial $R(X)$ over $\text{d}\underset{\cdot}{\text{l}}\underset{\cdot}{\text{l}}\text{Ⅎ}\dot{Int}s$, but $c = 03$ over $\text{d}\underset{\cdot}{\text{l}}\underset{\cdot}{\text{l}}\text{Ⅎ}\breve{P}olys$. Investigating the precise cause for this regularity is out of the scope of this paper.

Table 12: Cycle length for an LFSR for $\text{d}\underset{\cdot}{\text{l}}\underset{\cdot}{\text{l}}\text{Ⅎ}\curvearrowleft Fields$ for $D = 16$ and various N.

N	4	6	8	10	12	14
Cycle $\text{d}\underset{\cdot}{\text{l}}\underset{\cdot}{\text{l}}\text{Ⅎ}\dot{Int}s$	28	124	508	1,860	2,380	32,764
Cycle $\text{d}\underset{\cdot}{\text{l}}\underset{\cdot}{\text{l}}\text{Ⅎ}\breve{P}olys$	28	124	508	1,860	2,380	32,764
N	16	18	20	22	24	26
Cycle $\text{d}\underset{\cdot}{\text{l}}\underset{\cdot}{\text{l}}\text{Ⅎ}\dot{Int}s$	13,1068	152,908	2,064,132	507,780	33,554,428	2,336,292
Cycle $\text{d}\underset{\cdot}{\text{l}}\underset{\cdot}{\text{l}}\text{Ⅎ}\breve{P}olys$	131,068	152,908	2,064,132	507,780	33,554,428	2,336,292
N	28	30	32			
Cycle $\text{d}\underset{\cdot}{\text{l}}\underset{\cdot}{\text{l}}\text{Ⅎ}\dot{Int}s$	349,524	20,053,404	8,388,604			
Cycle $\text{d}\underset{\cdot}{\text{l}}\underset{\cdot}{\text{l}}\text{Ⅎ}\breve{P}olys$	349,524	20,053,404	8,388,604			
N	5	7	9	11	13	15
Cycle $\text{d}\underset{\cdot}{\text{l}}\underset{\cdot}{\text{l}}\text{Ⅎ}\dot{Int}s$	60	252	252	3,556	13,020	47,244
Cycle $\text{d}\underset{\cdot}{\text{l}}\underset{\cdot}{\text{l}}\text{Ⅎ}\breve{P}olys$	60	252	252	3,556	13,020	47,244
N	17	19	21	23	25	27
Cycle $\text{d}\underset{\cdot}{\text{l}}\underset{\cdot}{\text{l}}\text{Ⅎ}\dot{Int}s$	1,020	1,015,684	3,047,052	16,777,212	8,388,604	1,195,740
Cycle $\text{d}\underset{\cdot}{\text{l}}\underset{\cdot}{\text{l}}\text{Ⅎ}\breve{P}olys$	1,020	1,015,684	3,047,052	16,777,212	8,388,604	1,195,740

Reduction Polynomial $R(X)$ for the above .

$R(X)$	$\text{d}\underset{\cdot}{\text{l}}\underset{\cdot}{\text{l}}\text{Ⅎ}\dot{Int}s$	$\text{d}\underset{\cdot}{\text{l}}\underset{\cdot}{\text{l}}\text{Ⅎ}\breve{P}olys$
N even	$X^N + X^2 + X + 1$	$X^N + X^2 + X + 03$
N odd	$X^N + X^{N-2} + X + 1$	$X^N + X^{N-2} + X + 03$

Maximal cycle length for $N = 11, 12$ and variable dimension d. Using the same reduction polynomials as above, we now increase the parameter d. The results are given in tables 13 and 14.

In particular, we use the same polynomials as in Tab. 12, but this times for $N = 11, 12$, respectively. The results are interesting as we can see differences between ɖiɭɟ$\dot{Int}s$ and ɖiɭɟ$\breve{P}olys$. While ɖiɭɟ$\dot{Int}s$ double the length of the cycle for every increase of dimension d, ɖiɭɟ$\breve{P}olys$ only do so for every power of 2. Apart from that, the cycle length is exactly the same. Similarly, the cycle length for $N = 11$ are exactly the same. Interestingly, they are sometimes *higher* than for $N = 12$. Unfortunately, there is not such a clear law for the boundaries of cycle length for ɖiɭɟ$\breve{P}olys$ as for $N = 12$. It is out of the scope of this paper to dig deeper into these questions.

Table 13: Empirical results on the cycle length for $N = 11$ and variable d.

d	log CycLen	CycLen
3	11.762	3,472
4	12.762	6,944
5	13.762	13,888
6	14.762	27,776
7	15.762	55,552
8	16.762	111,104
9	17.762	222,208
10	18.762	444,416
11	19.762	888,832
12	20.762	1,777,664
13	21.762	3,555,328
14	22.762	7,110,656
15	23.762	1,4221,312
16	24.762	2,8442,624

$R(X) = X^N + X^{N-2} + X + 1$
ɖiɭɟ$\dot{Int}s$

d	log CycLen	CycLen
3..5	11.762	3,472
6..10	12.762	6,944
11..21	13.762	13,888
22..42	14.762	27,776
43..69	15.762	55,552

$R(X) = X^N + X^{N-2} + X + 03$
ɖiɭɟ$\breve{P}olys$

Summary. Obviously, these observations only scratch the surface of a potential theory of LFSR over ɖiɭɟ↶$Fields$. Assuming we can characterize the elements of the maximal cycle for a given LFSR, we could embed a message or key into it and therefore avoid smaller cycles for a given LFSR—similarily we avoid the all-0-state for binary LFSR. Among others, this contains the following **uədO P̆roblem 11**

i. How can the difference in cycle length between ɖiɭɟ$\dot{Int}s$ and ɖiɭɟ$\breve{P}olys$ be explained?

ii. Do ɖiɭɟ$\breve{P}olys$ always have similar performance or outperform ɖiɭɟ$\dot{Int}s$ as Tables 10 and 11 suggest?

iii. Is there a general formula or algorithm, that can predict maximal cycle length and the corresponding reduction polynomial for any choice of parameters D, N? If so—how does it look like?

iv. Are there cryptographic primities that could benefit from the use of LFSR over ɖiɭɟ↶$Fields$?

4.7 Non-Linear Feedback Shift Registers (NLFSR)

To increase the cycle length for a given feedback polynomial and a given state size, we now consider non-linear feedback shift registers.

Table 14: Empirical results on the cycle length for $N = 12$ and variable d.

d	log_2(CycLen)	CycLen
3	11.217	2,380
4	12.217	4,760
5	13.217	9,520
6	14.217	19,040
7	15.217	38,080
8	16.217	76,160
9	17.217	152,320
10	18.217	304,640
11	19.217	609,280
12	20.217	1,218,560
13	21.217	2,437,120
14	22.217	48,74,240
15	23.217	9,748,480
16	24.217	19,496,960

$R(X) = X^N + X^2 + X + 1$
$\mathrm{Fli}\mathrm{p}\mathrm{F}\dot{Int}s$

d	log_2(CycLen)	CycLen
3	11.217	2,380
4..7	12.217	4,760
8..15	13.217	9,520
16..31	14.217	19,040
32..63	15.217	38,080
64..65	16.217	76,160

$R(X) = X^N + X^2 + X + 03$
$\mathrm{Fli}\mathrm{p}\mathrm{F}\check{P}olys$

Affine case. We start by comparing the univariate and the multivariate representation. Both are actually quite similar and even equivalent under most circumstances. Consider

$$R(X) := X^N + \sum_{i=0}^{N-1} r_i X^i \text{ and } p(x) := \sum_{i=0}^{N-1} b_i x_i + c$$

for $b_i, c, r_i \in \tilde{\mathfrak{C}}$ and some $N \in \mathbb{N}$ and the vector $x := (x_0, \ldots, x_{N-1}$.

By construction, if we have $r_i = b_i$ for $0 \leq i < N$ and $c = 0$, the univariate and the multivariate representation yield the same output for the same initial sequence. Consider the state

$$A(X) := \sum_{i=0}^{N-1} a_i X^i \text{ and } S := [s_0, \ldots, s_{N-1}]$$

for $a_i, s_i \in \tilde{\mathfrak{C}}$. We call $A(X)$ the univariate and S the multivariate state, respectively. The two states are identical if $a_i = s_i \ \forall 0 \leq i < N$.

Corollary 3. *When both $R(X), p(x)$ and $A(X), S$ are equivalent, their output sequence is the same.*

There is one subtlety though: Note the constant c in the multivariate polynomial $p(x)$. This has no equivalent in the univariate representation. However, technically we don't need it as the coefficients a_0, r_0 correlate with the very tab in $R(X)$—which again correlates with b_0, x_0. We can see the difference in app. C and D.

Higher degree case. Dwelling on the results from the previous section, an obvious question is: Can we do better, maybe with non-linear feedback shift registers (NLFSR)? And if so—how do we construct such NLFSR with maximal cycle length? All in all, this endeavour is way more

Table 15: Summary of the results on dilFInts NLFSR from App. D. Given are the degree N of linear polynomials and the longest cycle length for given number of variables N via a genetic algorithm.

Empirical findings for affine multivariate polynomials, cf. App. D.2.

	$N=4$	N=5	$N=6$	$N=7$	$N=8$	$N=9$	$N=10$	$N=11$
$D=4$	15	31	63	127	255	511	1,023	2,047
$D=8$	30	62	126	254	510	1,022	2,046	4,094
$D=16$	60	124	252	508	1,020	2,044	4,092	8,188

	$N=12$	$N=13$	$N=14$	$N=15$	$N=16$	$N=17$
$D=4$	4,095	8,191	16,383	32,767	65,535	131,071
$D=8$	8,190	16,382	32,766	65,534	131,070	262,142
$D=16$	16,380	32,764	65,532	131,068	262,140	524,284

Empirical findings for affine multivariate polynomials, cf. App. D.3.

	$N=4$	N=5	$N=6$	$N=7$	$N=8$	$N=9$	$N=10$	$N=11$
$D=4$	15	31	63	127	255	511	1,023	2,047
$D=8$	30	62	126	254	510	1,022	2,046	4,094
$D=16$	60	124	252	508	1,020	2,044	4,092	8,188

	$N=12$	$N=13$	$N=14$	$N=15$	$N=16$	$N=17$
$D=4$	4,095	8,191	16,383	32,767	65,535	131,071
$D=8$	8,190	16,382	32,766	65,534	131,070	262,142
$D=16$	16,380	32,764	65,532	131,068	262,140	524,284

difficult as the number of possible choices becomes far larger as we can now use any multivariate monomial in the state coefficients $a_i \in \tilde{\mathfrak{C}}, 0 \leq i < N$. So to tackle this problem, we are using a genetic algorithm to search for NLFSR with a long period, cf. [Wi24] and also App. D.1 for the concrete idea used. The corresponding implementation is given in the Python method `gaRun`. A summary of the results of these experiments are given in Tab. 15.

There are several things to notice. First, the results of Tab. 15 exactly mirror the results of Tab. 10—with a slight twist, namely the values for N in Tab. 15 are the same values as for $N+1$ in Tab. 10. This is very much in line with the discussion about the extra constant term c from above. In addition, it gives us some confidence that the genetic algorithm actually did not only find a good solution but actually the optimal solution—after all, the results in Sect. 4.6 were achieved by considering *all* possible solutions, *i.e.* we know they are really optimal. Interestingly, the genetic algorithm usually outputs only affine polynomials and no linear polynomials. Based on the values in Tab. 15 we can infer the following.

Conjecture 1. *We conjecture that the maximal cycle length is given by*

$$\begin{cases} 2^{N+d-1} - d & \text{for } d = 1, 2 \\ 2^{N+d-1} - (d+1) & \text{for } d = 3 \end{cases}$$

for both affine and higher degree multivariate polynomials $p(x)$ in N variables for $N \geq 4$ when the coefficients are choosen over $\mathrm{d}\underset{\cdot}{\text{i}}\underset{\cdot}{\text{l}}\mathrm{F}\mathrm{Int}\dot{\mathrm{s}}$.

All in all, this results is surprising: The cycle length is way smaller than the maximal possible length, which would be 2^{Nd}. In particular it means that we only gain a factor of 2 when adding one more element for the corresponding $\mathrm{d}\underset{\cdot}{\text{i}}\underset{\cdot}{\text{l}}\mathrm{F}\mathrm{Int}\dot{\mathrm{s}}$ with $D = 4, 8, 16$. Hence, the maximal cycle of a NLFSR and a LFSR over GF(2) drift further and further away from each other for a larger number of variables N. Secondly, using higher degree polynomials rather than affine polynomials does not improve the cycle length. Maybe, a cycle length higher than given by the above conjecture are so rare that they are not spotted via the genetic algorithm.

However, in an additional experiment, we did not only vary the polynomial but also the initial state. The results are absolutely identical, which supports the above conjecture. However, as they are identical, the corresponding polynomials and states were not included into the appendix. Alas—this means that there might be a theoretical reason why $\mathrm{d}\underset{\cdot}{\text{i}}\underset{\cdot}{\text{l}}\mathrm{F}\mathrm{Int}\dot{\mathrm{s}}$ cannot go full cycle length 2^{Nd}. It is out of the scope of this paper to address this question further.

On the other hand, LFSR over $\mathrm{d}\underset{\cdot}{\text{i}}\underset{\cdot}{\text{l}}\mathrm{F}\mathrm{Int}\dot{\mathrm{s}}$ have a different structure than LFSR over GF(2). Hence, mixing them into some cryptographic primitive might force the cryptanalyst to spend overproportional effort to break this primitive.

uədO P̆roblem 12

i. Why do affine polynomials do as good as polynomials of higher degree in terms of cycle length?

ii. Can we construct a polynomial that actually improves over the bound of the above conjecture?

iii. More general: Is there a way to proof or disproof for the above conjecture?

iv. Is there a general formula or algorithm, that can predict maximal cycle length and the corresponding reduction polynomial for any choice of parameters D, N? If so—how does it look like?

v. What is the cycle structure for non-linear feedback shift registers?

vi. Are there cryptographic primitives that could benefit from the use of NLFSR over $\mathrm{d}\underset{\cdot}{\text{i}}\underset{\cdot}{\text{l}}\mathrm{F}\curvearrowleft\mathit{Fields}$?

4.8 Ring mod $X^N + 1$

In some cryptographic applications such as Saber or ML-KEM, we use operations in the polynomial ring mod $X^N + 1$ for some $N \in \mathbb{N}, N > 0$, in particular to keep the internal state small. We see here why we cannot use this in the case of dịlƎ*Inṫs*. The same is also true for dịlƎ*P̆olys*.

First, the polynomial X^N+1 cannot be expressed over an dịlƎ*Inṫs*as the tally operator needs an odd number of operands. To keep things simple, we propose a concurrent reduction by X^N+1 and $X^{N+1} + 3$. We see why this solves our problem by considering the polynomial $P(X) := a_N X^N + a_{N-1} + \ldots + a_1 X + a_0$. Using a reduction by $X^N + 1$, we obtain $a'_0 := [a_0, a_N, 3a_{N-1}]$ — which is an element of the corresponding dịlƎ*Inṫs*, assuming $a_0, a_N, 3a_{N-1} \in \tilde{\mathbb{N}}$, due to the virtute of the definition of the tally-operator.

Definition 4. *For some $K \in \mathbb{N}$ we have*

$$P(X) := \sum_{i=0}^{K} a_i X^i$$

We call this polynomial $P(X)$ ***full*** *as all of its coefficients a_i are drawn from the corresponding dịlƎ*Inṫs*, i.e. $a_i \in \tilde{\mathbb{N}}$ for $0 \leq i \leq K$.*

Corollary 4. *All full polynomials $P(X) := \sum_{i=0}^{K} a_i X^i$ with $K = 2\tau + N - 1$ for some $\tau \in \mathbb{N}$ can be fully reduced via the two polynomials $X^N + 1$ and $X^{N+1} + 3$ to $P'(X) := \sum_{i=0}^{N-1} a'_i X^i$.*

Obviously, there are more elaborate structures possible and this approach can be trailered to the needs of specific cryptographic systems, but they are out of the scope of this paper, so we state them as **uədO P̆roblem 13** Which other cryptographically relevant rings are there to allow an efficient or cryptographically relevant reduction of polynomials $P(X)$ over dịlƎ*Inṫs* or dịlƎ*P̆olys*, respectively.

4.9 Polynomial Multiplication

While multiplication is a very basic operation, it is a bit more complex over dịlƎ↶*Field*s such as dịlƎ*Inṫs*. Let $\tilde{\mathbb{N}}$ be some dịlƎ*Inṫs*. We start with a toy example. Let $A(X), B(X)$ be two polynomials of degree 2 and 3, respectively. We have coefficients $a_i, b_j \in \tilde{\mathbb{N}}$ for $i, j \in \mathbb{N} : 0 \leq i \leq 2, 0 \leq j \leq 2$. Hence we have the two polynomials

$$A(X) := a_0 + a_1 X + a_2 X^2 \text{ and } B(X) := b_0 + b_1 X + b_2 X^2 + b_3 X^3 .$$

Adding them is simple as we have $R(X) := (a_0 + b_0) + (a_1 + b_1)X + (a_2 + b_2)X^2 + b_3 X^3$. For the multiplication $R(X) := A(X) \cdot B(X)$, see Table 16 for intermediate steps.

Table 16: Multiplication of $A(X)$ and $B(X)$ showing (a) the resulting degrees and (b) the product of the corresponding coefficients.

(a)	B(X)			
	0	1	2	3
A(X)	1	2	3	4
	2	3	4	5

(b)	B(X)			
	a_0b_0	a_0b_1	a_0b_2	a_0b_3
A(X)	a_1b_0	a_1b_1	a_1b_2	a_1b_3
	a_2b_0	a_2b_1	a_2b_2	a_2b_3

On this basis, we can tabulate the coefficients of the resulting polynomial $R(X)$, see Table 17. We see that for r_1, r_4 we cannot form a tally as the number of terms is even, not odd. We call

Table 17: Multiplication of $R(X) := A(X) \cdot B(X)$, resulting coefficients in $R(X)$

Closed	r_i	X	sum/tally
	r_0	1	$[a_0b_0]_1$
$\rightarrow$	r_1	X	$a_1b_0 + a_0b_1$
	r_2	X^2	$[a_2b_0, a_1b_1, a_0b_2]_3$
	r_3	X^3	$[a_2b1, a_1b_2, a_0b_3]_3$
$\rightarrow$	r_4	X^4	$a_2b_2 + a_1b_3$
	r_5	X^5	$[a_2b_3]_1$

them **closed equations**, for r_0, r_2, r_3, r_5, we call those **active equations**. To remedy this, we require $r_1 = r_4 = 0$ over the integers. From the point of view of the corresponding dịlƎ*Inṫs*, this means that these coefficients vanish. Hence, we require

$$\begin{cases} a_1b_0 = -a_0b_1 \text{ from } r_1 \\ a_2b_2 = -a_1b_3 \text{ from } r_4 \end{cases}$$

While this looks easy at first glance, consider that this is a quadratic equation, so for larger degrees of the two polynomials $A(X)$ and $B(X)$ this becomes more and more difficult to solve. This is very different from the situation over finite fields or rings where we can multiply any two polynomials and expect the result to lie in the corresponding polynomial ring. On the other hand, this poses problems for cryptanalytic algorithms and hence gives hope that parameters for cryptographic problems over dịlƎ↶*Field*s can be lowered without compromising security. All in all, we started with 7 coefficients a_i, b_j in the above toy example but are now left with only 3 coefficients we can choose freely, *e.g.* to store key material for a key exchange, compute for a particular message for a signing scheme or choose at random for blinding purposes.

Still, we have the following **uədO P̆roblem 14** over dịlƎ↶*Field*s: Given a polynomial $A(X) \in \widetilde{C}$, can we efficiently compute a (preferably random) polynomial $B(X) \in \widetilde{C}$ for given $A(X)$ s/t $R(X) := A(X) \cdot B(X)$ also is defined over the same dịlƎ↶*Field*, *i.e.* we have $R(X) \in \widetilde{C}[X]$. Alternatively, construct a group that is closed under multiplication and addition in the polynomial ring $\widetilde{C}(X)$. If neither is possible, show that this problem is difficult, *e.g.* $\mathcal{NP}$-complete.

For special cases, this problem is easy, such as our example in Tab. 17. However, for other cases such as degree $A(X) = 3$ and degree $B(X) = 5$, the problem becomes more convoluted. In particular, we have the following results for Tab. 18.

Table 18: Multiplication of $R(X) := A(X) \cdot B(X)$ with degree 7, resulting problematic coefficients for the polynomial $R(X)$

Closed	r_i	X	sum/tally
$\rightarrow$	r_1	X	$a_0b_1 + a_1b_0$
$\rightarrow$	r_3	X^3	$a_0b_3 + a_1b_2 + a_2b_1 + a_3b_0$
$\rightarrow$	r_4	X^4	$a_0b_4 + a_1b_3 + a_2b_2 + a_3b_1$
$\rightarrow$	r_5	X^5	$a_0b_5 + a_1b_4 + a_2b_3 + a_3b_2$
$\rightarrow$	r_7	X^7	$a_2b_5 + a_3b_4$

As we see, several coefficients such as a_0 or b_3 appear several times on the rhs. This hinders efficient optimization. To obtain a system of equations with maximal freedom, we set all of the

above equations to zero, *i.e.*

$$\begin{cases} a_0b_1 + a_1b_0 = 0 \\ a_0b_3 + a_1b_2 + a_2b_1 + a_3b_0 = 0 \\ a_0b_4 + a_1b_3 + a_2b_2 + a_3b_1 = 0 \\ a_0b_5 + a_1b_4 + a_2b_3 + a_3b_2 = 0 \\ a_2b_5 + a_3b_4 = 0 \end{cases}$$

Solving such a system for example with Gröbner bases is not possible as they lead to many parasiteric solutions. The problem is very similar to the one discussed in Sect. 4.2 regarding Gaussian elimination. Hence, we concentrate on a different special case now.

Assume the polynomial $A(X)$ is fixed and we have some freedom to choose the polynomial $B(X)$, *e.g.* it is a randomly choosen element. We are now looking for a version of the above system of equations that relies on a minimal number of variables on the rhs, while yielding elements of $\mathfrak{C}$ for all variables on the lhs.

Lemma 9. *Let $M, N \in \mathbb{N}$ with $M \leq N$. We have the two polynomials $A(X) := \sum_{i=0}^{M-1} a_i X^i$ and $B(X) := \sum_{j=0}^{N-1} b_j X^j$ with $A(X)$ being completely determined. We have*

$$R(X) = \sum_{k=0}^{M+N-2} r_k X^k := \left(\sum_{i=0}^{M-1} a_i X^i \right) \cdot \left(\sum_{j=0}^{N-1} b_j X^j \right) = A(X) \cdot B(X) \,.$$

Then the number of closed equations given by this is

$$\begin{cases} M-1 & \text{for } M \text{ odd} \\ N-M+1 & \text{for } M \text{ even} \end{cases}$$

On average, we expect the following number of free variables b_j in the polynomial $B(X)$:

$$\begin{cases} N-M+1 & \text{for } M \text{ odd} \\ 1 & \text{for } M \text{ even} \end{cases}$$

Proof. **Case $M = N$:** We start with the two polynomials $A(X), B(X)$ having the same degree. We observe that r_0 is an active equation while r_1 is a closed equation. This pattern repeats alternating for r_k, *i.e.* for k even, it is an active equation while for k odd, it is a cloded equation. As $2N-2$ is an even number, r_{2N-2} is an active equation, too. A simple count yields $M-1$ closed equations. On the other hand, we have a total of M coefficients in $B(X)$. We hence except 1 free variable in $B(X)$ on average.
Case $M \leq N$: We now generalize the result to establish the claim of this lemma. In particular, we observe that for all new equations for $N > M$, we simply repeat the number of terms for $k = M$. Hence, if M is even, these are all closed equations. In total, there are $(N-M)$ of these new equations. We hence need to add them to the number of closed equations and obtain $1+N-M = N-M+1$ for M even. Alternatively, if M is odd, we only obtain active equations and hence do not need to increase the number of closed equations. The corresponding number will therefore still be $M-1$. Subtracting this from the number of variables in $B(X)$ completes the proof. □

From a practical point of view, we need to choose N larger than M and also M odd. In this case, an average of $M-1$ coefficients of $B(X)$ will be determined while $N-M+1$ coefficients

can be choosen freely. Most likely, trade-offs between choosing parts of $A(X)$ and $B(X)$ will be possible. However, this is out of the scope of this paper. There are some intermediate results though. A first thought might be to rearrange this system of equations using Gröbner bases. In particular, it seems to make sense to choose an ordering s/t the most frequent variables stay on the rhs—although this is a heuristic. Alas —as confirmed via a quick test on Sage— we run in a similar problem as for Gaussian Elimination (cf Sect. 4.2: The corresponding algorithm is written over the corresponding integer ring 2^N or the finite field $\mathrm{GF}(2^N)$ and the results are not useful over a $\mathrm{d}\dot{\mathrm{I}}\mathrm{l}\mathrm{F}\curvearrowleft Field$. While it might be possible to transfer Buchberger's algorithm or F_5 to $\mathrm{d}\mathrm{I}\mathrm{l}\mathrm{F}\curvearrowleft Field$s, this is out of the scope of this paper. Most likely, there will be problems similar to the ones outlined in App. A.2, *i.e.* decomposition when implementing such an algorithm. Hence, a deeper investigation is needed at this point.

However, there are two more similar alternatives. In particular, ElimLin [CS+12, QW15] has a very easy structure and might hence be tweaked for our purpose. Therefore, we guess one variable a_i/b_j and use potential linear equations for further simplifying the system of equations.

As the overall problem is similar to the $\mathcal{NP}$-hard problem *maximum coverage*, we may use a version of the corresponding Greedy algorithm to solve it and adapt it accordingly.

i. First, we choose the variable that is contained in the highest number of quadratic terms, denote it with x_i

ii. Then, we choose a random value from $\mathfrak{C}$ and assign it to x_i

iii. We repreat this process until all quadratic terms have vanished

iv. Now we solve the remainind system of linear equations using the algorithm from App. A.2 If no such solution exists, reset all variables and go back to Step 1

Table 19: Running `symRewriteEqsLin` for different sets of parameters.

$\lvert a_i\rvert$	$\lvert s_j\rvert$	|output|
31	31	31
31	32	32
32	31	32
32	32	32

$\lvert a_i\rvert$	$\lvert s_j\rvert$	|output|
32	33	32
32	64	32
33	32	32
33	33	33

$\lvert a_i\rvert$	$\lvert s_j\rvert$	|output|
33	34	34
33	64	64
34	33	34
34	34	34

$\lvert a_i\rvert$	$\lvert s_j\rvert$	|output|
34	16	16
34	15	34
35	34	34
35	35	35

$\lvert a_i\rvert$	$\lvert s_j\rvert$	|output|
128	128	128
128	129	128
129	128	128
129	129	129

To tackle the problem in a very specific setting, we use a combination of the two approaches, see function `symRewriteEqsLin`. We assume two polynomials $a(X) := \sum_{i=0}^{M-1} a_i X^i$ and $s(X) := \sum_{j=0}^{N-1} s_j X^j$ with coefficients $a_i, s_j \in \tilde{\mathbb{N}}$ with some parameters $d \in \mathbb{N}$ and $D := 2^{d+1}$. They are combined to generate an output polynomial $r(X) := a(X) \cdot s(X)$. Note that `symRewriteEqsLin` is a symbolic algorithm. In a first step, we multiply out $a(X)$ and $s(X)$ to obtain $r(X)$ in its symbolic form. Second, we assign random values to the coefficients s_j. This way, we obtain a linear system of equations. Finally, we identify all closed equations, solve them for one variable and substitute into all other equations. The process stops when there are no more closed

equations left. Note that we had to modify the concept of "closed equation" slightly to make this approach work. In particular, we do not count the number of *terms* anymore, but the *sum* of all coefficients. Let $c_0a_0 + \ldots + c_ka_k$ be such a polynomial. We call this polynomial **closed** iff $\sum_{i=0}^{k} c_0 \mod D$ is an even number and active otherwise. This is necessary as the substitution operator is not completely closed in the $\text{dịlƎ}\mathit{Int}\dot{s}$ and produces some even coefficients. This is captured in the above definition.

Running some small-scale experiments with $D = 512$ and setting s_j to values from $\{1, 3, 5\}$ (uniform distribution), we see that `symRewriteEqsLin` behaves nicely. Each experiment was run 10 times, all results for each set of parameters were the same. We note that the number of (active) output equations is either $|a_i|$ or $|s_i|$. It is not clear at present which of them will dominate in which case. Note that this does not violate Lem. 9 as we assume both sets of variable to be free. In our case, only the variables a_i are free while s_j has been assigned to specific values. However, from a cryptological point of view, the situation is satisfactory as we can always pick pairs $(|a_i|, |s_i|)$ to achieve the number of output equations we want. So, we should have flexibility in the parameter $|s_j|$ when constructing a real-world system. This leads to the following **uədO Ŏ̆roblem 15**.

i. Can we predict output for given a,s?

ii. Can we do better? Can we replace `symRewriteEqsLin` with a more sophisticated, non-heuristic algorithm?

4.10 Miscellaneous

Lemma 10. *Let $d \in \mathbb{N}, d \geq 2$ be the dimension of a* $\text{dịlƎ}\mathrm{Int}\dot{s}$. *For some element $c \in \tilde{\mathbb{N}}$ consider a polynomial equation $P(x) = c \bmod 2^{d+1}$ that has root x. Denote $q := 2^d$,* i.e. *$D = 2q$. In addition, we consider the set of all exponents of $P(x)$ with V and the set of all odd exponents with O. If $|O|$ is even, this equation must also have the root $x + 2^d = [x, 2^{d-1} - 1, 2^{d-1} + 1]$,* i.e.

$$P(x) = c, |O| \text{ even } \Rightarrow P(x + q) = c \text{ or over } \text{dịlƎ}\mathrm{Int}\dot{s}\ P([x, 2^{d-1} - 1, 2^{d-1} + 1]) = c\,.$$

Proof. For each non-constant term we have

$$(x + q)^n = \sum_{k=0}^{n} \binom{n}{k} x^{n-k} q^k \equiv x^n + nqx^{n-1} \bmod D$$

as all terms containing $2q = D$ and hence q^2 vanish. So the only non-zero terms in the above equivalence come from the values $k = 0, 1$. For even exponents n, we can even tighten this result as even n time one factor of q yields the product $2q = D$ and is hence zero (mod D).

$$(x + q)^n \equiv \begin{cases} x^n & n \text{ even} \\ x^n + nqx^{n-1} & n \text{ odd} \end{cases} \pmod{D}\,.$$

Now let us see if there are other solutions, too. For some polynomial P and some $x \in \tilde{\mathbb{N}}$ consider $P(x + q)$. Subtracting the two polynomials yields:

$$P(x + q) - P(x) \equiv q \sum_{k \in V} \alpha_i k x^{k-1} \pmod{D}$$

with $\alpha_i \in \tilde{\mathbb{N}}$ the coefficients of P. As we have $q = 2^d$, the above expression will vanish modulo D iff $\sum_{k \in V} \alpha_i k \equiv 0 \pmod 2$. As all α_i and x are odd, this can only happen if $\sum_{k \in V} k$ is even. However, this can only happen if $\sum_{k \in O} k \equiv 0 \pmod 2$ or $|O|$ is even which yields the result. □

Definition 5. *For some $a \in \tilde{\mathbb{N}}$, we call $a+q$ its* ***conjugate*** *and denote this via $\overline{x} = x + q$.*

This definition is motivated by Lem. 10. Note that we have $x = \overline{\overline{x}} \pmod{2q}$ with $D = 2q$.

Corollary 5. *Consider $a, b \in \tilde{\mathbb{N}}$ and their conjugates $\overline{a} = a + 2^d$ and $\overline{b} = b + 2^d$. We have*

$$\overline{a}\overline{b} \equiv ab \pmod{2^{d+1}}.$$

Proof. Consider

$$\overline{a}\overline{b} = (a + 2^d)(b + 2^d) = ab + 2^d(a + b) + 2^{2d}.$$

Since a and b are odd, $(a + b)$ is even. Hence there exists $k \in \mathbb{N}, k \geq 1$ such that $a + b = 2k$, and therefore

$$2^d(a + b) = 2^{d+1}k \equiv 0 \pmod{2^{d+1}}.$$

In addition, since $d \geq 1$, we have

$$2^{2d} = 2^{d+1}2^{d-1} \equiv 0 \pmod{2^{d+1}}.$$

Consequently,

$$\overline{a}\overline{b} \equiv ab \pmod{2^{d+1}}.$$

□

Lemma 11. *Let $d \in \mathbb{N}, d \geq 1$ be the dimension of a dyadic Int, so $D = 2^{d+1}$. Let $P(x) := []_{v \in V} \alpha_k x^k$ be a polynomial with exponents $V \subset \mathbb{N}$ and coefficients $\alpha_k \in \tilde{\mathbb{N}}$ for $k \in V$. Let $O := \{k \in V, 2 \nmid k\}$ be the odd exponents of the polynomial $P(x)$. Then the following equivalence holds:*

$$P(x) = P(-x) \ (mod\ D) \iff \sum_{k \in O} \alpha_k x^k \equiv 0 \ (mod\ 2^d)$$

.

Proof. Denote with $E := V \backslash O$ the even exponents. Now we can write

$$P(-x) = []_{k \in V} \alpha_k (x)^k = []_{k \in E} \alpha_k (-x)^k - []_{k \in O} \alpha_k (x)^k$$

Over the integers, subtracting this from $P(x)$ yields

$$P(x) - P(-x) = 2 \sum_{k \in O} \alpha_k x^k$$

as we have $(-1)^k = 1$ if k is even and hence even terms cancel out in the above subtraction. Using the equivalence

$$P(x) - P(-x) = 2 \sum_{k \in O} \alpha_k x^k \equiv 0 \bmod 2^{d+1} \iff \sum_{k \in O} \alpha_k x^k \equiv 0 \bmod 2^d$$

completes the proof. □

Corollary 6. *So aimed with these two lemmata, we can see why $a^2 - 1 = 0$ over a dyadic Int has the four solutions*

$$\{-1, +1, 2^d - 1, 2^d + 1\} = \{-1, +1, q - 1, q + 1\}.$$

Proof. Obviously, $a = 1$ is a solution. Using Lem. 11 and substituting we obtain

$$1^2 - 1 = 0$$

and hence, P(-1) is a solution as well. Employing Lem. 10, we can also include the two conjugates $q - 1$ and $q + 1$ to our set of solutions. □

We can tighten this by verifying that they are actually the only solutions:

Lemma 12. *For some $d\underset{\cdot}{\mathrm{Fl}}\mathrm{Int}\dot{\mathrm{s}}$ with $d \geq 2, D = 2^{d+1}, q = 2^d$ consider $a^2 = 1$. This equation has exactly the solutions $\{-1, 1, q-1, q+1\}$.*

Proof. Rewrite $a^2 - 1 = (a-1)(a+1)$ yields the solutions $\{-1, 1\}$ both over $\mathbb{Z}$ and $d\underset{\cdot}{\mathrm{Fl}}\mathrm{Int}\dot{\mathrm{s}}$. Using Lem. 10 yields the set $\{-1, 1, q-1, q+1\}$ as $(q-1), (q+1)$ are conjugates of -1, 1, respectively.

We now show that these are the only possible solutions for $d \geq 2$. Since $a\tilde{\mathbb{N}}$ is odd, both $(a-1)$ and $(a+1)$ are even. In addition we have

$$(a+1) - (a-1) = 2\,.$$

Therefore, they cannot be both divisible by 4. In fact, exactly one is divisible by 4.
Case $4 \mid (a-1)$**.** As we have $(a+1) = (a-1) + 2$ we can rewrite this as

$$a + 1 \equiv 2 \pmod 4\,.$$

Using two integers $u, v \in \mathbb{Z}$ with v odd this becomes

$$a - 1 = 4u,\ a + 1 = 2v$$

From above we have

$$2^{d+1} | (a-1)(a+1) \Longleftrightarrow 2^{d+1} | 8uv\,.$$

Since v is odd and we have $d \geq 2$, we factor out 8 and obtain

$$2^{d-2} | u \text{ equivalently } 2^d | (a-1)\,.$$

As we work modulo 2^{d+1}, this yields the following solutions:

$$a = 1,\ a = q + 1$$

Case $4 \mid (a+1)$**.** The proof is symmetrical to the case above and yields $a = -1, a = q - 1$. Hence, the only solutions for $a^2 = 1$ and $d \geq 2$ are -1, 1 and their conjugates. □

Solvability of $P(x) = c$. We will now look at criteria under which conditions the polynomial equation $P(x) = c$ for $c \in \tilde{\mathbb{N}}$ can be solved.

Lemma 13. *Consider the polynomial $P(x) = []_{k \in V} \alpha_k x_i$ with exponents $V \subset \mathbb{N}$, coefficients $\alpha_k \in \tilde{\mathbb{N}}$ and $|V|$ odd. In addition, we have $O \subset V$ the odd exponents of V,* i.e. *we have $O := \{k \in V : 2 \nmid k\}$. Let $c \in \tilde{\mathbb{N}}$ and donsider the equation*

$$P(x) = c\,.$$

If $|O|$ is odd, then $P(x) = c$ always has a unique solution over $d\underset{\cdot}{\mathrm{Fl}}\mathrm{Int}\dot{\mathrm{s}}$.

Proof. Let $P'(x) := []_{k \in V, k \geq 1} k \alpha_k x^{k-1}$ be the *derivative* of $P(x)$. As x and all α_k are odd, we have

$$\alpha_k x^{k-1} \equiv 1 \pmod 2\,.$$

Therefore, we need to concentrate on the exponents k to derive the parity of the derivative $P'(x)$. We hence consider

$$P'(x) \equiv []_{k \in V} k \pmod 2\,.$$

Inspecting the tally we see that all even exponents contribute exactly 0 while every odd exponent contributes exactly 1. We therefore only consider the set of odd exponents O and obtain

$$P'(x) \equiv |O| \equiv 1 \equiv c \pmod 2 .$$

As c is odd, this equation always holds.

In addition we observe that 1 is always a solution modulo 2 as we have

$$P(1) = []_{k \in V} \alpha_i \equiv []_{k \in V} 1 = |V| \equiv 1 \equiv c \pmod 2 .$$

Therefore the prerequisites of the Hensel lifting from Lem. 5 are met as we have

$$P'(x) = 1, P(x) = 1 \pmod 2$$

. and we can hence use it to compute a unique solution. □

The above lemma seems to be in violation of Lem. 10 as it states solution and conjugate solutions, *i.e.*a total of two solutions. However, when we look at the prerequisistes, we that both differ in the number of odd exponents O. For Lem. 10, we need $|O|$ to be even while for Lem. 13, we need $|O|$ to be odd. Similarly, we have proven in Lem. 11 that $P(x) = c$ implies $P(-x) = c$. Again we see that this lemma can only be true if $|O|$ is even. Hence, all three lemmata are consistent with each other.

5 Security Reductions

To see how $\mathrm{d}_{\mathrm{I\!L\!F}}\dot{Int}s$ relate to problems that are already known to be hard, we start with investigating $\mathcal{M}$ultivariate $\mathcal{Q}$uadratic equations or $\mathcal{MQ}$ for short. Our goal is to recude $\mathcal{MQ}$ over GF(2) to $\mathcal{MQ}$ over $\mathrm{d}_{\mathrm{I\!L\!F}}\dot{Int}s$. To this aim, we want to establish a NOT-XOR-AND-basis for Boolean values—which naturally transfer to operations in GF(2).

After formally, defining these two $\mathcal{MQ}$-problems, we start with a negative result though, namaly that the AND-operator cannot be constructed using polynomials alone over $\mathrm{d}_{\mathrm{I\!L\!F}}\dot{Int}s$.

5.1 $\mathcal{M}$ultivariate $\mathcal{Q}$uadratic Equations

We first define $\mathcal{MQ}$ over GF(2) and $\mathcal{MQ}$ over $\mathrm{d}_{\mathrm{I\!L\!F}}\curvearrowleft Fields$.

Definition 6. *Let $n, m \in \mathbb{N}$ be the number of variables n and the number of equations m, respectively. Let $\alpha_e, \gamma_{e,i,j} \in$ GF(2) for $0 \leq e < m$ indicating the equation and $0 \leq i < j < n$ indicate the quadratic coefficient for the variables x_i, x_j, respectively. Then the following system of equations forms an $\mathcal{M}$ultivariate $\mathcal{Q}$uadratic Equations Problem over the finite field GF(2), or $\mathcal{MQ}$-GF(2) for short.*

$$\begin{cases} \sum_{i=0}^{n-1} \sum_{j=i+1}^{n-1} \gamma_{0,i,j} x_i x_j + \alpha_0 & = 0 \\ \vdots & \vdots \\ \sum_{i=0}^{n-1} \sum_{j=i+1}^{n-1} \gamma_{n-1,i,j} x_i x_j + \alpha_n - 1 & = 0 \end{cases}$$

Remark. There are two things two mention here. First, we can move the constant α_e freely between the l.h.s. and the r.h.s. in equation e as we have 1 = -1 in GF(2) and hence substraction is addition. This allows us to easily normalize the system of equations to have all constants α_e on the l.h.s. Second, by introducing two variables a, b and the equation $ab = 1$, we could remove all linear terms and make them quadratic terms by multiplying them by a or b, respectively. This does not change fact if the original system with linear terms has a solution—nor the solution itself (except of course the two variables $a = b = 1$, but we can easily discard those).

Similar to Def. 6, we now define the $\mathcal{MQ}$-problem over $\mathrm{d}_{\mathrm{I\!L\!F}}\curvearrowleft Fields$.

Definition 7. *Let $n, m \in \mathbb{N}$ be the number of variables n and the number of equations m, respectively. Consider the variables $V := x_0, \ldots, x_{n-1}$. For $0 \leq i < m$ let L_i be a left tally. Let N be the maximal number of terms in each tally, $k \in \mathbb{N}, 0 \leq k < N$ and consider the monomials $\mu_{i,k} := \alpha_{i,k} \cdot \beta_{i,k}$ with*

$$\alpha_{i,k} \in \tilde{\mathfrak{C}} \text{ and } \beta_{i,k} \in \tilde{\mathfrak{C}} \cup \{\alpha x : \alpha \in \tilde{\mathfrak{C}}, x \in V\} \cup \{\alpha xy : \alpha \in \tilde{\mathfrak{C}}, x, y \in V\}\,.$$

Note that we have an odd number of monomials in L_i, i.e. *is $|L_i|$ an odd number. Define the right tally R_i accordingly.*

Then the following system of equations forms an $\mathcal{M}$ultivariate $\mathcal{Q}$uadratic Equations Problem over $\mathrm{d}_{\mathrm{I\!L\!F}}\curvearrowleft Fields$, or $\mathcal{MQ} - \tilde{\mathfrak{C}}$ for short.

$$\begin{cases} []_{\mu in L_1} \mu & = []_{\mu in R_1} \mu \\ \vdots & \vdots \\ []_{\mu in L_{m-1}} \mu & = []_{\mu in R_{m-1}} \mu \end{cases}$$

We note that monomial can repeat in the above definition, *e.g.* x_0 and $3x_0$ can be part of the same tally. We see below that we will need this property for our reduction. By collecting terms, we can achive $\binom{n}{2} + 2n + 1$ as upper bound on the number of monomials per tally.

Note that we will need the linear terms $\alpha_{i,k}x_k$ for our reduction. We hence do not remove them in our definition. Secondly, this definition specializes easily for $\mathrm{d}_{\mathrm{I\!L\!F}}\dot{Int}s$ $(\mathcal{MQ} - \tilde{\mathbb{N}})$ and $\mathrm{d}_{\mathrm{I\!L\!F}}\check{P}olys$ $(\mathcal{MQ} - \tilde{\mathbb{P}})$.

5.2 Reducing $\mathcal{MQ}$-GF(2) to $\mathcal{MQ} - \tilde{\mathbb{N}}$

We want to construct a Boolean circut over $\mathrm{d}_{\mathrm{I\!L\!F}}\dot{Int}s$. Our encoding $E(b) : GF(2) \to \tilde{\mathbb{N}}^d$ is

$$E(b) := \begin{cases} 0 \to 1 \\ 1 \to q+1 \end{cases}$$

There is also a closed formula, namely $E(a) = 1 + qb$.

Lemma 14. *Denote $d \in \mathbb{N}, d \leq 2$ with $\tilde{\mathbb{N}}^d$ the set of elements in this $\mathrm{d}_{\mathrm{I\!L\!F}}\dot{\mathrm{I}}\mathrm{n}\dot{\mathrm{t}}\mathrm{s}$. In addition we have $q = 2^d$ and $D = 2q = 2^{d+1}$.*

XOR-gate: $X(x, y) = xy$

AND-gate: $z = A(x, y)$ *with*

$$\begin{aligned} x &= [1, -c, -1, cr_x, r_x] \\ y &= [1, -c, -1, cr_y, r_y] \\ z &= [1, -c, -1, cr_z, r_z] \\ [r_z, r_z, 1] &= [r_x r_y, -r_x, -r_y, 3, 1] \end{aligned}$$

Proof.
XOR-gate. Our goal is to find a polynomial equation such that $z = X(x, y)$ is the XOR-function over GF(2). Consider $i = 1 + qb_i$ and $b_i \in \{0, 1\}$. As all computations are performed modulo 2^{d+1}, all multiples of 2^{d+1} reduce to zero in this ring. Consider

$$\begin{aligned} xy &= (1 + qb_x)(1 + qb_y) \\ &= 1 + q(b_x + b_y) + q^2 b_x b_y \\ &\equiv 1 + q(b_x + b_y) \end{aligned}$$

Assume $b_x = b_y = 1$. In this case, $q(b_x + b_y) = 2q = 2^{d+1}$ and hence reduces to zero. Therefore,

$$xy = 1 + q(b_x \oplus b_y) \pmod{2^{d+1}}.$$

This yields exactly the desired function.

AND-gate. We want to construct some function $z = A(x, y)$ such that $b_z = b_x b_y$. We start with introducing an auxiliary variable $r_i \in \tilde{\mathbb{N}}$ satisfying

$$i = 1 - 2^{d-1} + 2^{d-1} r_i .$$

Using some constant $c := 2^{d-1} - 1$ with $c \in \tilde{\mathbb{N}}$ we can rewrite this as

$$i = [1, -c, -1, cr_i, r_i]$$

We verify correctness via back-substitution:

$$\begin{aligned} [1, -c, -1, cr_i, r_i] &= 1 - c - 1 + cr_i + r_i \\ &= -c + 2^{d-1} r_i \\ &= 1 - 2^{d-1} + 2^{d-1} r_i \\ &= 1 - 2^{d-1} + 2^{d-1}(2b_i + 1) \\ &= 1 + qb_i = x \end{aligned}$$

We aim for $b_z = b_x b_y$ and hence $r_z = 2(b_x b_y) + 1$. Consider

$$2r_z = r_x r_y - r_x - r_y + 3$$

We can verify this identity via

$$\begin{aligned} r_x r_y - r_x - r_y + 3 &= (2b_x + 1)(2b_y + 1) - (2b_x + 1) - (2b_y + 1) + 2 \\ &= 4b_x b_y + 2 \\ &= 2(2b_x b_y + 1) \\ &= 2r_z \end{aligned}$$

This leads to the tally

$$[r_z, r_z, 1] = [r_x r_y, -r_x, -r_y, 3, 1]$$

As we have $r_i \in \{1, 3\}$, we need a decoder that maps $1 \to 1, 3 \to 1 + q$. We construct this as follows: $i = [1, -c, -1, cr_i, r_i]$. We verify correctness by inspecting the two cases $r_i \in \{1, 3\}$.
For $r_i = 1$ we get $i = 1 - 2^d + 2^d = 1$.
For $r_i = 3$ we get $i = 1 - 2^d + 3.2^d = 1 + 2^d$.
In summary, we obtain the following tallys:

$$\begin{aligned} x &= [1, -c, -1, cr_x, r_x] \\ y &= [1, -c, -1, cr_y, r_y] \\ z &= [1, -c, -1, cr_z, r_z] \\ [r_z, r_z, 1] &= [r_x r_y, -r_x, -r_y, 3, 1] \end{aligned}$$

We have introduced the 4 intermediate variables r_x, r_y, r_z, z and 4 new equations. □

Corollary 7. *Consider the tally*

$$i = [1, -c, -1, cr_i, r_i] .$$

with $i, r_i \in \tilde{\mathbb{N}}$. Then we have $i \in \{1, q + 1\}$.

Proof. Revisiting the above proof, we have $i = 1 + 2^{d-1}(r_i - 1) \pmod{2^{d+1}}$. As r_i is odd, we can write $r_i = 2k + 1$ for some integer k. Substituting we obtain

$$i = 1 + 2^{d-1}(2k + 1 - 1) = 1 + 2^d k \pmod{2^{d+1}} .$$

We not inspect the term $2^d k$. As all computations are performed modulo 2^{d+1}, the only two possible values for $2^d k$ are 0 and q, regardless of k and hence $i \in \{1, 2^d + 1\}$. □

We are now ready to transfer a system of equations $\mathcal{MQ}$-GF(2) to $\mathcal{MQ} - \tilde{\mathbb{N}}$. We start with one polynomial.

Lemma 15. *Let $P(X) = \sum_{i=0}^{n-1} \sum_{j=i+1}^{n-1} \gamma_{i,j} X_i X_j + \alpha$ with $\alpha, \gamma_{i,j} \in$ GF(2). We can transfer $P(X)$ to $d\mathcal{LI}$Inṫs of dimension $d \geq 2$ using $2n + 4o - 2$ variables and $n + 4o - 1$ equations.*

Proof. W.l.o.g. let us assume that the input contains neither the all-zero equation (0=0 nor the equation (1=0). The former is a tautology, whereas the latter is unsatisfiable. Hence, both equations can be removed from the instance. Consequently, every remaining equation contains at least one variable. Note that this means that each equation contains at least one variable.

All variables over $\tilde{\mathbb{N}}$ are denoted with x_i for $0 \leq i < n$. We now inspect the quadratic terms $X_i X_j$ of the polynomial $P(X)$. Whenever we have $\gamma_{i,j} = 1$, we add the corresponding

AND-gates from Lem. 14. Denote with $0 \leq o \leq \binom{n}{2}$ the number of non-zero quadratic terms in the GF(2)-polynomial $P(X)$. Denote the intermediate variable with $z_{i,j}$. We store them in a list Z. Note that we have $|Z| = o$.

Assume we have exactly one quadratic term in the input. THen we can directly form the equation $z_0 = E(\alpha)$. Alternatively, we have at least two quadratic terms in the original polynomial $P(X)$. Assuming we have at least two terms we we introduce XOR-gates to sum up all intermediate variables from Z. We do this via

$$a_0 := Z_0 \oplus Z_1, a_k = a_{k-1} \oplus Z_{k+1} \text{ for } 1 \leq k \leq o-2\,.$$

We need a total of $o-1$ such gates and $o-1$ new intermediate variables a_k for $0 \leq k < o-1$.

Finally, we encode the constant $\alpha \in \mathrm{GF}(2)$ using the embedding $E(.)$. Add one final equation $a_{o-2} = E(\alpha)$.

Last but not least, we add a filter equation from Cor. 7 for all outer variables. These are

$$x_0, \ldots x_{n-1}, z_0, \ldots z_{o-1}, a_0, \ldots a_{o-2}\,.$$

Therefore, we obtain a total number of $n + 2o - 1$ variables and also equations. We add new variables r_i with $0 \leq i < n + 2o - 1$ as intermediate filter variables.

Layer	Variables	Equations
Initial	n	-
AND	o	o
XOR	$o-1$	$o-1$
α	-	1
Filter	$n+2o-1$	$n+2o-1$
Σ	$2n+4o-2$	$n+4o-1$

□

With these numbers, the overall endeavour seems hopeless: The number of variables is much higher than the number of equations. However, we have only transfered *one* equation, so for each additional equation we will save variables both in the layer Initial as well as Filter. Hence we reinspect the above lemma, but now for a full system of equations.

Corollary 8. *Let $\mathcal{P}$ be a system of Multivariate Quadratic polynomial equations over GF(2). In n variables and m equations. We have $\mathcal{P} = (P_0, \ldots, P_{m-1})$.*

Proof. We construct the transfer for all equations simultaneously. The essential difference from the single-equation construction in Lem. 15 is that the variables in the initial layer and the corresponding filter variables and equations are shared by all input equations $P_0(X) = 0, \ldots, P_{m-1}(X) = 0$. As before, we exclude $0 = 0$ (tautologgy) and $0 = 1$ (not satisfiable).

First, we introduce n variables $x_0, \ldots, x_{n-1}$ corresponding to the Boolean variables $X_0, \ldots, X_{n-1}$. These variables are introduced only *once* for the entire system $\mathcal{P}$.

Now consider an arbitrary polynomial $P_i(X)$. For every non-zero, quadratic coefficient $\gamma_{i,j,k} = 1$, we introduce the corresponding AND-gate from Lem. 14. Denote its intermediate variable $z_{i,j,k}$. If the polynomial $P_i(X)$ contains o_i non-zero quadratic terms, this gives o_i new AND variables and also o_i AND equations. We collect these AND outputs variables in a list $Z_i = (Z_{i,0}, \ldots, Z_{i,o_i-1})$ Thus, the list Z_i contains exactly o_i elements.

Because we have excluded both the tautological equation $0 = 0$ and the unsatisfiable equation $1 = 0$, every equation contains at least one quadratic term. Consequently, Z_i is non-empty and we have $o_i \geq 1$.

If $|Z_i| = 1$, there is no XOR and we can hence connect the output variable of this layer with the constant term. Consequently, we add the equation $Z_{i,0} = E(\alpha_i)$ directly.

Now suppose we have $o_i \geq 2$. This means we have to connect all elements of the list Z_i via

$$a_{i,0} := Z_{i,0} \oplus Z_{i,1}, a_{i,k} = a_{i,k-1} \oplus Z_{0,k+1} \text{ for } 1 \leq k \leq o-2\,.$$

Transfering these equations over GF(2) to $\tilde{\mathbb{N}}$ requires $o_i - 1$ such gates and $o_i - 1$ new intermediate variables $a_{i,k}$ for $0 \leq k < o_i - 1$.

Finaly, we add $a_{i,o_i-2} = E(\alpha_i)$ to capture the constant term of the GF(2)-polynomial $P(X)$.

To recapture, we have added a total of o_i AND variables and $o_i - 1$ XOR variables and equations, respectively. In addition, we needed one extra equation to capture the constant term.

Now consider the complete system. Let O be the number of non-zero quadratic terms. We have $O := \sum_{i=0}^{m-1} o_i$.

Layer	Variables	Equations
Initial	n	-
AND	O	O
XOR	$O - m$	$O - m$
α_i	-	m
Filter	$n + 2O - m$	$n + 2O - m$
Σ	$2n + 4O - 2m$	$n + 4O - 2m$
without filter	$n + 2O - m$	$2O$

□

As the number of equations is smaller than the number of variables, the system seems underdetermined. However, the newly introduced filter variables can only take on the values $1, 3$, see Lem. 14. Hence, it is more meaningful to compare the number of equations and variables for the sum of all other layers. Here we see that the number of equations exceeds the number of variables for $m > n$. This is exactly the desired behaviour for the original system over GF(2).

Finally, we can complete the proof that $\mathcal{M}$ultivariate $\mathcal{Q}$uadratic systems of equations over $\mathfrak{dl}\dot{I}nt\dot{s}$ are not only $\mathcal{NP}$-hard but also $\mathcal{NP}$-complete.

Corollary 9. *Given a system $\mathcal{P}$ from $\mathcal{MQ} - \tilde{\mathbb{N}}$. After guessing n variables, we can verify for all m equations is the solution is correct. This can be done in polynomial time. Using Cor. 8 we have established that $\mathcal{MQ} - \tilde{\mathbb{N}}$ is $\mathcal{NP}$-complete.*

Note that all proves can also be adapted to $\mathfrak{dl}\breve{P}olys$ so the previous corollary can be rephrased that both $\mathcal{MQ} - \tilde{\mathbb{N}}$ and $\mathcal{MQ} - \tilde{\mathbb{P}}$ are $\mathcal{NP}$-complete.

Specifically for $d = 2$, we can simplify the above proof as a reduction from 3-SAT to $\mathfrak{dl}\dot{I}nt\dot{s}$.

Lemma 16. *Let $B_0, \ldots, B_{n-1}$ be Boolean variables. We have a 3-SAT instance with m clauses $C_i := (L_{i,0} \vee L_{i,1} \vee L_{i,2})$ for each literal $L_{i,k} = B_j$ or $L_{i,k} = \overline{B_j}$. We can transfer it in polynomial time to an $\mathcal{MQ}$-fInt-instance.*

Proof. We start with the embedding

$$E(B) = \begin{cases} 1, 5 & \text{for } B = 0 \\ 3, 7 & \text{for } B = 1 \end{cases}.$$

Note that we can ignore the upper bit and hence write its inverse function as

$$E^{-1}(x) = \begin{cases} 0 & \text{for } x \equiv 1 \pmod 4 \\ 1 & \text{for } x \equiv 3 \pmod 4 \end{cases} .$$

All transfer gates will be constructed accordingly.

NOT-gate. We consider $x + y = 4\text{mod } 8$. We see that this equation gives us the desired result inspecting the following truth table:

x	y	$\overline{X}$	X
1	3	1	0
3	1	0	1
5	7	1	0
7	5	0	1

Rewriting the above equation into a tally yields

$$[x, y, 1] = 5\,.$$

AND-gate. Consider $uv + 3 = u + v + 2t$ or

$$[uv, 3, 1] = [u, v, t, t, 1]\,.$$

For arbitrary values $u, v, t \in \tilde{\mathbb{N}}$, this tally equation is equivalent to

$$E^{-1}(t) = E^{-1}(u)E^{-1}(v)\,.$$

To see why this is the case rearrange the equation as

$$2t = uv - u - v - 3 \pmod 8$$

Dividing both sides by 2 yields

$$t = \frac{uv - u - v - 3}{2} \pmod 4 \ .$$

For given input u, v, the nominator is always even, so the division is well-defined over $\mathbb{Z}$. Hence, as we move from modulo 8 to modulo 4, it is enough to consider the two lower bits. We now inspect the corresponding truth table:

u	v	t	AND	U	V
1	1	1	0	0	0
1	3	1	0	0	1
3	1	1	0	1	0
3	3	3	1	1	1

So ignosing the highest bit does indeed compute the AND function and we have $T = E(t) = UV$ for $uv + 3 = u + v + 2t$. We note that we need one tally equation and one auxiliary variable per AND gate.

OR-gate. We observe that we have $X \vee Y = \overline{(\overline{X} \wedge \overline{Y})}$. Using both the AND- and the NOT-gate we obtain the following two equations for $\text{OR}(X, Y) = Z$

$$(4 - x)(4 - y) + 3 = (4 - x) + (4 - y) + 2t \text{ and } z + t = 4\,.$$

For the first equation, expanding and collecting terms leads to

$$xy + 5x + 5y = 2t + 5\,.$$

This leads to the following two tallys

$$[xy, 5x + 5y] = [t, t, 5] \text{ and } [z, t, 1] = 5$$

for $\mathrm{OR}(X, Y) = Z$ over $\mathrm{d}\dot{\imath}\mathrm{l}\mathrm{\exists}\mathit{Int}\dot{s}$. Note that we need one auxilary variable t for each OR-gate.
Full reduction. First we note that the NOT-gate is a linear operations as we have $t = 4 - x$. Hence, we can simply substitute for each $\overline{X}$ the expression $4-x$, simplify and collect. Therefore, we do not need to introduce extra variables for negative literals.

Second, we construct two OR-gates for each clause, adding one extra auxiliary variable. Our construction is $A_i = L_{i,0} \vee L_{i,1}$ and $B_i = L_{i,2} \vee A_i$ for clause C_i. This way, each clause is transferred with 4 equations and 2 extra variables.

As every clause produces exactly one equation and the full system is only satisfied if all equations are satisfied, we do not need the final AND. It sufficies to introduce the equation

$$B_i = E(\text{true}) = 3$$

for each clause. This asds an extra m equations.

Counting the number of variables and equations yields:

Layer	Variables	Equations
Initial	n	-
Clauses	2m	4m
final AND	-	m
Σ	n+4m	5m

□

Corollary 10. *Assume we have a system of equations over* $\mathrm{d}\dot{\imath}\mathrm{l}\mathrm{\exists}\mathrm{Int}\dot{s}$. *Then we can lift this solution σ from d to $d + 1$.*

Proof. The number of variables is n. Consider the set of variables $X := x_0, \ldots, x_{n-1}$. We now have the set of monomials $M := \{1\} \cup X \cup \{x_i x_j : x_i, x_j \in X : i \leq j\}$. The number of terms in P is T. Consider $\mu \in M^T$ the list of all monomials. Note that μ might contain double entries. Similarly consider the list of all coefficients as $\alpha \in \tilde{\mathbb{N}}^T$.

Let $P(x) := []_{0 \leq k < T}(\alpha_k \mu_k)$ be our multivariate polynomial. Define $Q(x)$ similarly for μ', α' and T'.

Assume that our system of equations over $\tilde{\mathbb{N}}^d$ has some solution $\sigma \in \tilde{\mathbb{N}}^n$. We now show how we can lift this solution to $\tilde{\mathbb{N}}^{d+1}$. Consider

$$P(x) = Q(x) \bmod (2^{d+1})$$

By multiplying by 2 we lift this equation to $\tilde{\mathbb{N}}^{d+1}$ and hence modulo 2^{d+2} as we have

$$2 \sum_{0 \leq i < T} \alpha_i \mu_i = 2 \sum_{0 \leq k < T'} \alpha'_k \mu'_k \pmod{2^{d+2}}.$$

At first glance, this is not a valid tally, however, we can write

$$[P(x), P(x), 1] = [Q(x), Q(x), 1]$$

which is a valid tally, as can be seen after expanding both $P(x)$ and $Q(x)$ and counting the terms. Note that we have $2T + 2T' + 2$ terms in total with $2T + 1$ on the l.h.s. and $2T' + 1$ on the r.h.s., so the tally condition is satisfied. Moreover consider

$$2P(x) \equiv 2Q(x) \pmod{2^{d+2)}}$$

This is equivalent to

$$P(x) \equiv Q(x) \pmod{2^{d+1}}$$

Hence lifting by multiplication of 2 preserves the solution condition. Note that we might obtain more than one solution for σ as both σ_i or $\sigma_i + 2^{d+1}$ can be valid over $\tilde{\mathbb{N}}^{d+1}$. Note that these are the conjugates from Def. 5.

Applying this transformation to each polynomial $P_i(x)$ and $Q_i(x)$ in the original system of equations yields the result. □

6 Generalizations

After having looked at $\mathrm{d}_{\text{IJ}}\text{F}\curvearrowleft Fields$ in detail, we now look at popential generalizations.

6.1 General Basis b

Consider some basis $b \in \mathbb{N}, b \geq 1$. We define

$$\tilde{\mathbb{N}}_b^n := \{i \in \mathbb{N} : i = 1 \pmod b, i < b^{n+1}\}.$$

Generalize the tally operator accordingly for $k \in \mathbb{N}, k \geq 1, k \equiv 1 \bmod b$ and $n \in \mathbb{N}, n \geq 1$ as

$$[a_1, \ldots, a_k]_{k,b}^n := \sum_{i=0}^{k-1} a_i \mod b^{n+1} .$$

Note that for $b = 1$, the tally coincides with regular addition over the integer ring modular 2^{n+1} while $b = 2$ leads to $\mathrm{d}_{\text{IJ}}\text{F}\curvearrowleft Fields$. Obviously, all of the above proves and definitions carry over, *i.e.* the idea is generalized to structures for $b \geq 3$. While certainly interesting on their own rights, we concentrate on the case $b = 2$ for cryptographic applications, as this yields the most compact and efficient representation while leading to a structure different from Finite Fields at the same time.

6.2 Matrices

We already used matrices in several parts of the paper. As long as the number of rows and columns is odd (or equivalent 1 mod b for some $b \geq 2$), the corresponding matrix and matrix-vector multiplication are defined. However, we have difficulties to define the multiplicative inverse of a matrix here.

$\text{uəd}_{\mathbf{O}}$ Problem 16

i. How does the group of invertible matrices over a $\mathrm{d}_{\text{IJ}}\text{F}\curvearrowleft Fields$ look like?

ii. How can we characterize its elements efficiently and compute multiplicative inverses of matrices directly?

6.3 Polynomials

Defining polynomials over $\mathrm{d}_{\text{IJ}}\text{F}\curvearrowleft Fields$ is not a problem. The problem arises when multiplying these polynomials, cf. Sect. 4.9. Moreover, adding two polynomials is not possible as this directly violates the tally operator. Hence, we need a structure that requires an odd number of polynomials to be added. All in all, the use of polynomials is difficult as polynomial multiplication is not closed within $\mathrm{d}_{\text{IJ}}\text{F}\curvearrowleft Fields$.

6.4 Extension Fields

Extension fields are a natural extension of finite fields and hence the question arises if we can also extend $\mathrm{d}_{\text{IJ}}\text{F}\curvearrowleft Fields$. The answer is very much related to the existence of efficient polynomial multiplication algorithms, see the previous section and Sect. 4.9. As we saw in the latter, multiplying two polynomials yields closed equations which hence break the structure of polynomial-polynomial-multiplication. Unless there is a general structure of such polynomials, the chances of constructing extension fields over $\mathrm{d}_{\text{IJ}}\text{F}\curvearrowleft Fields$ is very slim. However—there

might be a tweak of the axioms that allows extension-field-like structures.

uədO Ǐroblem 17 Can we construct a structure similar to an extension field over dịlꟻ↶*Field*s or at least a structure with similar properties? If yes—how does it look like?

6.4.1 Inner⊙dịlꟻ↶Fields

While technically not an extension field, it nicely fits into the framework. Consider $d, \delta \in \mathbb{N}$ with $1 \leq \delta < d$. We denote the two carrieres as $\tilde{\mathfrak{C}}^{\delta}, \tilde{\mathfrak{C}}^{d}$. We see that $\tilde{\mathfrak{C}}^{\delta} \subset \tilde{\mathfrak{C}}^{d}$ is a proper subset. For any element $a \in \tilde{\mathfrak{C}}^{d}$, we compute $a' \mod 2^{\delta+1}$. In a sense, we embed the whole structure of the dịlꟻ↶*Field* associated with $\tilde{\mathfrak{C}}^{d}$ into the Inner⊙dịlꟻ↶Field associated with $\tilde{\mathfrak{C}}^{\delta}$. In a sense, this is the inverse operation to the Hensel lifting from Sect. 4.1.

Definition 8. *Let $\delta, d \in \mathbb{N} : 0 < \delta < d$, then we call the dịlꟻ↶Field associated with $\tilde{\mathfrak{C}}^{\delta}$ an* ***Inner⊙dịlꟻ↶Field*** *of the carrier $\tilde{\mathfrak{C}}^{d}$ associated with this dịlꟻ↶Field. Conversely, we call the dịlꟻ↶Field for this carrier $\tilde{\mathfrak{C}}^{d}$ an* ***Outer⋎dịlꟻ↶Field*** *for the dịlꟻ↶Field for the carrier $\tilde{\mathfrak{C}}^{\delta}$*

6.5 Complex numbers and Quaternions

To understand why extension fields do not work, we inspect both complex numbers and quaternions. We start with the complex numbers and then move on to quaternions. Alas, as we will see, none of these constructions works, mainly for reasons already outlined in the previous section.

Let $\alpha_i := a_i + b_i$ for some $i \in \mathbb{N}$ and $a_i, b_i \in \tilde{\mathbb{N}}$ and define $i^2 := -1$. Addition is defined as $\alpha_1 + \alpha_2 := (a_1 + a_2) + (b_1 + b_2)i$, *i.e.* we can easily extend this to our tally operator as a 3-nary operator. For inspecting multiplication, let

$$\alpha_1\alpha_2 = a_1a_2 + a_1b_2i + a_2b_1i - b_1b_2 = (a_1a_2 - b_1b_2) + (a_1b_2 + a_2b_1)i\,.$$

While each individual term is an odd number, each bracket contains an even number of terms and hence leads to even numbers as a result. In Sect. refsec:flipInts we had a similar problem with addition and solved it with the introduction of a ternary tally operator. We will briefly investigate if a ternary multiplication solves our problem here, too. Starting with $\alpha_1\alpha_2\alpha_3$ we obtain

$$(a_1a_2a_3 - a_1b_2b_3 - a_2b_1b_3 - a_3b_1b_2) + (a_1a_2b_3 + a_1a_3b_2 + a_2a_3b_1 - b_1b_2b_3)i\,.$$

Hence the same pattern as below—the number of terms for each bracket so this yields an even number in either case.

The final show-stopper comes when we look at the multiplicative inverse of a complex number. Here we have

$$\alpha^{-1} = \frac{1}{a+bi} = \frac{a-bi}{a^2+b^2}\,.$$

While both a^2 and b^2 or odd numbers, the sum $a^2 + b^2$ is not. Hence, we cannot compute the multiplicative inverse of $(a^2 + b^2)$ and therefore α^{-1}.

We are now moving on to Quaternions. They are different enough to complex numbers to yield a generic extension to dịlꟻ*Int̀s*. This structure has been formally described by the Irish mathematician William Hamilton in 1843 and independently in 1840 by the French mathematician Olinde Rodrigues. Also Carl Friedrich Gauss has independently discovered Quaternions in 1819, but this work was only published in 1900. All in all, they are mostly credited to Hamilton, *i.e.* the symbol for Quaternions is $\mathbb{H}$. It is a non-commutative extension to the complex numbers. Hence, the Hidden Subgroup Problem from Sect. 3.4.6 could benefit from the existence of

such a group. In particular, the multiplicative group of the Quaternions could be used here—if the Quaternions could be used as $\mathrm{d}\dot{\imath}\mathrm{l}\mathrm{\exists}\mathit{Int}\dot{s}$. For this purpose, we define

$$a + bi + cj + dk \text{ with } a, b, c, d \text{ come from } \mathrm{d}\dot{\imath}\mathrm{l}\mathrm{\exists}\mathit{Int}\dot{s} \text{ while } i, j, k \text{ are symbols.}$$

In this context, we set these symbols to $i^2 := -1, j^2 := -1, k^2 := -1$ and $ijk := -1$. As above, we write $\alpha := a + bi + cj + dk$. In the case of addition, this leads to

$$(a_1 + b_1 i + c_1 j + d_1 k) + (a_2 + b_2 i + c_2 j + d_2 k) = (a_1 + a_2) + (b_1 + b_2)i + (c_1 + c_2)j + (d_1 + d_2)k$$

Multiplying these numbers yields

$$(a_1 + b_1 i + c_1 j + d_1 k) \cdot (a_2 + b_2 i + c_2 j + d_2 k) =$$

$$a_1 a_2 - b_1 b_2 - c_1 c_2 - d_1 d_2 + (a_1 b_2 + b_1 a_2 + c_1 d_2 - d_1 c_2)i$$
$$+(a_1 c_2 - b_1 d_2 + c_1 a_2 + d_1 b_2)j + (a_1 d_2 + b_1 c_2 - c_1 b_2 + d_1 a_2)k$$

Unfortunately, all elements are even now, see Sect. 4.9 for a more detailed discussion.

Even worse, for computing the multiplicative inverse of some α, we need to compute

$$\alpha^{-1} = \frac{a - bi - cj - dk}{a^2 + b^2 + c^2 + d^2}\,.$$

As the denominator is the sum of 4 odd numbers, it becomes even and hence, the multiplicative inverse is not defined over $\mathrm{d}\dot{\imath}\mathrm{l}\mathrm{\exists}\mathit{Int}\dot{s}$. Hence, we cannot use Quaternions for our purposes.

All of the above arguments can be easily adapted to $\mathrm{d}\dot{\imath}\mathrm{l}\mathrm{\exists}\breve{P}\mathit{olys}$.

7 Concrete Implementations

7.1 Motivation

While the Python code has helped in the understanding of dịlƎ↶*Field*s, it s not a reference implementation. However, it follows the path called *Falsifiability* outlined by Karl Popper [Po35]. Interestingly, this philosophical concept resonates well with the software engineering principles of rapid prototyping, test driven development and unit testing [Be99]. All of those were employed during the development of the supporting Python code. For example, the function `rndMatTest` is used to determine if the heuristic outlined in Sect. 4 or function `solveMatVec` are sufficient to solve matrix/vector equations $Mx = y$ for given matrix $M \in \tilde{\mathbb{N}}^{s\times s}, y \in \tilde{\mathbb{N}}^s$ and some integer carrier $\tilde{\mathbb{N}}$. As you can see in App. B.1, there is a sharp decline in the solvability of the above equation between $s = 11$ and $s = 13$—so the answer is negative. The reason is an open question but might be explained in another paper.

However, using the insights of [Po35, Be99] by actively seeking *Falsifiability* rather than *Validation*, it was possible to disregard this idea very quickly and move on to LU decomposition, see App. B.2 and later Bespoken LU decomposition, see App. A.2. So using empirical methods did foster mathematical theory, but do not replace its rigour. In the words of Karl Popper, `solveMatVec` presents an initial hypothesis that was *Falsified* and hence replaced with a better theory, *i.e.* LU decomposition from App. B.2. To identify this false hypothesis, using empirical testing was more efficient method than using a mathematical prove—as only one contradicting example shakes a theory to the ground. The opposite is not true, though: No matter how much empirical evidence is presented, it does not constitute theoretical proof. The latter must be constructed with mathematical rigour.

Note that the Python code was optimized for maximal correctness, not for speed nor usability in actual cryptographic schemes. For example, the topic of side channel attacks or cryptographic secure number generation has been fully ignored. In particular, the arithmetic of dịlƎ*P̆olys* has not been optimized—hence it took nearly three times longer to generate the polynomials for dịlƎ*P̆olys* in comparison to dịlƎ*Inṫs* for Sect. 4.6, see App.C for the exact timings. With a full optimized implementation, it is rather expected that dịlƎ*P̆olys* outperform dịlƎ*Inṫs* for concrete implementations. Hence, it can only form the *basis* for a careful secure and efficient implementation of potential cryptosystems. Some considerations on efficient implementation can be found the following section.

7.2 Python Implementation

This section gives an initial idea of the Python implementation for dịlƎ↶*Field*s, dịlƎ*Inṫs*, and dịlƎ*P̆olys*. You see the corresponding files in Tab. 20. In each file, the function `testAll` is a

Table 20: Overview of the files of the Python implementation (Version 1.15).

Structure	Filename	number of lines
dịlƎ↶*Field*	`flipFields.py`	2,367
dịlƎ*Inṫs*	`flipInts.py`	4,940
dịlƎ*P̆olys*	`flipPolys.py`	279

good starting point as it shows the expected input and output behaviour of all functions in the corresponding file.

The implementation makes strong use of the Python `assert` function to see if the input of a function is valid. We can see this with the following functions:

isRing(D) Is D odd and larger than 3? Is D of the form 2^{d+1}? Or in other word—does D represent some valid integer carrier? See Sect. 2.1 for the formal definition of an integer carrier.

isElem(a,D) Is a an element of $\tilde{\mathbb{N}}^{d+1}$ for $D = 2^{d+1}$?

isVec(v,D) Checks if the coefficients of a given vector v are actually from the $\text{dịlƎ}\mathit{Int\dot{s}}$ $\tilde{\mathbb{N}}^d$ for $D = 2^{d+1}$.

isPoly(p,D) Checks if all coefficients of the polynomial p come from the $\text{dịlƎ}\mathit{Int\dot{s}}$ $\tilde{\mathbb{N}}^d$ for $D = 2^{d+1}$. The polynomial is implemented as a `dict` in Python.

Both in `flipInts.py` and `flipPolys.py`, single elements are represented as numbers. We show this with the following examples:

Bitstring	Number	Polynomial
0001_b	1	1
0011_b	3	$X + 1$
0101_b	5	$X^2 + 1$
1101_b	13	$X^3 + X^2 + 1$

This allows for a very compact representation of both numbers ($\text{dịlƎ}\mathit{Int\dot{s}}$) and polynomials ($\text{dịlƎ}\breve{P}\mathit{olys}$). However, the least significant bit is always 1 in this representation. This leads to an optimization: So we do a "compact" representation of the elements by skipping the last bit—at the cost of speed as our implementation needs to "add" this last bit during the computation. Or do we opt for a "fast" implementation that uses more memory.

uədO Ǐroblem 18 What is the optimal point between a compact or a fast implementation in the current case? Are there differences between software and hardware implementations?

7.3 Symbolic Computation

In the file `flipInts.py`, there is a small symbolic computation engine. Its main purpose is to investigate properties of $\text{dịlƎ}\mathit{Int\dot{s}}$, with focus on the linear case. It is kept lean while still allowing meaningful experiments. First, all computations are done on polynomials. Each polynomial is represented as a string, and this as a sum of terms. For example, we have

`1*a*` or `0*+1*a*+3*a*b*`.

Within a term, the first item is always the coefficient, followed by the variables, in alphabetical order. Each term is terminated by an asterix (*). For example the following terms are not well-formed:

A. `a*b*` Starts with a variable, not a number.

B. `3*a*b` Misses the terminating asterix (*).

C. `3*b*a*` Not alphabetically sorted.

All methods in this context start with `sym`. For example, `symMultMatMat(A,B,D)` takes two matrices with $A \in \tilde{\mathbb{N}}^{l\times m}$ and $B \in \tilde{\mathbb{N}}^{m\times n}$ and computes the result as $C := A.B$ for $C \in \tilde{\mathbb{N}}^{l\times n}$. `symMultMatMat(A,B,D)` also allows the coefficients of the input matrices A, B to be symbolic and hence produces a matrix C with polynomial entries. An example of such a matrix can be found in Sect. 4.5.

A more powerful method is `symRewriteEqsLin`. It aims at solving the problem for the multiplication of polynomials, see Sect. 4.9. Here we try to eliminate all sums with an even number of terms.

Finally, there is `symSolveLinSys(eqs,D)`. While we cannot solve linear equations in all cases over $\mathrm{d}\underline{\mathrm{i}}\underline{\mathrm{I}}\mathit{Int}\dot{s}$, there are still some that can be solved, *e.g.* for the input system having a triangular structure. In a nutshell, the algorithm selects the shortest equation, extracts one variable and applies it to the other equations. It is basically ElimLin [CS+12, QW15] which has been slightly tweaked for the use over $\mathrm{d}\underline{\mathrm{i}}\underline{\mathrm{I}}\mathit{Int}\dot{s}$. Note that we have intermediate equations with coefficients divisible by 2. We take care of this by dividing each equation by the gcd of all coefficients in the corresponding polynomial. To name but a few, there are several helper functions for `symSolveLinSys`. The parameter D is always 2^{d+1} for the dimension of the corresponding $\mathrm{d}\underline{\mathrm{i}}\underline{\mathrm{I}}\mathit{Int}\dot{s}$.

symIsPoly(p, v, D, relaxed) Verifies if p is actually a polynomial with the given variables in the set v. If relaxed is set, even numbers are also allowed as coefficients of the polynomial.

symSubtractTerm(eqStr, subStr, D) Remove a given term from an equation. Is used to extract a single variable.

symMultFacPoly(f, p, v, D) Multiplies a given polynomial p by an integer factor f. The polynomial must only contain variables from the set v.

symSetEqsPoly(eqs, v, var2poly, D) Replaces a given variable by a polynomial as specified in the dictionary var2poly. Multiplies out and collects terms in the process. The result is a valid polynomial in the variables given by the set v.

symSetEqs(eqs, v, var2val, D) For a given list of equations, replace one variable by an integer value, as given in the dictionary var2val.

symIsSol(eqs,sol,D) For a system of equations, verify that the solution is actually correct.

The concrete algorithm can be found in `flipInts.py`, with several testcases in the method `testAll`.

8 Discussion / Conclusions / **uədO P̆roblems**

We have investigated dịɭɟ↶*Field*s, dịɭɟ*Inṫs*, and dịɭɟ*P̆olys* as an alternative to Finite Fields. First, computing multiplicative inverses via the Hensel Lifting is more efficient than using the Extended Euclidean Algorithm, cf. Sect. 4.1. Second, we want to increase the variety of potential algorithms by changing their basic Finite Fields to a different structure so cryptanalysts need to expand their effort. Third, there is a mild hope that cryptanalytic algorithms will become more difficult for structures over dịɭɟ↶*Field*s: We saw throughout the paper that algorithms such a Gaussian elimination or polynomial multiplication do not work anymore as before and assume this carries through to cryptanalytic attacks against the corresponding schemes.

We have studied several cryptographic primitives to see if they could be adopted to dịɭɟ↶*Field*s. For Unbalanced Oil and Vinegar (Sect. 3.1), Learning with Errors (Sect. 3.2) and Saber (Sect. 3.3), the answer is affirmative. While it is out of the scope of this paper to do a full cryptanalysis, we at least could point to point to parameters sets that allow flip-variants of the above schemes. Further schemes have been discussed in Sect. 3.4. Here, the situation is not that clear, so more sophisticated methods are needed to derive a flip-variant of this scheme. Specifically useful algorithms over dịɭɟ↶*Field*s have been discussed in Sect. 4. In addition, we saw ways to generalize dịɭɟ↶*Field*s, cf. Sect. 6 and comments on the Python implementation, cf. Sect. 7.

As the field is very new, it is not surprising that there are a lot of 𝔒pen P̆roblems. For the benefit of the reader, we have summarized them here, together with a pointer to the concrete formulation in the text.

uədO P̆roblem 1, Page 7: Are there further structures that fall into the dịɭɟ↶*Field* framework? If so—which properties do they have?

uədO P̆roblem 2, Page 8: Is flipUOV secure? How does it perform in comparison to regular UOV? .

uədO P̆roblem 3, Page 10: Extend the above definition and both define end implement a full FlipSaber. In addition see how NTT (Number Theoretic Transformation) can be used over dịɭɟ↶*Field*s to speed up FlipSaber?

uədO P̆roblem 4, Page 12: Can we explain the regularity in the output of the function x^{-1}? Is there secure Key Encapsulation mechanism based on a coding based encryption scheme over dịɭɟ↶*Field*s?

uədO P̆roblem 5, Page 12: Formal definition of NTRU over an dịɭɟ*Inṫs* or dịɭɟ*P̆olys*, respectively.

uədO P̆roblem 6, Page 13: Is there a flip variant of ML-KEM (Module-Lattice-Based Key-Encapsulation Mechanism, FIPTS203) or MLWE (Module Learning with Errors) in general?

uədO P̆roblem 7, Page 13: Can we explain the regularity in the output of the function x^{-1}? Can we construct secure S-boxes? If so—which ones?

uədO P̆roblem 8, Page 17: Does the additional register for Trivium actually increase security? Why are there polynomials that yield long cycle length between different values of N and D.

uədO P̆roblem 9, Page 17: Quantum-resiliant version of the Hidden Subgroup Problem over dịɭɟ↶*Field*s.

uədO Problem 10, Page 22: How does the Hensel lifting compared to the Extended Euclidean algorithm for computing the multiplicative inverse?

uədO Problem 11, Page 36: How can we characterize LFSR regarding maximal cycle length and their cryptographic usefulness?

uədO Problem 12, Page 39: Why is there no difference in the maximal cycle length of affine and high degree polynomials, can we construct polynomials of with a larger maximal cycle length and what are the cryptographic applications of NLFSR?

uədO Problem 13, Page 40: Which polynomial rings over $\mathrm{d}_{\text{iI}}\text{Ⅎ}\curvearrowleft Field$s can be generated, with similar properties to $X^N + 1$?

uədO Problem 14, Page 41: For some polynomial, find a second polynomial so their multiplication also lies in an $\mathrm{d}_{\text{iI}}\text{Ⅎ}\curvearrowleft Field$. Alternatively, construct a polynomial ring that is closed both under addition and multiplication or show that the problem is intractable.

uədO Problem 15, Page 44: How can we predict the number of free variables in `symRewriteEqsLin` or find a more efficient alternative?

uədO Problem 16, Page 56: How do we characterize a group of invertible matrices over $\mathrm{d}_{\text{iI}}\text{Ⅎ}\curvearrowleft Field$s?

uədO Problem 17, Page 57: Are extension fields over $\mathrm{d}_{\text{iI}}\text{Ⅎ}\curvearrowleft Field$s possible—or at least structures with reduced but maybe similar properties?

uədO Problem 18, Page 60: Optimal point between time and memory efficient implementation.

uədO Problem 19, Page 78: Sharp decline in the probability in solving matrix-vector equations using Gaussian elimination.

uədO Problem 20, Page 81: Sharp decline in the probability in solvable LU decomposition while the dimension of the problem seems to be irrelevant.

Acknowledgements

We want to thank Patrick Fitzpatrick (UCC) for giving an inspiring course in *Ring and Field Theory* in 2000/01 at *University College Cork (UCC), Ireland*, that inspired this paper.

Thank you to Vincent Rijmen (KU Leuven, Belgium) to point out ambiguity in Sect. 2.3. We want to thank Stefan Lucks (Bauhaus University Weimar, Germany) to point out the Hidden Subgroup Problem (Sect. 3.4.6). Following the suggestion from Lorenzo Grassi (TU Eindhoven, the Netherlands), we have added App. E (How to write $\mathrm{d}_{\text{iI}}\text{Ⅎ}\curvearrowleft Field$).

Artificial AI —specifically ChatGPT— was used to create this manuscript and the accompanying code.

References

[AG24] Joshua A. Grochow: *Multivariate Hensel Lifting Without Assuming as Many Variables as Equations.* The American Mathematical Monthly, 132(4), 350–355. https://doi.org/10.1080/00029890.2024.2440297 and https://home.cs.colorado.edu/~jgrochow/hensel-notes.pdf

[Be99] Kent Beck: *Extreme programming explained: embrace change.* Addison-Wesley, 1999, 190 pages, ISBN:978-0-201-61641-5

[BS96] E. Bach, J. Shallit: *Algorithmic Number Theory: Efficient algorithms, volume 1* The MIT Press (1996), 512 pages, ISBN 978-0262024051.

[BDK+21] A. Basso, J. P. D'Anvers, A. Karmakar, J. M. Bermudo Mera, S. S. Roy, M. Van Beirendonck, F. Vercauteren: *Saber Round 3 presentation: Third NIST Post-Quantum Cryptography Standardization Conference 2021.* 14 slides, https://www.esat.kuleuven.be/cosic/pqcrypto/saber/files/saberround3.pdf

[BDF20] Koen de Boer, Léo Ducas, and Serge Fehr: *On the Quantum Complexity of the Continuous Hidden Subgroup Problem.* In EUROCRYPT 2020, Anne Canteaut (Eds.), LNCS 12106, Springer, 2020, pp 341–370, https://doi.org/10.1007/978-3-030-45724-2_12 or https://eprint.iacr.org/2019/716

[CS+12] Nicolas T. Courtois, Pouyan Sepehrdad, Petr Susil and Serge Vaudenay: *ElimLin Algorithm Revisited.* In Fast Software Encryption - 19th International Workshop, FSE 2012, Anne Canteaut (Eds.), LNCS 7549, Springer, 2012. pp 306–325. https://doi.org/10.1007/978-3-642-34047-5_18

[DCP09] Christophe De Cannière and Bart Preneel: *Trivium*, In New Stream Cipher Designs - The eSTREAM Finalists, Matthew Robshaw and Olivier Billet (Eds.), LNCS 4986, Springer, 2008, pp. 244-266. https://link.springer.com/chapter/10.1007/978-3-540-68351-3_18

[DKRV18] Jan-Pieter D'Anvers, Angshuman Karmakar, Sujoy Sinha Roy, and Frederik Vercauteren: *Saber: Module-LWR Based Key Exchange, CPA-Secure Encryption and CCA-Secure KEM* In: A. Joux, A. Nitaj, T. Rachidi (eds). AFRICACRYPT 2018. LNCS 10831. Springer, pp 282–305 https://doi.org/10.1007/978-3-319-89339-6_16 See also https://eprint.iacr.org/2018/230 and https://www.esat.kuleuven.be/cosic/pqcrypto/saber/

[FIPS203] FIPS 203: *Module-Lattice-Based Key-Encapsulation Mechanism Standard*, National Institute of Standards and Technology, 56 pages, version 2024-08-13, https://csrc.nist.gov/pubs/fips/203/final

[FPC+25] S. Fluhrer, S. Prorock, M. Celi, J. Gray, K. Xagawa, and H. Kosuge: *NTRU Key Encapsulation.* Version 2025-03-03, https://www.ietf.org/archive/id/draft-fluhrer-cfrg-ntru-02.html

[GAA+25] Philippe Gaborit, Carlos Aguilar-Melchor, Nicolas Aragon *et al.*: *Hamming Quasi-Cyclic (HQC).* 51 pages, 22 Aug 2025, https://pqc-hqc.org/doc/hqc_specifications_2025_08_22.pdf

[HPS98] Jeffrey Hoffstein, Jill Pipher, Joseph H. Silverman: *NTRU: A Ring-Based Public Key Cryptosystem,* 22 pages. https://www.ntru.org/f/hps98.pdf

[KPG99] Aviad Kipnis, Jacques Patarin, and Louis Goubin: *Unbalanced Oil and Vinegar signature schemes.* In Advances in Cryptology—EUROCRYPT 1999, LNCS 1592, pp. 206—222. Jacques Stern, editor, Springer, 1999. https://www.iacr.org/cryptodb/data/paper.php?pubkey=2487.

[KS02] Alexander Klimov and Adi Shamir: *A New Class of Invertible Mappings* In CHES 2002—Cryptographic Hardware and Embedded Systems. LNCS 2523. Springer, pp. 470–483. https://doi.org/10.1007/3-540-36400-5_34

[KS04a] Masao Kasahara and Ryuichi Sakai: *A construction of public key cryptosystem for realizing ciphtertext of size 100 bit and digital signature scheme.* IEICE Trans. Fundamentals, E87-A(1):102—109, January 2004.

[KS04b] Masao Kasahara and Ryuichi Sakai: *A construction of public-key cryptosystem based on singular simultaneous equations.* In Symposium on Cryptography and Information Security — SCIS 2004. The Institute of Electronics, Information and Communication Engineers, January 27–30 2004. 6 pages.

[LN02] Rudolf Lidl and Harald Niederreiter: *Introduction to Finite Fields and their Applications—Revised Edition.* xii+416 pages, Cambride University Press, 2002.

[MOV18] Alfred J. Menezes, Paul C. van Oorschot and Scott A. Vanstone: *Handbook of Applied Cryptography.* xxiii+780 pages, CRC Press, 2018, ISBN 978-1-138-38597-9. https://cacr.uwaterloo.ca/hac/

[N-PQC] NIST: Post-Quantum Cryptography Standardization https://csrc.nist.gov/pqc-standardization

[NTRU] Cong Chen, Oussama Danba, Jerey Hostein, Andreas Hülsing, Joost Rijneveld, John M. Schanck, Tsunekazu Saito, Peter Schwabe, William Whyte, Keita Xagawa, Takashi Yamakawa, Zhenfei Zhang: *NTRU: Algorithm Speci cations And Supporting Documentation.* Version 2020-09-30 https://csrc.nist.gov/CSRC/media/Projects/post-quantum-cryptography/documents/round-3/submissions/NTRU-Round3.zip

[Sab25] *SABER: LWR-based KEM.* https://www.esat.kuleuven.be/cosic/pqcrypto/saber/. Version 2025-09-06.

[SML24] Ardianto Satriawan, Rella Mareta, and Hanho Lee: *A Complete Beginner Guide to the Number Theoretic Transform (NTT).* 21 pages. https://eprint.iacr.org/2024/585 See also https://ieeexplore.ieee.org/document/10177902

[Re05] Oded Regev: *On lattices, learning with errors, random linear codes, and cryptography.* In: STOC 05. pp. 84-93. ACM (2005), https://doi.acm.org/10.1145/1060590.1060603

[Po35] Karl Popper: *The Logic of Scientific Discovery.* 1959—German version: *Logik der Forschung. Zur Erkenntnistheorie der modernen Naturwissenschaft.* 1935, 513 pages.

[QW15] Frank Quedenfeld and Christopher Wolf: *Advanced Algebraic Attack on Trivium*, Sixth International Conference on Mathematical Aspects of Computer and Information Sciences (MACIS) 2015, 17 pages, accepted. *or* Cryptology ePrint Archive, Paper 2014/893, url = https://eprint.iacr.org/2014/893

[Sh94] Peter W. Shor: *Algorithms for quantum computation: Discrete logarithms and factoring.* Proceedings of the 35th Annual Symposium on Foundations of Computer Science, pp. 124—134. https://doi.org/10.1109/sfcs.1994.365700.1994.365700. ISBN 978-0-8186-6580-6.

[UOV25] Ward Beullens, Ming-Shing Chen, Jintai Ding, Boru Gong, Matthias J. Kannwischer, Jacques Patarin, Bo-Yuan Peng, Dieter Schmidt, Cheng-Jhih Shih, Chengdong Tao, Bo-Yin Yang: *UOV: Unbalanced Oil and Vinegar. Algorithm Specifications and Supporting Documentation. Version 2.0* from 5 February 2025. https://csrc.nist.gov/csrc/media/Projects/pqc-dig-sig/documents/round-2/spec-files/uov-spec-round2-web.pdf

[WGR24] Violetta Weger, Niklas Gassner and Joachim Rosenthal: *A Survey on Code-Based Cryptography.* 175 pages, 18th July 2024. https://arxiv.org/abs/2201.07119

[Wi24] Eyal Wirsansky: Hands-On Genetic Algorithms with Python, Packt Publishing, 2024, 418 pages. ISBN 978-1805123798

[Wo05] Christopher Wolf: *Multivariate Quadratic Polynomials in Public Key Cryptography.* Ph.D.-Thesis, KU Leuven, 2005. 156+xxiv pages. https://eprint.iacr.org/2005/393

A Matrix solving—Depreciated Methods

A.1 Generalized LU Decomposition

After unsuccessfully using a heuristic approach in the previous section, we now follow a more rigorous strategy. Let $s \in \mathbb{N}$ be the dimension of the matrix space. First, we observe that $MM^{-1} = I$ cannot be defined over the integer carrier $\tilde{\mathbb{N}}^s$ as it contains $s^2 - s$ zero coefficients, *i.e.* elements that are not defined over $\tilde{\mathbb{N}}$. We hence generalize this definition to $yMM^{-1} = y \; \forall y \in \tilde{\mathbb{N}}^s$ or equivalently $BMM^{-1} = B \; \forall B \in \tilde{\mathbb{N}}^{s \times t}$ for any $t \in \mathbb{N}, t \equiv 1 \pmod 2$. If either of these relationships hold, we call M invertibe and M^{-1} the inverse of M. Consequently, we can solve $Mx = b$ via $x = M^{-1}b$. Note that M is also defined over the integer ring, so results like the uniqueness of M^{-1} carry over, as for any dilFInts.

We now consider $PM = LU$ for some permutation P and two matrices $L, U \in \mathbb{N}^{s \times s}$. *W.l.o.g.*, we assume P to be the identity permutation, *i.e.* we have $M = LU$. In addition, generalize the two matrices L, U in a way that they can be defined over the integer carrier $\tilde{\mathbb{N}}$. For illustration, we start with the case $s = 5$ and then generalize to arbitrary s. Let

$$L := \begin{pmatrix} 1 & & & & \\ l_{1,0} & l_{1,1} & 1 & & \\ l_{2,0} & l_{2,1} & 1 & & \\ l_{3,0} & l_{3,1} & l_{3,2} & l_{3,3} & 1 \\ l_{4,0} & l_{4,1} & l_{4,2} & l_{4,3} & 1 \end{pmatrix} \text{ and } U := \begin{pmatrix} u_{0,0} & u_{0,1} & u_{0,2} & u_{0,3} & u_{0,4} \\ u_{1,0} & u_{1,1} & u_{1,2} & u_{1,3} & u_{1,4} \\ & & u_{2,2} & u_{2,3} & u_{2,4} \\ & & u_{3,2} & u_{3,3} & u_{3,4} \\ & & & & u_{4,4} \end{pmatrix}$$

for $l_{i,j}, u_{i.j} \in \tilde{\mathbb{N}}$. We say that these matrices L, U have a step-size of 2. Note that for a step-size of 1, the above definition matches with the canonical LU-decomposition. The 5 rows are grouped into 3 layers, namely $\{1\}, \{2, 3\}$ and $\{4, 5\}$.

Solving $Ly = b$. At first glance, L, U are not defined over the integer carrier $\tilde{\mathbb{N}}$. However, when inspecting the two matrices closer, we see that each row can be expressed via a tally operator. For given vectors $x, y \in \tilde{\mathbb{N}}^s$, the operations Ly and Ux are closed over $\tilde{\mathbb{N}}$. More precisely, some vector $y : Ly = b$ for some given vector $b \in \tilde{\mathbb{N}}^s$ and $x : Ux = y$.

$$L := \begin{pmatrix} 1 & & & & \\ l_{1,0} & l_{1,1} & 1 & & \\ l_{2,0} & l_{2,1} & 1 & & \\ l_{3,0} & l_{3,1} & l_{3,2} & l_{3,3} & 1 \\ l_{4,0} & l_{4,1} & l_{4,2} & l_{4,3} & 1 \end{pmatrix} \cdot \begin{pmatrix} y_0 \\ y_1 \\ y_2 \\ y_3 \\ y_4 \end{pmatrix} = \begin{pmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \\ b_4 \end{pmatrix}$$

for $y_i, b_i \in \tilde{\mathbb{N}}$ and a given vector b. Inspecting the above equation we see that $y_0 = b_0$. In addition, we have

$$\begin{cases} b_1 = [y_0 l_{1,0}, y_1 l_{1,1}, y_2] \\ b_2 = [y_0 l_{2,0}, y_1 l_{2,1}, y_2] \end{cases}$$

Guessing either y_1 or y_2 yields a fully determined system of equations over the integer carrier $\tilde{\mathbb{N}}$, guaranteeing closure of the tally operator. Last but not least, we have

$$\begin{cases} b_3 = [y_0 l_{3,0}, y_1 l_{3,1}, y_2 l_{3,2}, y_3 l_{3,3}, y_4]_5 \\ b_4 = [y_0 l_{4,0}, y_1 l_{4,1}, y_2 l_{4,2}, y_3 l_{4,3}, y_4]_5 \end{cases}$$

As before, guessing y_3 or y_4 yields a solution. Note that this approach is very similar to the Stepwise-Triangular Scheme (STS) as outlined in [Wo05, Sect. 3.1.2] and first proposed under

the names RSE(2)PKC [KS04a] and RSSE(2)PKC [KS04b], respectively. In addition we know from the work on STS that this construction does not yield one unique solution. However, in a signature scheme, we do not need *the* solution but only *one* solution and can discard the rest. We therefore know that the computational cost of solving each layer is bounded $O(\lfloor s/2 \rfloor Ds^2)$: for each layer, we have to exhaustively search one variable y_i and then compute two tally operators of size $O(s)$ each.

While the case $s = 1$ is trivial, the above constructions carries through for any other odd $s \in \mathbb{N}$ via induction.

Solving $Ux = y$. Using some y from the previous step, we now aim at solving the equation

$$U := \begin{pmatrix} u_{0,0} & u_{0,1} & u_{0,2} & u_{0,3} & u_{0,4} \\ u_{1,0} & u_{1,1} & u_{1,2} & u_{1,3} & u_{1,4} \\ & & u_{2,2} & u_{2,3} & u_{2,4} \\ & & u_{3,2} & u_{3,3} & u_{3,4} \\ & & & & u_{4,4} \end{pmatrix} \cdot \begin{pmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \\ x_4 \end{pmatrix} = \begin{pmatrix} y_0 \\ y_1 \\ y_2 \\ y_3 \\ y_4 \end{pmatrix}$$

Our strategy is similar as above—except we start with the last row and use

$$x_4 = u_{4,4}^{-1} y_4 \,.$$

Using this solution, and after guessing either x_2 or x_3, we solve the following system of 2 equations for the middle layer of U:

$$\begin{cases} y_2 = [u_{2,2}x_2, u_{2,3}x_3, u_{2,4}x_4] \\ y_3 = [u_{3,2}x_2, u_{3,3}x_3, u_{3,4}x_4] \end{cases}$$

Guessing either x_2 or x_3 yields a system of equations in one variable only. By the closure property of the tally operator, we have $x_2, x_3 \in \tilde{\mathbb{N}}$.
Finally, we consider Layer 0:

$$\begin{cases} y_0 = [u_{0,0}x_0, u_{0,1}x_1, u_{0,2}x_2, u_{0,3}x_3, u_{0,4}x_4]_5 \\ y_1 = [u_{1,0}x_0, u_{1,1}x_1, u_{012}x_2, u_{1,3}x_3, u_{1,4}x_4]_5 \end{cases}$$

Not surprisingly, fixing x_0 or x_1 gives two equations in only one unknown. We have $x_0, x_1 \in \tilde{\mathbb{N}}$. As for L, the algorithm generalizes for any odd $s \in \mathbb{N}$ via induction.

(Un)solvability of $Ly = b$. At first glance, the above construction looks sounds. Alas, there is a subtle problem at the column-level. *W.l.o.g.* consider Layer 1 in the lower triangular matrix L, *i.e.*

$$\begin{cases} b_1 = [y_0 l_{1,0}, y_1 l_{1,1}, y_2] \ (I) \\ b_2 = [y_0 l_{2,0}, y_1 l_{2,1}, y_2] \ (II) \end{cases}$$

Computing $(I) - (II)$ and solving for y_1 over the integers we obtain

$$y_1 = \frac{(b_1 - b_2) + y_0(l_{2,0} - l_{1,0})}{l_{1,1} - l_{2,1}} \,.$$

Denote $B := b_1 - b_2, \Lambda := l_{2,0} - l_{1,0}$ and the denominator $\Delta := l_{1,1} - l_{2,1}$. Obviously, all $B, \Lambda, \Delta \in \mathbb{N}$ are even numbers. Even worse, there is no multiplicative inverse for Δ over $\tilde{\mathbb{N}}^d$. Extracting all prime factors 2 from B, Λ, Δ we can write $B = 2^b\beta, \Lambda = 2^l\lambda$ and $\Delta = 2^d\delta$ with

$2 \nmid \beta, 2 \nmid \lambda$ and $2 \nmid \delta$. Iff $B + \lambda \pmod{D}$ is divisible by 2^l, there is a multiplicative inverse over the integer carrier $\tilde{\mathbb{N}}^d$. However, we can make this bound even tighter: y_1 is odd iff the denominator has exactly the same number of prime powers of 2 in its factorizations than the numerator. We clearly see this in the experiments given in App. B.2. Only for small values of s, we obtain solutions. So from a cryptographic point of view, this construction is not viable for signature schemes as we do not achive relevant solutions for relevant parameters.

Summary. Unfortunately, there is no way to have both an odd number of terms in the row and in the column at the same time. For example, consider $s = 5$ and a split into layers as $\{1, 3, 1\}$. While is is fine for Layer 0, there is a problem for Layer 1 as we have 4 variables in each column here. Hence, the generalized LU decomposition outlined in this subsection cannot be used for our purposes. However, we show in the following section a possible way to overcome this problem.

A.2 Bespoke LU Decomposition

While the previous section showed us the limits of generalized LU decomposition, it also holds the key for a solution. We revisit Layer 2 of the lower triangular matrix and hence the equations

$$\begin{cases} b_3 = [y_0 l_{3,0}, y_1 l_{3,1}, y_2 l_{3,2}, y_3 l_{3,3}, y_4]_5 \ (I) \\ b_4 = [y_0 l_{4,0}, y_1 l_{4,1}, y_2 l_{4,2}, y_3 l_{4,3}, y_4]_5 \ (II) \end{cases}$$

Computing $(II) - (I)$ and solving for y_3 over the integers we obtain

$$y_3 = \frac{(b_4 - b_3) + y_0(l_{4,0} - l_{3,0}) + y_1(l_{4,1} - l_{3,1}) + y_2(l_{4,2} - l_{3,2})}{l_{4,3} - l_{3,3}} .$$

As we subtract two odd numbers, *e.g.* $l_{4,0} - l_{3,0}$, we cannot make the equation yielding an odd value for y_3, but we can *nearly* ensure it.

Definition 9. *To this aim we consider a* ***near-odd*** *numbers $a \in \mathbb{N}$ for some odd $\alpha \in \tilde{\mathbb{N}}^d$ such that $a = 2\alpha$. Or in other words: there is only one 2 in the factorization of the near-odd number a.*

Denote the numbers $\beta := b_4 - b_3, \lambda_i := l_{4,i} - l_{3,i}$ for $1 \leq i \leq 3$. This allows to rewrite the above equation as

$$y_3 = \frac{\beta + y_0\lambda_0 + y_1\lambda_1 + y_2\lambda_2}{\lambda_3} .$$

As the partial solutions y_0, y_1, y_2 come from layers 0 and 1, they are all odd. Fixing $\beta, \lambda_0, \ldots, \lambda_3$ to near-odd numbers yields y_3 odd. This can be done by adjusting the coefficients in the lower triangular matrix L accordingly, as for the input vector b. By the closure of $\mathrm{d}_{\mathrm{i}}\mathrm{l}_{\mathrm{F}}\mathit{Int}\dot{s}$, this also yields an odd solution for y_4. *W.l.o.g.* this constructions carries through to all layers of the lower triangular matrix L. For some layer $i : 1 < i < s/2$ we have

$$y_{2i-1} := \frac{(b_{2i} - b_{2i-1}) + \sum_{k=0}^{2i-2} y_k(l_{2i,k} - l_{2i-1,k})}{l_{2i,2i-1} - l_{2i-1,2i-1}} . \tag{1}$$

As for Layer 2, we can compute y_{2i} using either of the two row equations in Layer i.

Similarly, we can use this idea for the upper triangular matrix U. To practically use this idea, we need to generate the two matrices L, U in such a way that the difference of neighbouring coefficients always yields a near-odd number, see equation (1) for the corresponding pairs.

Bespoke LU has been implemented in the functions `symSolveBeLUlower,symSolveBeLUupper`. Alas, it did not yield any meaninful results. The reason for this is unknown.

Discussion From a cryptanalytic point of view, it seems very helpful to have a correlation between the lower bits of the coefficients of the two matrices L, U already know. However—both matrices are large in comparison, and all other items are usually chosen at random. So in practice, the impact should be limited. In particular, the attacker usually does not have access to both matrices, but to their product LU. However, cryptographic schemes have already been broken for a smaller biases, so this property should be the be object of uttermost scrutiny from the cryptanalytic community.

A.3 Empirics on Solving Affine Equations $Ax = b$

As we saw particularly in Sect. 4.2, we cannot use Gaussian elimination to solve matrix equations for higher dimensions of $s \in \mathbb{N}$. Even worse, we cannot compute the inverse matrix A^{-1} directly for some given matrix $A \in \tilde{\mathbb{N}}^{n \times n}$. However, what about small values of s? In particular, we will look at $s \in \{3, 5\}$. We start with plotting an overview of pairs (s, d). The rows in the following table have have the following meaning. All values were computet with the Python method `solveMatVec`.

dim s is the dimension of the corresponding equation, so matrix $M \in \tilde{\mathbb{N}}^{s \times s}$ and vector $b \in \tilde{\mathbb{N}}^{s}$.

d **is the dimension.** We have $D = 2^{d+1}$ for the modulo of the carrier $\tilde{\mathbb{N}}$.

#rep is the number of equations tested for this set of parameters.

#cases is the number of cases yielding a solution, no solution do to an algorithm failure or the result contained at least one even element *(res fail)*. The sum of the cases of one line is equal to #rep.

ratio is the ratio between the three cases solution, alg fail and res fail. In every line, the values for ratio add up to 1—except for rounding errors.

solution Number of cases that yielded a solution $x \in \tilde{\mathbb{N}}^{s}$ for the given equation $Mx = b$. As the correctness of the solution was not verified, the actual number of solutions can be lower.

alg fail The algorithm aborted the computation due to an assertion error, in most cases, the matrix M was not invertible.

res fail Although the algorithm returned a result, it did not fit the previous equation, *e.g.* it conained even coefficients and was hence no solution over $\tilde{\mathbb{N}}$.

dim			solution		alg fail		res fail	
s	d	#rep	#cases	ratio	#cases	ratio	#cases	ratio
3	1	103400	9676	0.09358	64755	0.62626	28969	0.28016
	2	103365	13167	0.12738	50119	0.48487	40079	0.38774
	3	104972	14219	0.13546	48135	0.45855	42618	0.40599
	4	107573	14721	0.13685	48910	0.45467	43942	0.40849
	6	104922	14234	0.13566	47834	0.45590	42854	0.40844
	7	107856	14684	0.13614	49068	0.45494	44104	0.40892
	8	105156	14214	0.13517	48093	0.45735	42849	0.40748
	12	100810	13776	0.13665	45663	0.45296	41371	0.41039
	15	103675	14200	0.13697	47309	0.45632	42166	0.40671
	16	106869	14801	0.13850	48817	0.45679	43251	0.40471

dim			solution		alg fail		res fail	
s	d	#rep	#cases	ratio	#cases	ratio	#cases	ratio
	24	104913	14324	0.13653	47682	0.45449	42907	0.40898
	31	105758	14500	0.13711	48110	0.45491	43148	0.40799
	32	105379	14386	0.13652	47868	0.45425	43125	0.40924
	48	104103	14120	0.13563	47613	0.45736	42370	0.40700
	63	102778	14020	0.13641	46889	0.45622	41869	0.40737
	64	104705	14433	0.13784	47520	0.45385	42752	0.40831
	96	104481	14316	0.13702	47660	0.45616	42505	0.40682
	127	104867	14116	0.13461	48124	0.45891	42627	0.40649
	128	105072	14164	0.13480	48098	0.45776	42810	0.40743
	192	102503	13835	0.13497	46997	0.45849	41671	0.40653
	255	104698	14177	0.13541	47873	0.45725	42648	0.40734
	256	104882	14083	0.13427	48093	0.45854	42706	0.40718
	384	104021	13966	0.13426	47334	0.45504	42721	0.41070
	511	103089	13897	0.13481	46985	0.45577	42207	0.40942
	512	103053	13970	0.13556	46877	0.45488	42206	0.40956
	768	107408	14567	0.13562	49093	0.45707	43748	0.40731
	1023	102826	13973	0.13589	46757	0.45472	42096	0.40939
	1024	101465	13724	0.13526	46479	0.45808	41262	0.40666
	1536	104965	14283	0.13607	48031	0.45759	42651	0.40634
	2047	105590	14221	0.13468	48167	0.45617	43202	0.40915
	2048	104952	14419	0.13739	47796	0.45541	42737	0.40721
5	1	104086	2060	0.01979	70051	0.67301	31975	0.30720
	2	104073	3003	0.02885	56648	0.54431	44422	0.42684
	3	105402	3004	0.02850	56828	0.53915	45570	0.43234
	4	101597	2989	0.02942	54581	0.53723	44027	0.43335
	6	106627	3065	0.02875	57280	0.53720	46282	0.43406
	7	103557	3037	0.02933	55600	0.53690	44920	0.43377
	8	104205	2998	0.02877	56158	0.53892	45049	0.43231
	12	103502	3080	0.02976	55886	0.53995	44536	0.43029
	15	104663	2994	0.02861	56406	0.53893	45263	0.43246
	16	104132	2948	0.02831	56007	0.53785	45177	0.43384
	24	103752	2978	0.02870	56000	0.53975	44774	0.43155
	31	105284	3002	0.02851	56432	0.53600	45850	0.43549
	32	105943	2984	0.02817	56902	0.53710	46057	0.43473
	48	104124	2969	0.02851	55987	0.53770	45168	0.43379
	63	104572	3030	0.02898	55795	0.53356	45747	0.43747
	64	104165	3002	0.02882	55989	0.53750	45174	0.43368
	96	103893	2966	0.02855	56000	0.53902	44927	0.43244
	127	102302	2954	0.02888	54808	0.53575	44540	0.43538
	128	101732	2964	0.02914	54725	0.53793	44043	0.43293
	192	105097	3071	0.02922	56319	0.53588	45707	0.43490
	255	104150	3053	0.02931	55904	0.53676	45193	0.43392
	256	102835	3007	0.02924	55165	0.53644	44663	0.43432
	384	105412	3036	0.02880	56992	0.54066	45384	0.43054
	511	101925	2976	0.02920	54786	0.53751	44163	0.43329
	512	102421	2950	0.02880	55272	0.53965	44199	0.43154
	768	105769	3050	0.02884	56970	0.53863	45749	0.43254

Table 22: Number of solutions x for given dimensions s and d for a matrix $A \in (\tilde{\mathbb{N}}^d)^{s\times s}$ and the two vectors $x, b \in \left(\tilde{\mathbb{N}}^d\right)^s$ for 10,000 tries each.

n	d	Number of Solutions vs. Number of Occurrences													
3	1	0	7484	4	2516										
	2	0	8273	4	975	8	712	16	40						
	3	0	8519	4	900	8	373	16	191	32	17				
	4	0	8567	4	919	8	357	16	107	32	46	64	3	128	1
5	1	0	9360					16	640						
	2	0	9636					16	173	32	175	64	16		
		0	9658					16	188	32	95	64	55	128	4
	3	0	9674					16	184	32	86	64	49	128	7

dim			solution		alg fail		res fail	
s	d	#rep	#cases	ratio	#cases	ratio	#cases	ratio
	1023	100280	2968	0.02960	53814	0.53664	43498	0.43377
	1024	109388	3161	0.02890	58875	0.53822	47352	0.43288
	1536	105775	3149	0.02977	56769	0.53670	45857	0.43353
	2047	103998	2953	0.02839	55831	0.53685	45214	0.43476
	2048	107194	3037	0.02833	57717	0.53843	46440	0.43323

Excluding $d = 1$, we see that the number of solutions is more or less indepentend from the dimension $s \in \mathbb{N}$.

Before going deeper into the subject, we investigate how many solutions we can expect for an equation $Ax = b$ for a given matrix A and a vector $b \in \tilde{\mathbb{N}}^n$ using brute force. Obviously, this limits us to smaller dimensions s and smaller values of $D = 2^{d+1}$, respectively. So we use brute-force on the vector $x \in \tilde{\mathbb{N}}^n$ to make sure that we really catch all possible solutions. We have varied the parameters $s, D \in \mathbb{N}$ and run 10,000 experiments each using the Python function `printNumSolMat`. The results can be found in Tab. 22. For example, this means that for $d = 1, s = 3$, we have a total of 2516 cases that have 4 solutions while for $s = 3$ and $d = 2$, we have 8,519 times no solution but 975 times 4 solutions.

To get an idea why this happens, we look at the example

$$A = \begin{pmatrix} 1 & 1 & 3 \\ 1 & 3 & 1 \\ 1 & 5 & 5 \end{pmatrix}, b = \begin{pmatrix} 5 \\ 5 \\ 3 \end{pmatrix} \text{ and } x \in \left\{ \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix}, \begin{pmatrix} 5 \\ 5 \\ 1 \end{pmatrix}, \begin{pmatrix} 5 \\ 1 \\ 5 \end{pmatrix}, \begin{pmatrix} 1 \\ 5 \\ 5 \end{pmatrix} \right\}$$

for $d = 2$ and consequently $D = 8$. All four values of x solve the equation $Ax = b$. The interesting bit is that these four values do *not* form a vector space, but are derived from distributing the number 4=8/2 on all possible $\binom{3}{2} = 3$ places on the initial vector $(1, 1, 1)^T$, *i.e.*we have $5 = 1 + 4$ twice as $8 = 4 + 4$. Hence, this preserves the solution for different input vectors. As this is an inevitable feature of dịḷƎ*Inṫs*, we cannot expect unique solutions but always a multitude of possible solutions. The problem becomes worse with a higher number of dimensions—both for the matrix dimension s and the dịḷƎ*Inṫs* dimension d.

We start with an intermediate definition of *linear dependency of vectors*, which we capture as follows:

Definition 10. *Consider three vectors $u, v, w \in \tilde{\mathbb{N}}^s$ for some $s \in \mathbb{N}$. If there exist scalars $\alpha, \beta \in \tilde{\mathbb{N}}$ s/t $[u, \alpha v, \beta w] = u$, we say v, w are* ***linearly dependent regarding*** *u.*

Corollary 11. *Consider the vectorspace $\tilde{\mathbb{N}}^s$. If for some vector u, two vectors v, w are* ***linearly dependent regarding*** *u, they are also for some other vector $u' \in \tilde{\mathbb{N}}^s$.*

Proof. We know that there exists two numbers $\alpha, \beta \in \tilde{\mathbb{N}}$ with $u = [u, \alpha v, \beta w]$. W.l.o.g., assume $u'_0 = u_0 + a_0$ for some even number $a_0 \in \mathbb{N}$. In this case we can write

$$u = (u_0, \ldots, u_{s-1}) = [(u_0, \ldots, u_{s-1}), \alpha(v_0, \ldots, v_{s-1}), \beta(w_0, \ldots, w_{s-1})] \text{ and}$$

$$u' = (u_0 + a_0, \ldots, u_{s-1}) = [(u_0 + a_0, \ldots, u_{s-1}), \alpha(v_0, \ldots, v_{s-1}), \beta(w_0, \ldots, w_{s-1})] .$$

Adding some number $a_i \in \tilde{\mathbb{N}}$ to each component of the vector $u \in \tilde{\mathbb{N}}^s$ completes the proof. □

Hence we can relax Def. 10 to capture linear dependency in the vector space $\tilde{\mathbb{N}}^s$. We have

Definition 11. *Consider three vectors $u, v, w \in \tilde{\mathbb{N}}^s$ for some $s \in \mathbb{N}$. If there exist scalars $\alpha, \beta \in \tilde{\mathbb{N}}$ s/t $[u, \alpha v, \beta w] = u$, we say v, w are* ***linearly dependent****.*

This definition generalizes naturally for any k vectors $v^i \in \left(\tilde{\mathbb{N}}^s\right)^k$ for some odd number $k \in \mathbb{N}$.

Definition 12. *Consider a list of vectors $(v_0, \ldots, v_{s-1})$ with $v_i \in \tilde{\mathbb{N}}^s$ and $0 \leq i < k$ for some odd number $k \in \mathbb{N}$. If $v_1, \ldots, v_{k-1}$ are not linearly dependent with respect to some other vector $w \in \tilde{\mathbb{N}}^s$, we say that $(v_0, \ldots, v_{s-1})$ are* linearly independent*.*

As there is not identity matrix, we cannot define invertible matrices. However, we can capture the idea of invertibility to solvable via the following definition.

Definition 13. *For some odd number $s \in \mathbb{N}$, consider the matrix $A \in \tilde{\mathbb{N}}^{s \times s}$. We say that matrix A is* ***ideally solvable*** *iff $\forall b \in \tilde{\mathbb{N}}^s \ \exists x \in \tilde{\mathbb{N}}^s : Ax = b$.*

In practice, this notion from Def. 13 is not usable as we cannot test all vectors b for a given matrix A. We therefore retreat to a computation notion.

Definition 14. *Consider some matrix $A \in \tilde{\mathbb{N}}^{s \times s}$ and $b_i \in \tilde{\mathbb{N}}^s$ random vectors for $0 \leq i < k$, we say A is k-**computationally solvable** iff for each b_i we can compute some vector $x_i \in \tilde{\mathbb{N}}^s$ s/t $Ax_i = b_i$. If there is no such pair (b_0, x_0), we call the matrix A **0-computationally solvable***

If the corresponding number k is clear from the context, we will shorten this notion to *computationally solvable.*

Alas, we see that k-computationally solvability does not help us from a cryptographic point of view either, as it is clearly not associated with invertibility of matrices, see Tab. 23 for empirical details. While there is a slight advantage of being $(k+1)$-computationally solvable if a matrix is already k-computationally solvable, the effect is not as strong as we would need it to use some matrix A to be useful instead of an invertible matrix, say B^{-1}.

We therefore flip the definition from matrices A to vectors b. In particular, we ask the question, how many equations $Ax = b$ have a solution x for varying b.

Definition 15. *Consider some matrix $A \in \tilde{\mathbb{N}}^{s \times s}$ and a list of k vectors $\ell := \{b_0, \ldots, b_{k-1}\}$ for some number $k \in \mathbb{N}$. Then we call the matrix A r-**responsive** regarding the list ℓ, if there is a subset σ of size r with $\forall b \in \sigma \ \exists x \in \tilde{\mathbb{N}}^s : Ax = b$.*

Table 23: Empirical results on k-computationally solvability for some odd dimension $s \in \mathbb{N}$, some matrices $A \in \tilde{\mathbb{N}}^{s \times s}$ plotting the number of such matrices. Each line represents 10,000 trials.

s	d	k-computationally solvable vs. Number of matrices A.													
3	2	0	8720	1	979	2	234	3	49	4	16	5	2		
	3	0	8686	1	1044	2	206	3	49	4	14	5	1		
	7	0	8682	1	1056	2	202	3	41	4	15	5	3	7	1
	10	0	8620	1	1070	2	249	3	51	4	8	5	2		
5	2	0	9700	1	285	2	15								
	3	0	9710	1	271	2	18	3	1						
	7	0	9725	1	262	2	11	3	2						
	10	0	9728	1	262	2	9	3	1						
7	2	0	9939	1	60	2	1								
	3	0	9931	1	66	2	3								
	7	0	9929	1	69	2	2								
	10	0	9936	1	64										
9	2	0	9985	1	15										
	3	0	9982	1	18										
	7	0	9979	1	21										
	10	0	9982	1	17	2	1								
11	2	0	9994	1	6										
	3	0	9998	1	2										
	7	0	9998	1	2										
	10	0	9995	1	5										
45	2	0	10000												
	3	0	10000												
	7	0	10000												
	10	0	10000												

To use this method in practice, we see from Tab. 23 that there is a correlation between matrices that are already *1-computationally solvable* to matrices that are $(k+1)$-computationally solvable for $k \leq 1$, cf. App. B.3. Hence, we first use *1-computationally solvability* as a filter. After that, we determine which k-responsiveness they get. Note that we might also obtain 0-responsiveness this way. We observe from App. B.3 that k-responsiveness heavily depents on the dimension s. Within one dimension s, the different outputs between the $\mathrm{d}\dot{\mathrm{i}}\mathrm{l}\mathrm{F}\mathit{Int}\dot{s}$-dimension d are negligible.

Using the notion from above, we have somputed extensive data on the k-responsiveness of random matrices A. The corresponding data is given in App. B.3 and also captured in Fig. 1. As the vectors b_i are chosen independently and lead to a success/fail-event, the definition of responsiveness implies a Poisson stochastical model for our data. Hence for $d = 5$, we actually use Poisson. In particular, we use the function

$$P_{\lambda,\sigma}(x, \mu) := \sigma \frac{\mu^{x+\lambda}}{(x+\lambda)!} e^{\mu}$$

from the Python method `scipy.stats.poisson`. The parameters are

Slice x of the input.

Expected value μ for the Poisson distribution.

Relocation λ shifts the input n.

Stretch σ adapts the output to the number of cases.

To see how well our statistical model (S) captures the observed data (O), we use

$$\chi^2(O, S) := \frac{(o_i - s_i)^2}{|s_i|}$$

In particular, for the data given in App B.3 we obtain

d	μ	λ	σ	χ^2
5	9.00	-4	2940	33.1881
7	6.82	-9	2900	39.1797

using a Poisson model.

Fitting for matrix dimension $s = 3$ proved way more difficult - as we will see with a quick glance at Fig. 2. Obviously, we are facing a bimodal fitting problem here. All in all, parameters are as above, except the probability mass function now splits into two parts, namely

$$P^2_{\lambda_1,\sigma_1,\lambda_2,\sigma_2}(x, \mu_1, \mu_2) := P(\lambda_1, \sigma_1)(x, \mu_1) + P(\lambda_2, \sigma_2)(x, \mu_2)\,.$$

For the current data, we used the followings parameters for the fitting.

d	μ_1	λ_1	σ_1	μ_2	λ_2	σ_2	χ^2
3	6.97	5	2700	18.20	7	9850	49,830.7608

In all three cases $s = 3, 5, 7$, an alternative modelling is the Binomial modelling. Details including figures are given in App B.3. While it is a better fit for the matrix dimension $s = 7$, the Poisson modelling proves superior for the other cases. In addition, the overall amount of data for fitting is way smaller for matrix dimension $s = 7$. Hence, we work on the assumption that all data can be fit best with a Poisson distribution and will therefore use this for the reminder of this section.

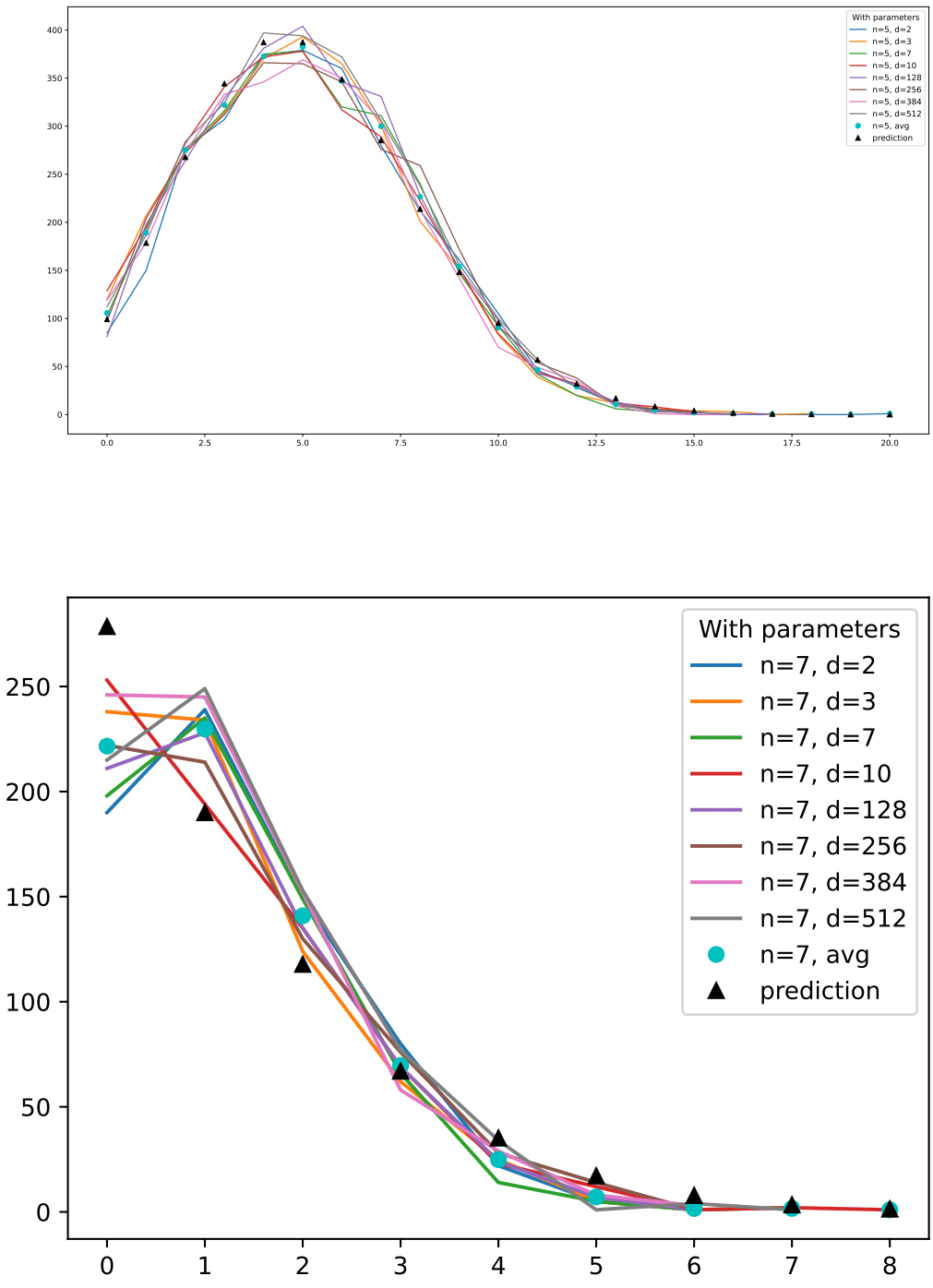

Figure 1: k-responsiviness for matrix dimensions (a) s=3, (b) s=5, (c) and s=7 for different dimension d.

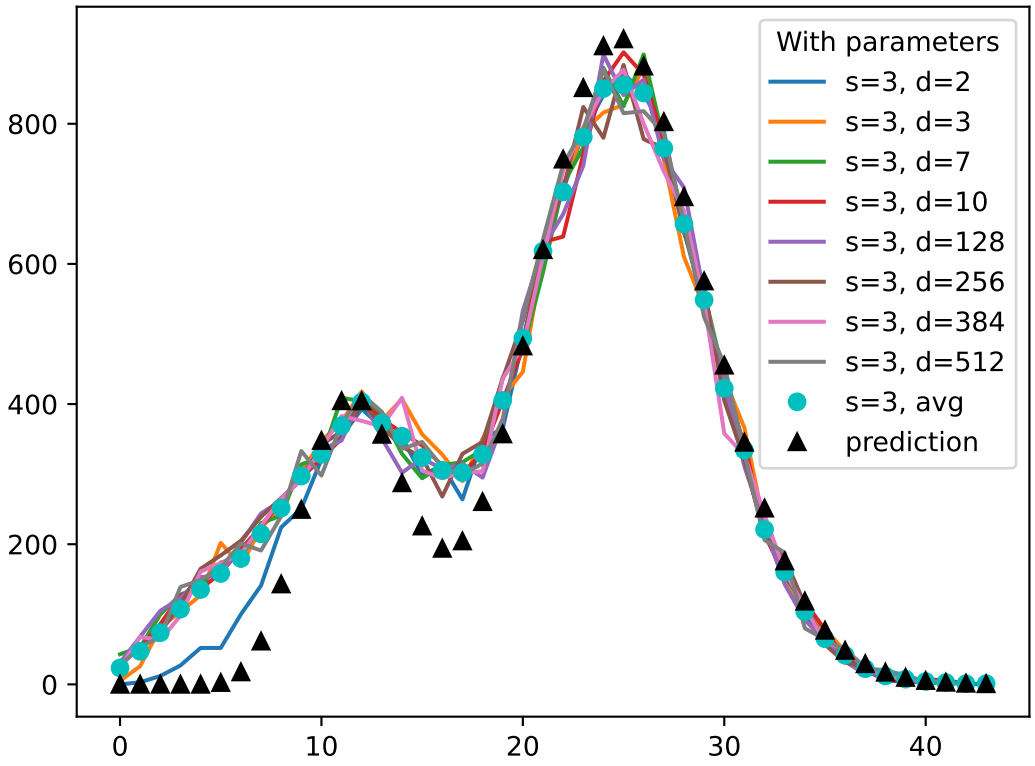

Figure 2: k-responsiviness for matrix dimensions s=3, (b) and and different $\mathrm{d_{IJ}}\dot{Int}s$ dimensions d. The prediction uses a bimodal Poisson model.

Discussion. We have seen that the Poisson distribution can be used to fit our data. For matrix dimension $s = 5, 7$, we can use unimodal models. This is not the case for matrix dimension $s = 3$, which yields bimodal data. The most likely explanation is that we have two different types of matrices with different behaviour for k-responsiveness.

B Matrix Operations—Empirical Findings

B.1 Gaussian Elimination Heuristic

Using a Python implementation of the heuristic algorithm from Sect. 4.2, we have tested the usefulness of this heuristic for cryptographic purposes. It is implemented in the two Python functions `rndMatTest` and `solveMatVec`, see also Sect. 7.2. The table below is the result of $\approx$ 45h of computation on am AMD Athlon Silver 3050U with 2.3 GHz and two cores, representing approx. 2,500,000 trials to solve randomly generated equations $Mx = b$ for $M \in \tilde{\mathbb{N}}^{s\times s}, b \in \tilde{\mathbb{N}}^s$. The Python implementation of the algorithm is given in `rndMatTest`. The rows have the following meaning:

dim s is the dimension of the corresponding equation, so matrix $M \in fInt^{s\times s}$ and vector $b \in \tilde{\mathbb{N}}^s$.

D **is the modulus used.** We have $D = 2^{d+1}$ for the dimension of the carrier $\tilde{\mathbb{N}}$.

#rep is the number of equations tested for this set of parameters.

#cases is the number of cases yielding a solution, no solution do to an algorithm failure or the result contained at least one even element *(res fail)*. The sum of the cases of one line is equal to #rep.

ratio is the ratio between the three cases solution, alg fail and res fail. In every line, the values for ratio add up to 1—except for rounding errors.

solution Number of cases that yielded a solution $x \in \tilde{\mathbb{N}}^s$ for the given equation $Mx = b$. As the correctness of the solution was not verified, the actual number of solutions can be lower.

alg fail The algorithm aborted the computation due to an assertion error, in most cases, the matrix M was not invertible.

res fail Although the algorithm returned a result, it did not fit the previous equation, *e.g.* it conained even coefficients and was hence no solution over $\tilde{\mathbb{N}}$.

We note that there is a sharp decline in the ratio of possible solutions for $s > 10$ and hence well below the cryptographically relevant dimensions. For example, for UOV, the corresponding parameters suggested in Sect. 3.1 is $s = 44$. Interestingly, the dimension d does not show a significant effect in the test set. The reason for these two questions is stated as **uədO Ďroblem 19**.

dim			solution		alg fail		res fail	
s	D	#rep	#cases	ratio	#cases	ratio	#cases	ratio
3	16	26595	3599	0.13533	12292	0.46219	10704	0.40248
	128	26657	3637	0.13644	12235	0.45898	10785	0.40458
	256	26056	3516	0.13494	12049	0.46243	10491	0.40263
	512	26264	3581	0.13635	11853	0.45130	10830	0.41235
	65536	26032	3541	0.13602	11785	0.45271	10706	0.41126
5	16	25896	704	0.02719	14028	0.54171	11164	0.43111
	128	26302	736	0.02798	14257	0.54205	11309	0.42997
	256	25858	710	0.02746	13916	0.53817	11232	0.43437
	512	26479	746	0.02817	14333	0.54130	11400	0.43053
	65536	26535	777	0.02928	14149	0.53322	11609	0.43750
7	16	26385	166	0.00629	14774	0.55994	11445	0.43377
	128	26079	189	0.00725	14585	0.55926	11305	0.43349
	256	25985	164	0.00631	14513	0.55851	11308	0.43517
	512	26078	188	0.00721	14539	0.55752	11351	0.43527
	65536	25909	192	0.00741	14409	0.55614	11308	0.43645
9	16	25986	35	0.00135	14464	0.55661	11487	0.44205
	128	26027	39	0.00150	14538	0.55857	11450	0.43993
	256	26536	40	0.00151	14869	0.56033	11627	0.43816
	512	26292	44	0.00167	14791	0.56257	11457	0.43576
	65536	26647	44	0.00165	15058	0.56509	11545	0.43326
11	16	26275	12	0.00046	14791	0.56293	11472	0.43661
	128	26562	8	0.00030	15022	0.56554	11532	0.43415
	256	26365	15	0.00057	14737	0.55896	11613	0.44047
	512	26023	14	0.00054	14642	0.56266	11367	0.43681

dim			solution		alg fail		res fail	
s	D	#rep	#cases	ratio	#cases	ratio	#cases	ratio
	65536	26727	12	0.00045	15124	0.56587	11591	0.43368
13	16	26305	8	0.00030	15029	0.57134	11268	0.42836
	128	25818	3	0.00012	14567	0.56422	11248	0.43567
	256	26277	1	0.00004	14868	0.56582	11408	0.43414
	512	26414	4	0.00015	14833	0.56156	11577	0.43829
	65536	26299	3	0.00011	14815	0.56333	11481	0.43656
17	16	26634	0	0.00000	14904	0.55959	11730	0.44041
	128	26523	0	0.00000	14966	0.56426	11557	0.43574
	256	25914	0	0.00000	14650	0.56533	11264	0.43467
	512	25739	0	0.00000	14599	0.56719	11140	0.43281
	65536	26558	0	0.00000	14835	0.55859	11723	0.44141
25	16	26170	0	0.00000	14739	0.56320	11431	0.43680
	128	25861	0	0.00000	14556	0.56286	11305	0.43714
	256	26512	0	0.00000	15006	0.56601	11506	0.43399
	512	26712	0	0.00000	15137	0.56667	11575	0.43333
	65536	25941	0	0.00000	14663	0.56524	11278	0.43476
37	16	26083	0	0.00000	14714	0.56412	11369	0.43588
	128	26083	0	0.00000	14836	0.56880	11247	0.43120
	256	26028	0	0.00000	14574	0.55994	11454	0.44006
	512	26094	0	0.00000	14768	0.56595	11326	0.43405
	65536	26333	0	0.00000	14896	0.56568	11437	0.43432
41	16	26545	0	0.00000	15119	0.56956	11426	0.43044
	128	25651	0	0.00000	14476	0.56434	11175	0.43566
	256	25920	0	0.00000	14594	0.56304	11326	0.43696
	512	26375	0	0.00000	14913	0.56542	11462	0.43458
	65536	26389	0	0.00000	14673	0.55603	11716	0.44397
43	16	26347	0	0.00000	14941	0.56709	11406	0.43291
	128	26173	0	0.00000	14720	0.56241	11453	0.43759
	256	26304	0	0.00000	14778	0.56182	11526	0.43818
	512	26162	0	0.00000	14755	0.56399	11407	0.43601
	65536	26159	0	0.00000	14725	0.56290	11434	0.43710
45	16	26144	0	0.00000	14772	0.56502	11372	0.43498
	128	26301	0	0.00000	14768	0.56150	11533	0.43850
	256	26404	0	0.00000	15011	0.56851	11393	0.43149
	512	25682	0	0.00000	14446	0.56250	11236	0.43750
	65536	26532	0	0.00000	14848	0.55963	11684	0.44037
47	16	26193	0	0.00000	14743	0.56286	11450	0.43714
	128	26545	0	0.00000	15080	0.56809	11465	0.43191
	256	26456	0	0.00000	14983	0.56634	11473	0.43366
	512	26226	0	0.00000	14773	0.56330	11453	0.43670
	65536	26283	0	0.00000	14849	0.56497	11434	0.43503
57	16	26211	0	0.00000	14808	0.56495	11403	0.43505
	128	26900	0	0.00000	15049	0.55944	11851	0.44056
	256	26803	0	0.00000	15110	0.56374	11693	0.43626
	512	26662	0	0.00000	15017	0.56324	11645	0.43676
	65536	26071	0	0.00000	14728	0.56492	11343	0.43508
69	16	25959	0	0.00000	14597	0.56231	11362	0.43769

dim			solution		alg fail		res fail	
s	D	#rep	#cases	ratio	#cases	ratio	#cases	ratio
	128	26633	0	0.00000	15131	0.56813	11502	0.43187
	256	26315	0	0.00000	14866	0.56492	11449	0.43508
	512	26716	0	0.00000	14986	0.56094	11730	0.43906
	65536	25775	0	0.00000	14591	0.56609	11184	0.43391
81	16	26918	0	0.00000	15148	0.56275	11770	0.43725
	128	25613	0	0.00000	14423	0.56311	11190	0.43689
	256	26224	0	0.00000	14786	0.56383	11438	0.43617
	512	26056	0	0.00000	14798	0.56793	11258	0.43207
	65536	25630	0	0.00000	14404	0.56200	11226	0.43800
93	16	26116	0	0.00000	14866	0.56923	11250	0.43077
	128	26066	0	0.00000	14736	0.56533	11330	0.43467
	256	26376	0	0.00000	14813	0.56161	11563	0.43839
	512	26301	0	0.00000	14926	0.56751	11375	0.43249
	65536	26728	0	0.00000	15103	0.56506	11625	0.43494
101	16	26791	0	0.00000	15144	0.56526	11647	0.43474
	128	26328	0	0.00000	14883	0.56529	11445	0.43471
	256	25972	0	0.00000	14660	0.56445	11312	0.43555
	512	27055	0	0.00000	15283	0.56489	11772	0.43511
	65536	25723	0	0.00000	14579	0.56677	11144	0.43323
113	16	25636	0	0.00000	14602	0.56959	11034	0.43041
	128	26065	0	0.00000	14683	0.56332	11382	0.43668
	256	26004	0	0.00000	14624	0.56238	11380	0.43762
	512	26198	0	0.00000	14739	0.56260	11459	0.43740
	65536	25986	0	0.00000	14764	0.56815	11222	0.43185

B.2 Generatlized LU Decomposition

To see if generalized LU decomposition from App. B.2 yields results, approx. 160,000 experiments running for approx. 13h were conducted, see the table below for the results. The Python implementation of the algorithm is given in `rndTestLU`. The rows in this table have the following meaning:

dim s is the dimension of the corresponding matrix equation, so matrix $M \in fInt^{s\times s}$ and vector $b \in \tilde{\mathbb{N}}^s$.

D **is the modulus used.** We have $D = 2^{d+1}$ for the dimension of the carrier $\tilde{\mathbb{N}}$.

#rep is the number of equations tested.

#cases is the number of cases yielding a solution, no solution do to an algorithm failure or the result contained at least one even element *(res fail)*. The sum of the cases of one line is equal to #rep.

ratio is the ratio between the three cases solution, alg fail and res fail. In every line, the values for ratio add up to 1—except for rounding errors.

solution Cases that yielded a solution $x \in \tilde{\mathbb{N}}^s$ for the given equation $LUx = b$. The correctness of the solution was not verified.

#sol The number of solutions to the original equations. The correctness of the solution was verified.

res fail Although the algorithm returned a result, it did not fit the previous equation.

As we see, the number of solutions rapidly degenerated with dimension s. Hence, using LU decomposition as outlined in App. B.2 does not work in a cryptographic setting, as the number of valid solutions is way too small to be of practical relevance. **uədO Problem 20** asks the question why there is a sharp decline of possible solutions between $s = 9$ and $s = 11$

dim			solution		#sol		res fail	
s	D	#rep	#cases	ratio	#cases	ratio	#cases	ratio
3	16	1724	271	0.15719	174	0.10093	1534	0.88979
	128	1760	191	0.10852	28	0.01591	1688	0.95909
	256	1843	209	0.11340	6	0.00326	1734	0.94086
	512	1688	191	0.11315	6	0.00355	1442	0.85427
	65536	1675	195	0.11642	0	0.00000	1796	1.07224
5	16	1770	43	0.02429	0	0.00000	1552	0.87684
	128	1727	23	0.01332	0	0.00000	848	0.49102
	256	1754	17	0.00969	0	0.00000	1568	0.89396
	512	1743	28	0.01606	0	0.00000	3728	2.13884
	65536	1825	25	0.01370	0	0.00000	2112	1.15726
7	16	1663	10	0.00601	0	0.00000	1408	0.84666
	128	1621	3	0.00185	0	0.00000	1344	0.82912
	256	1779	2	0.00112	0	0.00000	5120	2.87802
	512	1640	1	0.00061	0	0.00000	256	0.15610
	65536	1716	3	0.00175	0	0.00000	4224	2.46154
9	16	1640	2	0.00122	0	0.00000	1024	0.62439
	128	1612	0	0.00000	0	0.00000	0	0.00000
	256	1816	1	0.00055	0	0.00000	16384	9.02203
	512	1731	1	0.00058	0	0.00000	16384	9.46505
	65536	1699	0	0.00000	0	0.00000	0	0.00000
11	16	1615	0	0.00000	0	0.00000	0	0.00000
	128	1655	0	0.00000	0	0.00000	0	0.00000
	256	1765	0	0.00000	0	0.00000	0	0.00000
	512	1604	0	0.00000	0	0.00000	0	0.00000
	65536	1818	0	0.00000	0	0.00000	0	0.00000
13	16	1664	0	0.00000	0	0.00000	0	0.00000
	128	1606	0	0.00000	0	0.00000	0	0.00000
	256	1812	0	0.00000	0	0.00000	0	0.00000
	512	1693	0	0.00000	0	0.00000	0	0.00000
	65536	1767	0	0.00000	0	0.00000	0	0.00000
17	16	1766	0	0.00000	0	0.00000	0	0.00000
	128	1680	0	0.00000	0	0.00000	0	0.00000
	256	1717	0	0.00000	0	0.00000	0	0.00000
	512	1741	0	0.00000	0	0.00000	0	0.00000
	65536	1563	0	0.00000	0	0.00000	0	0.00000
25	16	1710	0	0.00000	0	0.00000	0	0.00000
	128	1815	0	0.00000	0	0.00000	0	0.00000
	256	1814	0	0.00000	0	0.00000	0	0.00000

dim			solution		#sol		res fail	
s	D	#rep	#cases	ratio	#cases	ratio	#cases	ratio
	512	1710	0	0.00000	0	0.00000	0	0.00000
	65536	1611	0	0.00000	0	0.00000	0	0.00000
37	16	1772	0	0.00000	0	0.00000	0	0.00000
	128	1783	0	0.00000	0	0.00000	0	0.00000
	256	1620	0	0.00000	0	0.00000	0	0.00000
	512	1824	0	0.00000	0	0.00000	0	0.00000
	65536	1752	0	0.00000	0	0.00000	0	0.00000
41	16	1668	0	0.00000	0	0.00000	0	0.00000
	128	1563	0	0.00000	0	0.00000	0	0.00000
	256	1710	0	0.00000	0	0.00000	0	0.00000
	512	1652	0	0.00000	0	0.00000	0	0.00000
	65536	1813	0	0.00000	0	0.00000	0	0.00000
43	16	1604	0	0.00000	0	0.00000	0	0.00000
	128	1687	0	0.00000	0	0.00000	0	0.00000
	256	1748	0	0.00000	0	0.00000	0	0.00000
	512	1742	0	0.00000	0	0.00000	0	0.00000
	65536	1676	0	0.00000	0	0.00000	0	0.00000
45	16	1592	0	0.00000	0	0.00000	0	0.00000
	128	1655	0	0.00000	0	0.00000	0	0.00000
	256	1615	0	0.00000	0	0.00000	0	0.00000
	512	1871	0	0.00000	0	0.00000	0	0.00000
	65536	1809	0	0.00000	0	0.00000	0	0.00000
47	16	1645	0	0.00000	0	0.00000	0	0.00000
	128	1586	0	0.00000	0	0.00000	0	0.00000
	256	1587	0	0.00000	0	0.00000	0	0.00000
	512	1717	0	0.00000	0	0.00000	0	0.00000
	65536	1874	0	0.00000	0	0.00000	0	0.00000
57	16	1648	0	0.00000	0	0.00000	0	0.00000
	128	1748	0	0.00000	0	0.00000	0	0.00000
	256	1694	0	0.00000	0	0.00000	0	0.00000
	512	1578	0	0.00000	0	0.00000	0	0.00000
	65536	1805	0	0.00000	0	0.00000	0	0.00000
69	16	1725	0	0.00000	0	0.00000	0	0.00000
	128	1735	0	0.00000	0	0.00000	0	0.00000
	256	1822	0	0.00000	0	0.00000	0	0.00000
	512	1669	0	0.00000	0	0.00000	0	0.00000
	65536	1546	0	0.00000	0	0.00000	0	0.00000
81	16	1720	0	0.00000	0	0.00000	0	0.00000
	128	1728	0	0.00000	0	0.00000	0	0.00000
	256	1673	0	0.00000	0	0.00000	0	0.00000
	512	1696	0	0.00000	0	0.00000	0	0.00000
	65536	1778	0	0.00000	0	0.00000	0	0.00000
93	16	1737	0	0.00000	0	0.00000	0	0.00000
	128	1549	0	0.00000	0	0.00000	0	0.00000
	256	1730	0	0.00000	0	0.00000	0	0.00000
	512	1678	0	0.00000	0	0.00000	0	0.00000

dim			solution		#sol		res fail	
s	D	#rep	#cases	ratio	#cases	ratio	#cases	ratio
	65536	1774	0	0.00000	0	0.00000	0	0.00000
101	16	1595	0	0.00000	0	0.00000	0	0.00000
	128	1612	0	0.00000	0	0.00000	0	0.00000
	256	1780	0	0.00000	0	0.00000	0	0.00000
	512	1730	0	0.00000	0	0.00000	0	0.00000
	65536	1615	0	0.00000	0	0.00000	0	0.00000
113	16	1808	0	0.00000	0	0.00000	0	0.00000
	128	1533	0	0.00000	0	0.00000	0	0.00000
	256	1730	0	0.00000	0	0.00000	0	0.00000
	512	1585	0	0.00000	0	0.00000	0	0.00000
	65536	1698	0	0.00000	0	0.00000	0	0.00000

B.3 General Solving of Affine Equations $Ax = b$

In this section, we consider k-responsiveness for a random matrix $A \in \tilde{\mathbb{N}}^{s \times s}$ and random vestors $b \in \tilde{\mathbb{N}}^s$ for a list of 100 vectors each. The total number of matrices tried was 100,000 for each pair $(s, d) \in \mathbb{N} \times \mathbb{N}$ and s odd. #noSol gives the number of matrices that are not 1-computationally solvable and are hence excluded from further tests. See App. A.3 for details on the definitions.

s	d	#noSol	r-reponsvieness vs. Number of vectors b for a given matrix A															
3	2	87217	1	3	2	12	3	27	4	52	5	52	6	100	7	141	8	224
			9	251	10	321	11	354	12	393	13	367	14	352	15	299	16	312
			17	264	18	350	19	365	20	491	21	619	22	706	23	787	24	841
			25	872	26	843	27	790	28	647	29	540	30	443	31	335	32	230
			33	151	34	103	35	59	36	50	37	20	38	6	39	3	40	4
			41	3	43	1												
	3	86418	0	5	1	26	2	81	3	104	4	127	5	202	6	169	7	223
			8	254	9	303	10	347	11	375	12	418	13	377	14	407	15	357
			16	328	17	290	18	350	19	410	20	446	21	611	22	711	23	789
			24	816	25	827	26	880	27	749	28	610	29	538	30	437	31	366
			32	225	33	145	34	102	35	79	36	47	37	22	38	15	39	10
			40	1	41	3												
	7	86314	0	43	1	53	2	100	3	128	4	140	5	174	6	181	7	230
			8	241	9	314	10	324	11	409	12	404	13	385	14	329	15	294
			16	313	17	317	18	335	19	438	20	478	21	588	22	714	23	766
			24	880	25	825	26	899	27	774	28	645	29	559	30	438	31	340
			32	215	33	158	34	111	35	62	36	35	37	26	38	10	39	7
			40	3	41	1												
	10	86379	0	33	1	52	2	86	3	127	4	136	5	159	6	193	7	225
			8	261	9	295	10	322	11	351	12	404	13	379	14	359	15	341
			16	312	17	301	18	331	19	405	20	479	21	631	22	639	23	756
			24	861	25	902	26	870	27	774	28	664	29	564	30	412	31	346
			32	214	33	155	34	118	35	78	36	37	37	24	38	5	39	6
			40	7	41	6	42	1										
	128	86472	0	31	1	68	2	105	3	123	4	153	5	155	6	202	7	244
			8	262	9	298	10	335	11	348	12	411	13	351	14	302	15	323

s	d	#noSol	r-reponsvieness vs. Number of vectors b for a given matrix A															
			16	307	17	315	18	295	19	373	20	533	21	624	22	671	23	740
			24	899	25	841	26	863	27	750	28	709	29	556	30	430	31	311
			32	225	33	143	34	92	35	62	36	33	37	14	38	17	39	6
			40	7	41	1												
	256	86406	0	26	1	55	2	75	3	113	4	165	5	184	6	205	7	239
			8	266	9	292	10	342	11	361	12	404	13	368	14	339	15	324
			16	268	17	329	18	346	19	436	20	519	21	627	22	706	23	824
			24	780	25	884	26	778	27	765	28	646	29	562	30	405	31	320
			32	212	33	180	34	114	35	57	36	31	37	25	38	12	39	7
			40	3														
	384	86473	0	23	1	68	2	63	3	98	4	161	5	173	6	186	7	226
			8	265	9	292	10	337	11	383	12	376	13	368	14	409	15	305
			16	294	17	302	18	304	19	436	20	482	21	611	22	733	23	795
			24	847	25	877	26	800	27	732	28	676	29	546	30	358	31	327
			32	242	33	161	34	110	35	60	36	50	37	25	38	11	39	8
			40	4	41	2	42	1										
3	512	86364	0	27	1	53	2	68	3	139	4	149	5	167	6	201	7	191
			8	240	9	333	10	298	11	372	12	414	13	389	14	336	15	346
			16	310	17	300	18	315	19	379	20	523	21	635	22	742	23	792
			24	878	25	815	26	818	27	786	28	659	29	524	30	457	31	329
			32	206	33	188	34	80	35	64	36	49	37	23	38	20	39	13
			40	4	41	2	42	2										
5	2	97215	0	85	1	150	2	277	3	307	4	374	5	379	6	360	7	280
			8	212	9	161	10	105	11	46	12	29	13	12	14	4	15	3
			20	1														
	3	97138	0	120	1	207	2	272	3	314	4	370	5	393	6	365	7	301
			8	201	9	151	10	83	11	39	12	20	13	12	14	6	15	4
			16	3	18	1												
	7	97180	0	103	1	192	2	275	3	316	4	375	5	378	6	320	7	311
			8	240	9	147	10	92	11	42	12	20	13	6	14	3		
	10	97140	0	129	1	195	2	282	3	341	4	372	5	378	6	317	7	289
			8	222	9	152	10	84	11	44	12	32	13	12	14	8	15	3
	128	97096	0	81	1	198	2	264	3	329	4	381	5	404	6	346	7	331
			8	228	9	151	10	98	11	43	12	31	13	13	14	5	17	1
	256	97129	0	97	1	204	2	273	3	312	4	366	5	365	6	346	7	276
			8	259	9	172	10	92	11	54	12	38	13	9	14	6	15	2
	384	97206	0	119	1	179	2	273	3	333	4	346	5	369	6	350	7	305
			8	213	9	142	10	70	11	49	12	35	13	9	14	1	16	1
	512	97028	0	112	1	187	2	284	3	324	4	397	5	394	6	372	7	306
			8	238	9	156	10	100	11	57	12	28	13	11	14	3	15	2
			16	1														
7	2	99314	0	190	1	239	2	149	3	80	4	22	5	5	6	1		
	3	99311	0	238	1	234	2	124	3	62	4	25	5	5	6	1		
	7	99330	0	198	1	235	2	149	3	66	4	14	5	5	6	1	7	2
	10	99310	0	253	1	194	2	135	3	69	4	23	5	12	6	1	7	2
			8	1														
	128	99325	0	211	1	228	2	135	3	69	4	24	5	7	6	1		

s	d	#noSol	r-reponsvieness vs. Number of vectors b for a given matrix A															
	256	99315	0	222	1	214	2	130	3	76	4	28	5	14	6	1		
	384	99259	0	246	1	245	2	152	3	58	4	29	5	8	6	3		
	512	99266	0	215	1	249	2	153	3	77	4	34	5	1	6	4	7	1

As we saw in App. A.3, the Poisson modelling is the first choice for the data above. However, there is an alternative, namely using a Binomial distribution to explain the data. To this purpose, we have to fit the data with the probability mass function

$$P_{\lambda,\sigma}(k,n,p,n_g) := \sigma\binom{n+\lambda}{k}p^k(p-1)^{n+\lambda-k}$$

and use the following prediction parameters:

Guessed variables n is the number of elements in the sequence.

Successful trials k and coincides with k-responsiveness the corresponding matrix A.

Probability p for k-respoonsiveness for a generally 1-responsive matrix for a given input k.

Shift λ changes the input n.

Stretch σ on the output to accomodate the data.

Using a genetic algorithm, we have fitted the input for s=5, 7 receiving the following parameters, see `statGArun` and `statPaint` for the code:

s	n_g	p	λ	σ	χ^2
5	23	0.5073	-7	2330	4057.7814
7	23	0.2822	-7	1460	18.0793

See Fig. 3 for a graphical representation of both the average of s and different values of d and $P(g,n,p,\lambda,\sigma)$.

Fitting for matrix dimension $s = 3$ proved way more difficult - as we will see with a quick glance at Fig. 4. Obviously, we are facing a bimodal fitting problem here and therefore generalize the probability mass function to

$$P_{\lambda_1,\sigma_1,\lambda_2,\sigma_2}(k,n,p_1,n_{g1},p_2,n_{g2}) := P_{\lambda_1,\sigma_1}(k,n,p_1,n_{g1}) + P_{\lambda_2,\sigma_2}(k,n,p_2,n_{g2})\,.$$

We fit the above data with

s	n_{g1}	p_1	λ_1	σ_1	n_{g2}	p_2	λ_2	σ_2	χ^2
3	37.0	0.6795	0	7000	50	0.2697	0	3500	$85.495 \cdot 10^{12}$

The extremely high value of χ^2 indicates that the Binomial distribution is not a good predictor for our data. This is in line with our theory.

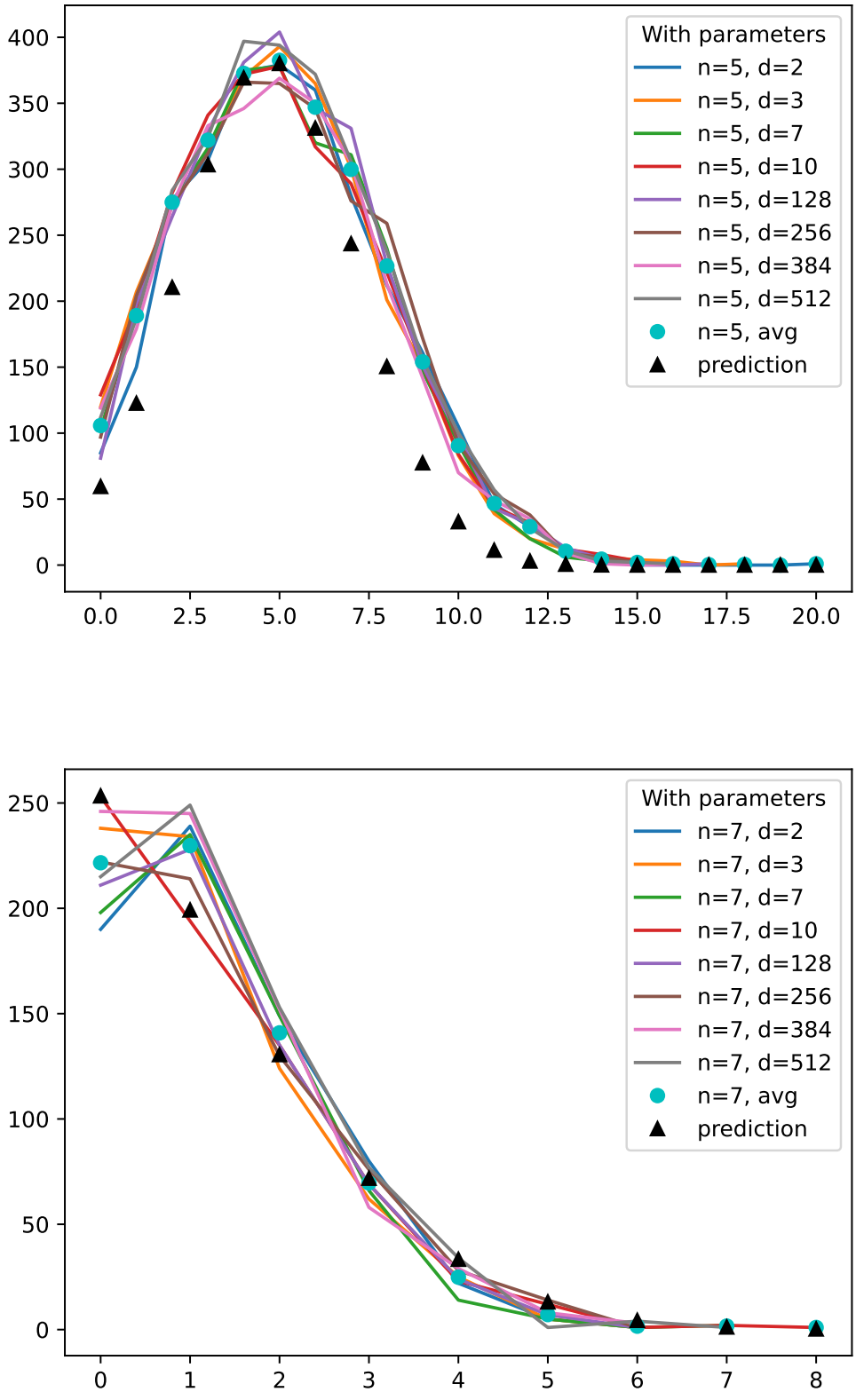

Figure 3: k-responsiviness for matrix dimensions (a) s=5, (b) and s=7 for different dimensions d. The prediction uses a binomial model.

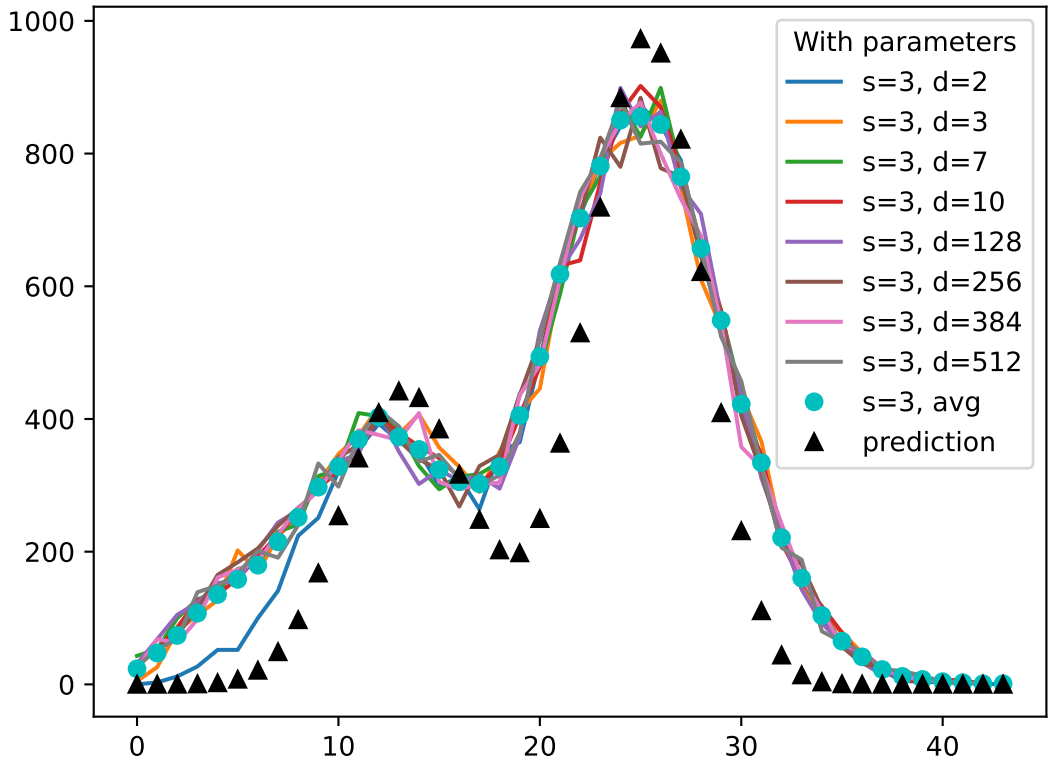

Figure 4: k-responsiviness for matrix dimensions s=3, (b) and and different $\mathrm{d}\underset{\cdot}{\mathrm{I}}\underset{\cdot}{\mathrm{I}}\mathit{Int}\dot{s}$ dimensions d. The prediction uses a binomial model.

C Implementing Linear Feedback Shift Registers (LFSR)

Following the outline in Sect. 4.6, Linear Feedback Shift Registers (LFSR) over $\mathrm{d}\underset{\cdot}{\mathrm{I}}\underset{\cdot}{\mathrm{I}}\mathit{Int}\dot{s}$ were implemented, see the Python functions `printFCL` and `lfsr_fcl_findCycleLength` in particular.

Main aim was to find polynomials $R(X)$ of degree $N \in \mathbb{N}$ that correspond to a particular LFSR with maximal cycle length. Starting point was always the all-one state polynomial $A(X) = \sum_{i=0}^{N-1} X^i$ for some $N \in \mathbb{N}$ and all coefficients equal to 1. First criterion was maximal cycle length, the second criterion was minimal number of terms in $R(S)X$ to reach this cycle length. In all cases, the best 10 polynomials $R(X)$ are reported in the following table for given degree N and cycle length, see the Python function `printFCL` for more details.

First, we list the corresponding results for $\mathrm{d}\underset{\cdot}{\mathrm{I}}\underset{\cdot}{\mathrm{I}}\mathit{Int}\dot{s}$ in App.C.1, then the corresponding results for $\mathrm{d}\underset{\cdot}{\mathrm{I}}\underset{\cdot}{\mathrm{I}}\breve{P}\mathit{olys}$ in App.C.2.

C.1 LFSR—Maximal cycle length over $\mathrm{d}\underset{\cdot}{\mathrm{I}}\underset{\cdot}{\mathrm{I}}\mathit{Int}\dot{s}$

The following results have been computed in about 1,25h. After around half a day, the computation was aborted, so there are no results for $N = 8, D = 16$.

N	D	cycle	$R(X)$
2	4	4	X^2+3
		2	X^2+3X
		1	X^2+1
		1	X^2+X
2	8	4	X^2+3
		4	X^2+5
		4	X^2+7
		2	X^2+3X
		2	X^2+5X
		2	X^2+7X
		1	X^2+1
		1	X^2+X
2	16	8	X^2+3
		8	X^2+5
		8	X^2+11
		8	X^2+13
		4	X^2+7
		4	X^2+9
		4	X^2+15
		4	X^2+3X
		4	X^2+5X
		4	X^2+11X
3	4	6	X^3+3
		4	X^3+3X
		4	X^3+X^2+X+1
		4	X^3+X^2+3X+3
		4	X^3+3X^2+X+3
		4	X^3+3X^2+3X+1
		2	X^3+3X^2
		1	X^3+1
		1	X^3+X
		1	X^3+X^2
3	8	8	X^3+X^2+X+1
		8	X^3+X^2+X+5
		8	X^3+X^2+3X+3
		8	X^3+X^2+3X+7
		8	X^3+X^2+5X+1
		8	X^3+X^2+5X+5
		8	X^3+X^2+7X+3

N	D	cycle	$R(X)$
		8	X^3+X^2+7X+7
		8	X^3+3X^2+X+3
		8	X^3+3X^2+X+7
3	16	16	X^3+X^2+X+1
		16	X^3+X^2+X+5
		16	X^3+X^2+X+9
		16	X^3+X^2+X+13
		16	X^3+X^2+3X+3
		16	X^3+X^2+3X+7
		16	$X^3+X^2+3X+11$
		16	$X^3+X^2+3X+15$
		16	X^3+X^2+5X+1
		16	X^3+X^2+5X+5
4	4	14	X^4+X^2+X+1
		14	X^4+X^2+3X+3
		14	X^4+3X^2+X+3
		14	X^4+3X^2+3X+1
		14	$X^4+X^3+X^2+1$
		14	$X^4+X^3+3X^2+3$
		14	$X^4+3X^3+X^2+3$
		14	$X^4+3X^3+3X^2+1$
		12	X^4+X^3+X+1
		12	X^4+X^3+3X+3
4	8	14	X^4+X^2+X+1
		14	X^4+X^2+X+3
		14	X^4+X^2+X+5
		14	X^4+X^2+3X+1
		14	X^4+X^2+3X+3
		14	X^4+X^2+3X+7
		14	X^4+X^2+5X+1
		14	X^4+X^2+5X+5
		14	X^4+X^2+5X+7
		14	X^4+X^2+7X+3
4	16	28	X^4+X^2+X+1
		28	X^4+X^2+X+3
		28	X^4+X^2+X+5
		28	X^4+X^2+X+9
		28	X^4+X^2+X+11
		28	X^4+X^2+X+13

N	D	cycle	$R(X)$
		28	X^4+X^2+3X+1
		28	X^4+X^2+3X+3
		28	X^4+X^2+3X+7
		28	X^4+X^2+3X+9
5	4	30	X^5+X^3+X+1
		30	X^5+X^3+3X+3
		30	X^5+3X^3+X+3
		30	X^5+3X^3+3X+1
		30	$X^5+X^4+X^2+1$
		30	$X^5+X^4+3X^2+3$
		30	$X^5+3X^4+X^2+3$
		30	$X^5+3X^4+3X^2+1$
		28	X^5+X^2+X+1
		28	X^5+X^2+3X+3
5	8	30	X^5+X^3+X+1
		30	X^5+X^3+X+3
		30	X^5+X^3+X+5
		30	X^5+X^3+3X+1
		30	X^5+X^3+3X+3
		30	X^5+X^3+3X+7
		30	X^5+X^3+5X+1
		30	X^5+X^3+5X+5
		30	X^5+X^3+5X+7
		30	X^5+X^3+7X+3
5	16	60	X^5+X^3+X+1
		60	X^5+X^3+X+3
		60	X^5+X^3+X+5
		60	X^5+X^3+X+9
		60	X^5+X^3+X+11
		60	X^5+X^3+X+13
		60	X^5+X^3+3X+1
		60	X^5+X^3+3X+3
		60	X^5+X^3+3X+7
		60	X^5+X^3+3X+9
6	4	62	X^6+X^2+X+1
		62	X^6+X^2+3X+3
		62	X^6+3X^2+X+3
		62	X^6+3X^2+3X+1
		62	$X^6+X^3+X^2+1$

		Flip Inits	
N	D	cycle	$R(X)$
		62	$X^6+X^3+3X^2+3$
		62	$X^6+3X^3+X^2+3$
		62	$X^6+3X^3+3X^2+1$
		62	$X^6+X^4+X^3+1$
		62	$X^6+X^4+3X^3+3$
6	8	62	X^6+X^2+X+1
		62	X^6+X^2+X+3
		62	X^6+X^2+X+5
		62	X^6+X^2+3X+1
		62	X^6+X^2+3X+3
		62	X^6+X^2+3X+7
		62	X^6+X^2+5X+1
		62	X^6+X^2+5X+5
		62	X^6+X^2+5X+7
		62	X^6+X^2+7X+3
6	16	124	X^6+X^2+X+1
		124	X^6+X^2+X+3
		124	X^6+X^2+X+5
		124	X^6+X^2+X+9
		124	X^6+X^2+X+11
		124	X^6+X^2+X+13
		124	X^6+X^2+3X+1
		124	X^6+X^2+3X+3
		124	X^6+X^2+3X+7
		124	X^6+X^2+3X+9
7	4	126	$X^7+X^4+X^2+1$
		126	$X^7+X^4+3X^2+3$
		126	$X^7+3X^4+X^2+3$
		126	$X^7+3X^4+3X^2+1$
		126	X^7+X^5+X+1
		126	X^7+X^5+3X+3
		126	$X^7+X^5+X^3+1$
		126	$X^7+X^5+3X^3+3$
		126	$X^7+X^5+X^4+X^2+X+3$
		126	$X^7+X^5+X^4+X^2+3X+1$
7	8	126	$X^7+X^4+X^2+1$
		126	$X^7+X^4+X^2+3$
		126	$X^7+X^4+X^2+5$
		126	$X^7+X^4+3X^2+1$

N	D	cycle	$R(X)$
		126	$X^7+X^4+3X^2+3$
		126	$X^7+X^4+3X^2+7$
		126	$X^7+X^4+5X^2+1$
		126	$X^7+X^4+5X^2+5$
		126	$X^7+X^4+5X^2+7$
		126	$X^7+X^4+7X^2+3$
7	16	252	$X^7+X^4+X^2+1$
		252	$X^7+X^4+X^2+3$
		252	$X^7+X^4+X^2+5$
		252	$X^7+X^4+X^2+9$
		252	$X^7+X^4+X^2+11$
		252	$X^7+X^4+X^2+13$
		252	$X^7+X^4+3X^2+1$
		252	$X^7+X^4+3X^2+3$
		252	$X^7+X^4+3X^2+7$
		252	$X^7+X^4+3X^2+9$
8	4	254	X^8+X^2+X+1
		254	X^8+X^2+3X+3
		254	X^8+3X^2+X+3
		254	X^8+3X^2+3X+1
		254	X^8+X^4+X+1
		254	X^8+X^4+3X+3
		254	$X^8+X^4+X^3+1$
		254	$X^8+X^4+3X^3+3$
		254	X^8+3X^4+X+3
		254	X^8+3X^4+3X+1
8	8	254	X^8+X^2+X+1
		254	X^8+X^2+X+3
		254	X^8+X^2+X+5
		254	X^8+X^2+3X+1
		254	X^8+X^2+3X+3
		254	X^8+X^2+3X+7
		254	X^8+X^2+5X+1
		254	X^8+X^2+5X+5
		254	X^8+X^2+5X+7
		254	X^8+X^2+7X+3

C.2 LFSR—Maximal cycle length over $\mathrm{d}_{\mathrm{I}}\mathrm{I}\exists\check{P}olys$

The following results have been computed in about 3,5h. After around half a day, the computation was aborted, so there are no results for $N = 8, D = 16$.

Coefficients are given in hexadecimal notation, so for an element a from the carrier $\tilde{\mathbb{P}}$ with generating monomial t, we write $1 \to 01, (t+1) \to 03, (t^2+1) \to 05, \ldots, (t^4+1) \to 011, \ldots$ for a compact representation.

		$\mathrm{d}_{\mathrm{I}}\mathrm{I}\exists\check{P}olys$	
N	D	cycle	$R(X)$
2	4	4	$X^2 + 03$
		2	$X^2 + 03X$
		1	$X^2 + 01$
		1	$X^2 + X$
2	8	8	$X^2 + 03$
		8	$X^2 + 07$
		4	$X^2 + 05$
		4	$X^2 + 03X$
		4	$X^2 + 07X$
		2	$X^2 + 05X$
		1	$X^2 + 01$
		1	$X^2 + X$
2	16	8	$X^2 + 03$
		8	$X^2 + 07$
		8	$X^2 + 0b$
		8	$X^2 + 0f$
		4	$X^2 + 05$
		4	$X^2 + 09$
		4	$X^2 + 0d$
		4	$X^2 + 03X$
		4	$X^2 + 07X$
		4	$X^2 + 11X$
3	4	6	$X^3 + 03$
		4	$X^3 + 03X$
		4	$X^3 + X^2 + X + 03$
		4	$X^3 + X^2 + 03X + 01$
		4	$X^3 + 03X^2 + X + 01$
		4	$X^3 + 03X^2 + 03X + 03$
		2	$X^3 + 03X^2$
		1	$X^3 + 01$
		1	$X^3 + X$
		1	$X^3 + X^2$

			$\mathbb{F}\mathrm{lip}\check{P}olys$
N	D	cycle	$R(X)$
3	8	12	$X^3 + 03$
		12	$X^3 + 07$
		8	$X^3 + 03X$
		8	$X^3 + 07X$
		8	$X^3 + X^2 + X + 03$
		8	$X^3 + X^2 + X + 07$
		8	$X^3 + X^2 + 03X + 01$
		8	$X^3 + X^2 + 03X + 05$
		8	$X^3 + X^2 + 05X + 03$
		8	$X^3 + X^2 + 05X + 07$
3	16	16	$X^3 + X^2 + X + 03$
		16	$X^3 + X^2 + X + 07$
		16	$X^3 + X^2 + X + 0b$
		16	$X^3 + X^2 + X + 0f$
		16	$X^3 + X^2 + 03X + 01$
		16	$X^3 + X^2 + 03X + 05$
		16	$X^3 + X^2 + 03X + 09$
		16	$X^3 + X^2 + 03X + 13$
		16	$X^3 + X^2 + 05X + 03$
		16	$X^3 + X^2 + 05X + 07$
4	4	14	$X^4 + X^2 + X + 03$
		14	$X^4 + X^2 + 03X + 01$
		14	$X^4 + 03X^2 + X + 01$
		14	$X^4 + 03X^2 + 03X + 03$
		14	$X^4 + X^3 + X^2 + 03$
		14	$X^4 + X^3 + 03X^2 + 01$
		14	$X^4 + 03X^3 + X^2 + 01$
		14	$X^4 + 03X^3 + 03X^2 + 03$
		12	$X^4 + X^3 + X + 03$
		12	$X^4 + X^3 + 03X + 01$
4	8	28	$X^4 + X^2 + X + 03$
		28	$X^4 + X^2 + X + 07$
		28	$X^4 + X^2 + 03X + 01$
		28	$X^4 + X^2 + 03X + 05$
		28	$X^4 + X^2 + 05X + 03$
		28	$X^4 + X^2 + 05X + 07$
		28	$X^4 + X^2 + 07X + 01$
		28	$X^4 + X^2 + 07X + 05$
		28	$X^4 + 03X^2 + X + 01$

N	D	cycle	$R(X)$
		28	$X^4 + 03X^2 + X + 05$
4	16	28	$X^4 + X^2 + X + 03$
		28	$X^4 + X^2 + X + 07$
		28	$X^4 + X^2 + X + 0b$
		28	$X^4 + X^2 + X + 0f$
		28	$X^4 + X^2 + 03X + 01$
		28	$X^4 + X^2 + 03X + 05$
		28	$X^4 + X^2 + 03X + 09$
		28	$X^4 + X^2 + 03X + 0d$
		28	$X^4 + X^2 + 05X + 03$
		28	$X^4 + X^2 + 05X + 07$
5	4	30	$X^5 + X^3 + X + 03$
		30	$X^5 + X^3 + 03X + 01$
		30	$X^5 + 03X^3 + X + 01$
		30	$X^5 + 03X^3 + 03X + 03$
		30	$X^5 + X^4 + X^2 + 03$
		30	$X^5 + X^4 + 03X^2 + 01$
		30	$X^5 + 03X^4 + X^2 + 01$
		30	$X^5 + 03X^4 + 03X^2 + 03$
		28	$X^5 + X^2 + X + 03$
		28	$X^5 + X^2 + 03X + 01$
5	8	60	$X^5 + X^3 + X + 03$
		60	$X^5 + X^3 + X + 07$
		60	$X^5 + X^3 + 03X + 01$
		60	$X^5 + X^3 + 03X + 05$
		60	$X^5 + X^3 + 05X + 03$
		60	$X^5 + X^3 + 05X + 07$
		60	$X^5 + X^3 + 07X + 01$
		60	$X^5 + X^3 + 07X + 05$
		60	$X^5 + 03X^3 + X + 01$
		60	$X^5 + 03X^3 + X + 05$
5	16	60	$X^5 + X^3 + X + 03$
		60	$X^5 + X^3 + X + 07$
		60	$X^5 + X^3 + X + 0b$
		60	$X^5 + X^3 + X + 0f$
		60	$X^5 + X^3 + 03X + 01$
		60	$X^5 + X^3 + 03X + 05$
		60	$X^5 + X^3 + 03X + 09$
		60	$X^5 + X^3 + 03X + 0d$

		$\check{Polys}^{\text{Flip}}$	
N	D	cycle	$R(X)$
		60	$X^5 + X^3 + 05X + 03$
		60	$X^5 + X^3 + 05X + 07$
6	4	62	$X^6 + X^2 + X + 03$
		62	$X^6 + X^2 + 03X + 01$
		62	$X^6 + 03X^2 + X + 01$
		62	$X^6 + 03X^2 + 03X + 03$
		62	$X^6 + X^3 + X^2 + 03$
		62	$X^6 + X^3 + 03X^2 + 01$
		62	$X^6 + 03X^3 + X^2 + 01$
		62	$X^6 + 03X^3 + 03X^2 + 03$
		62	$X^6 + X^4 + X^3 + 03$
		62	$X^6 + X^4 + 03X^3 + 01$
6	8	124	$X^6 + X^2 + X + 03$
		124	$X^6 + X^2 + X + 07$
		124	$X^6 + X^2 + 03X + 01$
		124	$X^6 + X^2 + 03X + 05$
		124	$X^6 + X^2 + 05X + 03$
		124	$X^6 + X^2 + 05X + 07$
		124	$X^6 + X^2 + 07X + 01$
		124	$X^6 + X^2 + 07X + 05$
		124	$X^6 + 03X^2 + X + 01$
		124	$X^6 + 03X^2 + X + 05$
6	16	124	$X^6 + X^2 + X + 03$
		124	$X^6 + X^2 + X + 07$
		124	$X^6 + X^2 + X + 0b$
		124	$X^6 + X^2 + X + 0f$
		124	$X^6 + X^2 + 03X + 01$
		124	$X^6 + X^2 + 03X + 05$
		124	$X^6 + X^2 + 03X + 09$
		124	$X^6 + X^2 + 03X + 0d$
		124	$X^6 + X^2 + 05X + 03$
		124	$X^6 + X^2 + 05X + 07$
7	4	126	$X^7 + X^4 + X^2 + 03$
		126	$X^7 + X^4 + 03X^2 + 01$
		126	$X^7 + 03X^4 + X^2 + 01$
		126	$X^7 + 03X^4 + 03X^2 + 03$
		126	$X^7 + X^5 + X + 03$
		126	$X^7 + X^5 + 03X + 01$
		126	$X^7 + X^5 + X^3 + 03$

			$\check{Polys}^{\mathrm{Flip}}$
N	D	cycle	$R(X)$
		126	$X^7 + X^5 + 03X^3 + 01$
		126	$X^7 + X^5 + X^4 + X^2 + X + 03$
		126	$X^7 + X^5 + X^4 + X^2 + 03X + 01$
7	8	252	$X^7 + X^4 + X^2 + 03$
		252	$X^7 + X^4 + X^2 + 07$
		252	$X^7 + X^4 + 03X^2 + 01$
		252	$X^7 + X^4 + 03X^2 + 05$
		252	$X^7 + X^4 + 05X^2 + 03$
		252	$X^7 + X^4 + 05X^2 + 07$
		252	$X^7 + X^4 + 07X^2 + 01$
		252	$X^7 + X^4 + 07X^2 + 05$
		252	$X^7 + 03X^4 + X^2 + 01$
		252	$X^7 + 03X^4 + X^2 + 05$
7	16	2052	$X^7 + X^4 + X^2 + 03$
		252	$X^7 + X^4 + X^2 + 07$
		252	$X^7 + X^4 + X^2 + 0b$
		252	$X^7 + X^4 + X^2 + 0f$
		252	$X^7 + X^4 + 03X^2 + 01$
		252	$X^7 + X^4 + 03X^2 + 05$
		252	$X^7 + X^4 + 03X^2 + 09$
		252	$X^7 + X^4 + 03X^2 + 0d$
		252	$X^7 + X^4 + 05X^2 + 03$
		252	$X^7 + X^4 + 05X^2 + 07$
8	4	254	$X^8 + X^2 + X + 03$
		254	$X^8 + X^2 + 03X + 01$
		254	$X^8 + 03X^2 + X + 01$
		254	$X^8 + 03X^2 + 03X + 03$
		254	$X^8 + X^4 + X + 03$
		254	$X^8 + X^4 + 03X + 01$
		254	$X^8 + X^4 + X^3 + 03$
		254	$X^8 + X^4 + 03X^3 + 01$
		254	$X^8 + 03X^4 + X + 01$
		254	$X^8 + 03X^4 + 03X + 03$
8	8	0508	$X^8 + X^2 + X + 03$
		508	$X^8 + X^2 + X + 07$
		508	$X^8 + X^2 + 03X + 01$
		508	$X^8 + X^2 + 03X + 05$
		508	$X^8 + X^2 + 05X + 03$
		508	$X^8 + X^2 + 05X + 07$

		$\mathrm{d}_{\dot{\mathrm{I}}\mathrm{l}}\mathbb{F}\check{P}olys$	
N	D	cycle	$R(X)$
		508	$X^8 + X^2 + 07X + 01$
		508	$X^8 + X^2 + 07X + 05$
		508	$X^8 + 03X^2 + X + 01$
		508	$X^8 + 03X^2 + X + 05$

C.3 LFSR—States over $\mathrm{d}_{\dot{\mathrm{I}}\mathrm{l}}\mathbb{F}\dot{Int}s$

This section contains some state developments for different values $D = 2^{d+1}$ and given dimension d for $\mathrm{d}_{\dot{\mathrm{I}}\mathrm{l}}\mathbb{F}\dot{Int}s$ and maximal degree N of the reduction polynomial $R(X)$. More details how to read the states can be found in Sect. 4.6.

Cycle length 14 for $N = 4, D = 4, R(X) = X^4 + X^2 + X + 1$.
$X^3 + X^2 + X + 1 \to 3X^3 + X^2 + X + 1 \to 3X^3 + 3X^2 + X + 1 \to X^3 + 3X^2 + 3X + 1 \to$
$3X^3 + X^2 + 3X + 3 \to 3X^3 + 3X^2 + X + 3 \to 3X^3 + 3X^2 + 3X + 1 \to 3X^3 + 3X^2 + 3X + 3 \to$
$X^3 + 3X^2 + 3X + 3 \to X^3 + X^2 + 3X + 3 \to 3X^3 + X^2 + X + 3 \to X^3 + 3X^2 + X + 1 \to$
$X^3 + X^2 + 3X + 1 \to X^3 + X^2 + X + 3$
$\rightsquigarrow (X^3 + X^2 + X + 1)$ *repetition*

Cycle length 14 for $N = 4, D = 8, R(X) = X^4 + X^2 + X + 1$.
$X^3 + X^2 + X + 1 \to 3X^3 + X^2 + X + 1 \to 3X^3 + 3X^2 + X + 1 \to 5X^3 + 3X^2 + 3X + 1 \to$
$7X^3 + 5X^2 + 3X + 3 \to 3X^3 + 7X^2 + 5X + 3 \to 7X^3 + 3X^2 + 7X + 5 \to 7X^3 + 7X^2 + 3X + 7 \to$
$X^3 + 7X^2 + 7X + 3 \to X^3 + X^2 + 7X + 7 \to 7X^3 + X^2 + X + 7 \to X^3 + 7X^2 + X + 1 \to$
$X^3 + X^2 + 7X + 1 \to X^3 + X^2 + X + 7$
$\rightsquigarrow (X^3 + X^2 + X + 1)$ *repetition*

Cycle length 28 for $N = 4, D = 16, R(X) = X^4 + X^2 + X + 1$.
$X^3 + X^2 + X + 1 \to 3X^3 + X^2 + X + 1 \to 3X^3 + 3X^2 + X + 1 \to 5X^3 + 3X^2 + 3X + 1 \to$
$7X^3 + 5X^2 + 3X + 3 \to 11X^3 + 7X^2 + 5X + 3 \to 15X^3 + 11X^2 + 7X + 5 \to 7X^3 + 15X^2 + 11X + 7 \to$
$X^3 + 7X^2 + 15X + 11 \to X^3 + X^2 + 7X + 15 \to 7X^3 + X^2 + X + 7 \to 9X^3 + 7X^2 + X + 1 \to$
$9X^3 + 9X^2 + 7X + 1 \to X^3 + 9X^2 + 9X + 7 \to 9X^3 + X^2 + 9X + 9 \to 3X^3 + 9X^2 + X + 9 \to$
$3X^3 + 3X^2 + 9X + 1 \to 13X^3 + 3X^2 + 3X + 9 \to 15X^3 + 13X^2 + 3X + 3 \to 3X^3 + 15X^2 + 13X + 3 \to$
$15X^3 + 3X^2 + 15X + 13 \to 15X^3 + 15X^2 + 3X + 15 \to X^3 + 15X^2 + 15X + 3 \to X^3 + X^2 + 15X + 15 \to$
$15X^3 + X^2 + X + 15 \to X^3 + 15X^2 + X + 1 \to X^3 + X^2 + 15X + 1 \to X^3 + X^2 + X + 15$
$\rightsquigarrow (X^3 + X^2 + X + 1)$ *repetition*

Cycle length 30 for $N = 5, D = 4, R(X) = X^5 + X^3 + X + 1$.
$X^4 + X^3 + X^2 + X + 1 \to 3X^4 + X^3 + X^2 + X + 1 \to 3X^4 + 3X^3 + X^2 + X + 1 \to X^4 + 3X^3 +$
$3X^2 + X + 1 \to X^4 + X^3 + 3X^2 + 3X + 1 \to X^4 + X^3 + X^2 + 3X + 3 \to 3X^4 + X^3 + X^2 + X + 3 \to$
$X^4 + 3X^3 + X^2 + X + 1 \to X^4 + X^3 + 3X^2 + X + 1 \to 3X^4 + X^3 + X^2 + 3X + 1 \to$
$X^4 + 3X^3 + X^2 + X + 3 \to 3X^4 + X^3 + 3X^2 + X + 1 \to 3X^4 + 3X^3 + X^2 + 3X + 1 \to$
$3X^4 + 3X^3 + 3X^2 + X + 3 \to 3X^4 + 3X^3 + 3X^2 + 3X + 1 \to 3X^4 + 3X^3 + 3X^2 + 3X + 3 \to$
$X^4 + 3X^3 + 3X^2 + 3X + 3 \to X^4 + X^3 + 3X^2 + 3X + 3 \to 3X^4 + X^3 + X^2 + 3X + 3 \to$
$3X^4 + 3X^3 + X^2 + X + 3 \to 3X^4 + 3X^3 + 3X^2 + X + 1 \to X^4 + 3X^3 + 3X^2 + 3X + 1 \to$
$3X^4 + X^3 + 3X^2 + 3X + 3 \to 3X^4 + 3X^3 + X^2 + 3X + 3 \to X^4 + 3X^3 + 3X^2 + X + 3 \to$
$3X^4 + X^3 + 3X^2 + 3X + 1 \to X^4 + 3X^3 + X^2 + 3X + 3 \to X^4 + X^3 + 3X^2 + X + 3 \to$

$X^4+X^3+X^2+3X+1 \to X^4+X^3+X^2+X+3$
$\rightsquigarrow (X^4+X^3+X^2+X+1)$ *repetition*

Cycle length 30 for $N=5, D=8, R(X)=X^5+X^3+X+1$.
$X^4+X^3+X^2+X+1 \to 3X^4+X^3+X^2+X+1 \to 3X^4+3X^3+X^2+X+1 \to$
$5X^4+3X^3+3X^2+X+1 \to 5X^4+5X^3+3X^2+3X+1 \to X^4+5X^3+5X^2+3X+3 \to$
$3X^4+X^3+5X^2+5X+3 \to X^4+3X^3+X^2+5X+5 \to 5X^4+X^3+3X^2+X+5 \to$
$7X^4+5X^3+X^2+3X+1 \to X^4+7X^3+5X^2+X+3 \to 3X^4+X^3+7X^2+5X+1 \to$
$7X^4+3X^3+X^2+7X+5 \to 7X^4+7X^3+3X^2+X+7 \to 7X^4+7X^3+7X^2+3X+1 \to$
$3X^4+7X^3+7X^2+7X+3 \to X^4+3X^3+7X^2+7X+7 \to X^4+X^3+3X^2+7X+7 \to$
$7X^4+X^3+X^2+3X+7 \to 3X^4+7X^3+X^2+X+3 \to 3X^4+3X^3+7X^2+X+1 \to$
$5X^4+3X^3+3X^2+7X+1 \to 3X^4+5X^3+3X^2+3X+7 \to 7X^4+3X^3+5X^2+3X+3 \to$
$X^4+7X^3+3X^2+5X+3 \to 7X^4+X^3+7X^2+3X+5 \to X^4+7X^3+X^2+7X+3 \to$
$X^4+X^3+7X^2+X+7 \to X^4+X^3+X^2+7X+1 \to X^4+X^3+X^2+X+7$
$\rightsquigarrow (X^4+X^3+X^2+X+1)$ *repetition*

Cycle length 40 for $N=5, D=16, R(X)=X^5+X^3+X+1$.
$X^4+X^3+X^2+X+1 \to 3X^4+X^3+X^2+X+1 \to 3X^4+3X^3+X^2+X+1 \to$
$5X^4+3X^3+3X^2+X+1 \to 5X^4+5X^3+3X^2+3X+1 \to 9X^4+5X^3+5X^2+3X+3 \to$
$11X^4+9X^3+5X^2+5X+3 \to X^4+11X^3+9X^2+5X+5 \to 5X^4+X^3+11X^2+9X+5 \to$
$15X^4+5X^3+X^2+11X+9 \to 9X^4+15X^3+5X^2+X+11 \to 11X^4+9X^3+15X^2+5X+1 \to$
$15X^4+11X^3+9X^2+15X+5 \to 15X^4+15X^3+11X^2+9X+15 \to 7X^4+15X^3+15X^2+11X+9 \to$
$3X^4+7X^3+15X^2+15X+11 \to X^4+3X^3+7X^2+15X+15 \to X^4+X^3+3X^2+7X+15 \to$
$7X^4+X^3+X^2+3X+7 \to 11X^4+7X^3+X^2+X+3 \to 11X^4+11X^3+7X^2+X+1 \to$
$13X^4+11X^3+11X^2+7X+1 \to 3X^4+13X^3+11X^2+11X+7 \to 15X^4+3X^3+13X^2+11X+11 \to$
$9X^4+15X^3+3X^2+13X+11 \to 7X^4+9X^3+15X^2+3X+13 \to 9X^4+7X^3+9X^2+15X+3 \to$
$9X^4+9X^3+7X^2+9X+15 \to X^4+9X^3+9X^2+7X+9 \to 9X^4+X^3+9X^2+9X+7 \to$
$X^4+9X^3+X^2+9X+9 \to 11X^4+X^3+9X^2+X+9 \to 11X^4+11X^3+X^2+9X+1 \to$
$5X^4+11X^3+11X^2+X+9 \to 5X^4+5X^3+11X^2+11X+1 \to X^4+5X^3+5X^2+11X+11 \to$
$11X^4+X^3+5X^2+5X+11 \to X^4+11X^3+X^2+5X+5 \to 5X^4+X^3+11X^2+X+5 \to$
$7X^4+5X^3+X^2+11X+1 \to X^4+7X^3+5X^2+X+11 \to 3X^4+X^3+7X^2+5X+1 \to$
$7X^4+3X^3+X^2+7X+5 \to 15X^4+7X^3+3X^2+X+7 \to 15X^4+15X^3+7X^2+3X+1 \to$
$3X^4+15X^3+15X^2+7X+3 \to 9X^4+3X^3+15X^2+15X+7 \to 9X^4+9X^3+3X^2+15X+15 \to$
$7X^4+9X^3+9X^2+3X+15 \to 11X^4+7X^3+9X^2+9X+3 \to 3X^4+11X^3+7X^2+9X+9 \to$
$13X^4+3X^3+11X^2+7X+9 \to 3X^4+13X^3+3X^2+11X+7 \to 15X^4+3X^3+13X^2+3X+11 \to$
$X^4+15X^3+3X^2+13X+3 \to 15X^4+X^3+15X^2+3X+13 \to X^4+15X^3+X^2+15X+3 \to$
$X^4+X^3+15X^2+X+15 \to X^4+X^3+X^2+15X+1 \to X^4+X^3+X^2+X+15$
$\rightsquigarrow (X^4+X^3+X^2+X+1)$ *repetition*

C.4 LFSR—States over dılƎ*P̆olys*

This section contains some state developments for different values $D=2^{d+1}$ and given dimension d for dılƎ*P̆olys* and maximal degree N of the reduction polynomial $R(X)$. More details how to read the states can be found in Sect. 4.6.

Coefficients are given in hexadecimal notation, so for an element a from the carrier $\tilde{\mathbb{P}}$ with generating monomial t, we write $1 \to 01, (t+1) \to 03, (t^2+1) \to 05, \ldots, (t^4+1) \to 011, \ldots$ for a compact representation.

Cycle length 14 for $N = 4, D = 4, R(X) = X^4 + X^2 + X + 3$.
$X^3+X^2+X+01 \to 03X^3+X^2+X+01 \to 03X^3+03X^2+X+01 \to X^3+03X^2+03X+01 \to 03X^3 + X^2 + 03X + 03 \to 03X^3 + 03X^2 + X + 03 \to 03X^3 + 03X^2 + 03X + 01 \to 03X^3 + 03X^2 + 03X + 03 \to X^3 + 03X^2 + 03X + 03 \to X^3 + X^2 + 03X + 03 \to 03X^3 + X^2 + X + 03 \to X^3 + 03X^2 + X + 01 \to X^3 + X^2 + 03X + 01 \to X^3 + X^2 + X + 03$
$\rightsquigarrow (X^3 + X^2 + X + 01)$ *repetition*

Cycle length 28 for $N = 4, D = 8, R(X) = X^4 + X^2 + X + 3$.
$X^3+X^2+X+01 \to 03X^3+X^2+X+01 \to 03X^3+03X^2+X+01 \to X^3+03X^2+03X+01 \to 03X^3+X^2+03X+03 \to 07X^3+03X^2+X+03 \to 07X^3+07X^2+03X+01 \to 07X^3+07X^2+07X+03 \to 05X^3+07X^2+07X+07 \to X^3+05X^2+07X+07 \to 03X^3+X^2+05X+07 \to 05X^3+03X^2+X+05 \to 05X^3+05X^2+03X+01 \to 05X^3+05X^2+05X+03 \to 05X^3+05X^2+05X+05 \to 07X^3+05X^2+05X+05 \to 07X^3+07X^2+05X+05 \to 05X^3+07X^2+07X+05 \to 07X^3 + 05X^2 + 07X + 07 \to 03X^3 + 07X^2 + 05X + 07 \to 03X^3 + 03X^2 + 07X + 05 \to 03X^3 + 03X^2+03X+07 \to X^3+03X^2+03X+03 \to 05X^3+X^2+03X+03 \to 07X^3+05X^2+X+03 \to X^3 + 07X^2 + 05X + 01 \to X^3 + X^2 + 07X + 05 \to X^3 + X^2 + X + 07$
$\rightsquigarrow (X^3 + X^2 + X + 01)$ *repetition*

Cycle length 28 for $N = 4, D = 16, R(X) = X^4 + X^2 + X + 3$.
$X^3+X^2+X+01 \to 03X^3+X^2+X+01 \to 03X^3+03X^2+X+01 \to X^3+03X^2+03X+01 \to 03X^3+X^2+03X+03 \to 07X^3+03X^2+X+03 \to 07X^3+07X^2+03X+01 \to 07X^3+07X^2+07X+03 \to 05X^3+07X^2+07X+07 \to 09X^3+05X^2+07X+07 \to 0bX^3+09X^2+05X+07 \to 05X^3 + 0bX^2 + 09X + 05 \to 0dX^3 + 05X^2 + 0bX + 09 \to 05X^3 + 0dX^2 + 05X + 0b \to 05X^3 + 05X^2 + 0dX + 05 \to 07X^3 + 05X^2 + 05X + 0d \to 07X^3 + 07X^2 + 05X + 05 \to 0dX^3+07X^2+07X+05 \to 0fX^3+0dX^2+07X+07 \to 03X^3+0fX^2+0dX+07 \to 0bX^3+03X^2+0fX+0d \to 0bX^3+0bX^2+03X+0f \to 09X^3+0bX^2+0bX+03 \to 05X^3+09X^2+0bX+0b \to 0fX^3+05X^2+09X+0b \to X^3+0fX^2+05X+09 \to X^3+X^2+0fX+05 \to X^3+X^2+X+0f$
$\rightsquigarrow (X^3 + X^2 + X + 01)$ *repetition*

Cycle length 30 for $N = 5, D = 4, R(X) = X^5 + X^3 + X + 3$.
$X^4 + X^3 + X^2 + X + 01 \to 03X^4 + X^3 + X^2 + X + 01 \to 03X^4 + 03X^3 + X^2 + X + 01 \to X^4 + 03X^3 + 03X^2 + X + 01 \to X^4 + X^3 + 03X^2 + 03X + 01 \to X^4 + X^3 + X^2 + 03X + 03 \to 03X^4 + X^3 + X^2 + X + 03 \to X^4 + 03X^3 + X^2 + X + 01 \to X^4 + X^3 + 03X^2 + X + 01 \to 03X^4+X^3+X^2+03X+01 \to X^4+03X^3+X^2+X+03 \to 03X^4+X^3+03X^2+X+01 \to 03X^4+03X^3+X^2+03X+01 \to 03X^4+03X^3+03X^2+X+03 \to 03X^4+03X^3+03X^2+03X+01 \to 03X^4+03X^3+03X^2+03X+03 \to X^4+03X^3+03X^2+03X+03 \to X^4+X^3+03X^2+03X+03 \to 03X^4+X^3+X^2+03X+03 \to 03X^4+03X^3+X^2+X+03 \to 03X^4+03X^3+03X^2+X+01 \to X^4+03X^3+03X^2+03X+01 \to 03X^4+X^3+03X^2+03X+03 \to 03X^4+03X^3+X^2+03X+03 \to X^4+03X^3+03X^2+X+03 \to 03X^4+X^3+03X^2+03X+01 \to X^4+03X^3+X^2+03X+03 \to X^4 + X^3 + 03X^2 + X + 03 \to X^4 + X^3 + X^2 + 03X + 01 \to X^4 + X^3 + X^2 + X + 03$
$\rightsquigarrow (X^4 + X^3 + X^2 + X + 01)$ *repetition*

Cycle length 60 for $N = 5, D = 8, R(X) = X^5 + X^3 + X + 3$.
$X^4 + X^3 + X^2 + X + 01 \to 03X^4 + X^3 + X^2 + X + 01 \to 03X^4 + 03X^3 + X^2 + X + 01 \to X^4 + 03X^3 + 03X^2 + X + 01 \to X^4 + X^3 + 03X^2 + 03X + 01 \to X^4 + X^3 + X^2 + 03X + 03 \to 07X^4 + X^3 + X^2 + X + 03 \to 05X^4 + 07X^3 + X^2 + X + 01 \to 05X^4 + 05X^3 + 07X^2 + X + 01 \to 07X^4 + 05X^3 + 05X^2 + 07X + 01 \to X^4 + 07X^3 + 05X^2 + 05X + 07 \to 03X^4 + X^3 + 07X^2 + 05X + 05 \to 03X^4 + 03X^3 + X^2 + 07X + 05 \to 03X^4 + 03X^3 + 03X^2 + X + 07 \to 03X^4 + 03X^3 + 03X^2 + 03X + 01 \to 03X^4 + 03X^3 + 03X^2 + 03X + 03 \to 05X^4 + 03X^3 + 03X^2 + 03X + 03 \to$

$05X^4 + 05X^3 + 03X^2 + 03X + 03 \to 03X^4 + 05X^3 + 05X^2 + 03X + 03 \to 03X^4 + 03X^3 + 05X^2 + 05X + 03 \to 03X^4 + 03X^3 + 03X^2 + 05X + 05 \to X^4 + 03X^3 + 03X^2 + 03X + 05 \to 07X^4 + X^3 + 03X^2 + 03X + 03 \to 07X^4 + 07X^3 + X^2 + 03X + 03 \to X^4 + 07X^3 + 07X^2 + X + 03 \to 03X^4 + X^3 + 07X^2 + 07X + 01 \to 05X^4 + 03X^3 + X^2 + 07X + 07 \to 05X^4 + 05X^3 + 03X^2 + X + 07 \to 05X^4 + 05X^3 + 05X^2 + 03X + 01 \to 05X^4 + 05X^3 + 05X^2 + 05X + 03 \to 05X^4 + 05X^3 + 05X^2 + 05X + 05 \to 07X^4 + 05X^3 + 05X^2 + 05X + 05 \to 07X^4 + 07X^3 + 05X^2 + 05X + 05 \to 05X^4 + 07X^3 + 07X^2 + 05X + 05 \to 05X^4 + 05X^3 + 07X^2 + 07X + 05 \to 05X^4 + 05X^3 + 05X^2 + 07X + 07 \to 03X^4 + 05X^3 + 05X^2 + 05X + 07 \to X^4 + 03X^3 + 05X^2 + 05X + 05 \to X^4 + X^3 + 03X^2 + 05X + 05 \to 03X^4 + X^3 + X^2 + 03X + 05 \to 05X^4 + 03X^3 + X^2 + X + 03 \to 07X^4 + 05X^3 + 03X^2 + X + 01 \to 07X^4 + 07X^3 + 05X^2 + 03X + 01 \to 07X^4 + 07X^3 + 07X^2 + 05X + 03 \to 07X^4 + 07X^3 + 07X^2 + 07X + 05 \to 07X^4 + 07X^3 + 07X^2 + 07X + 07 \to X^4 + 07X^3 + 07X^2 + 07X + 07 \to X^4 + X^3 + 07X^2 + 07X + 07 \to 07X^4 + X^3 + X^2 + 07X + 07 \to 07X^4 + 07X^3 + X^2 + X + 07 \to 07X^4 + 07X^3 + 07X^2 + X + 01 \to 05X^4 + 07X^3 + 07X^2 + 07X + 01 \to 03X^4 + 05X^3 + 07X^2 + 07X + 07 \to 03X^4 + 03X^3 + 05X^2 + 07X + 07 \to 05X^4 + 03X^3 + 03X^2 + 05X + 07 \to 07X^4 + 05X^3 + 03X^2 + 03X + 05 \to X^4 + 07X^3 + 05X^2 + 03X + 03 \to X^4 + X^3 + 07X^2 + 05X + 03 \to X^4 + X^3 + X^2 + 07X + 05 \to X^4 + X^3 + X^2 + X + 07$
$\rightsquigarrow (X^4 + X^3 + X^2 + X + 01)$ *repetition*

Cycle length 60 for $N = 5, D = 16, R(X) = X^5 + X^3 + X + 3$.
$X^4 + X^3 + X^2 + X + 01 \to 03X^4 + X^3 + X^2 + X + 01 \to 03X^4 + 03X^3 + X^2 + X + 01 \to X^4 + 03X^3 + 03X^2 + X + 01 \to X^4 + X^3 + 03X^2 + 03X + 01 \to X^4 + X^3 + X^2 + 03X + 03 \to 07X^4 + X^3 + X^2 + X + 03 \to 05X^4 + 07X^3 + X^2 + X + 01 \to 05X^4 + 05X^3 + 07X^2 + X + 01 \to 07X^4 + 05X^3 + 05X^2 + 07X + 01 \to X^4 + 07X^3 + 05X^2 + 05X + 07 \to 0bX^4 + X^3 + 07X^2 + 05X + 05 \to 0bX^4 + 0bX^3 + X^2 + 07X + 05 \to 03X^4 + 0bX^3 + 0bX^2 + X + 07 \to 03X^4 + 03X^3 + 0bX^2 + 0bX + 01 \to 0bX^4 + 03X^3 + 03X^2 + 0bX + 0b \to 05X^4 + 0bX^3 + 03X^2 + 03X + 0b \to 05X^4 + 05X^3 + 0bX^2 + 03X + 03 \to 03X^4 + 05X^3 + 05X^2 + 0bX + 03 \to 0bX^4 + 03X^3 + 05X^2 + 05X + 0b \to 0bX^4 + 0bX^3 + 03X^2 + 05X + 05 \to X^4 + 0bX^3 + 0bX^2 + 03X + 05 \to 07X^4 + X^3 + 0bX^2 + 0bX + 03 \to 0fX^4 + 07X^3 + X^2 + 0bX + 0b \to X^4 + 0fX^3 + 07X^2 + X + 0b \to 03X^4 + X^3 + 0fX^2 + 07X + 01 \to 05X^4 + 03X^3 + X^2 + 0fX + 07 \to 05X^4 + 05X^3 + 03X^2 + X + 0f \to 05X^4 + 05X^3 + 05X^2 + 03X + 01 \to 05X^4 + 05X^3 + 05X^2 + 05X + 03 \to 05X^4 + 05X^3 + 05X^2 + 05X + 05 \to 0fX^4 + 05X^3 + 05X^2 + 05X + 05 \to 0fX^4 + 0fX^3 + 05X^2 + 05X + 05 \to 05X^4 + 0fX^3 + 0fX^2 + 05X + 05 \to 05X^4 + 05X^3 + 0fX^2 + 0fX + 05 \to 05X^4 + 05X^3 + 05X^2 + 0fX + 0f \to 0bX^4 + 05X^3 + 05X^2 + 05X + 0f \to X^4 + 0bX^3 + 05X^2 + 05X + 05 \to X^4 + X^3 + 0bX^2 + 05X + 05 \to 0bX^4 + X^3 + X^2 + 0bX + 05 \to 05X^4 + 0bX^3 + X^2 + X + 0b \to 07X^4 + 05X^3 + 0bX^2 + X + 01 \to 07X^4 + 07X^3 + 05X^2 + 0bX + 01 \to 0fX^4 + 07X^3 + 07X^2 + 05X + 0b \to 0fX^4 + 0fX^3 + 07X^2 + 07X + 05 \to 07X^4 + 0fX^3 + 0fX^2 + 07X + 07 \to X^4 + 07X^3 + 0fX^2 + 0fX + 07 \to X^4 + X^3 + 07X^2 + 0fX + 0f \to 0fX^4 + X^3 + X^2 + 07X + 0f \to 07X^4 + 0fX^3 + X^2 + X + 07 \to 07X^4 + 07X^3 + 0fX^2 + X + 01 \to 05X^4 + 07X^3 + 07X^2 + 0fX + 01 \to 0bX^4 + 05X^3 + 07X^2 + 07X + 0f \to 03X^4 + 0bX^3 + 05X^2 + 07X + 07 \to 05X^4 + 03X^3 + 0bX^2 + 05X + 07 \to 0fX^4 + 05X^3 + 03X^2 + 0bX + 05 \to X^4 + 0fX^3 + 05X^2 + 03X + 0b \to X^4 + X^3 + 0fX^2 + 05X + 03 \to X^4 + X^3 + X^2 + 0fX + 05 \to X^4 + X^3 + X^2 + X + 0f$
$\rightsquigarrow (X^4 + X^3 + X^2 + X + 01)$ *repetition*

D Non-Linear Feedback Shift Registers (NLFSR)

D.1 Genetic algorithm for finding NLFSR with large cycle length

A genetic algorithm (GA) is a tool from discrete optimization that can be used even if there is only limited internal structure in the problem to work with. It consists of the following steps:

1. Generate initial **population**.
2. Select fit candidates for the next generation, either use mutation or crossover.
3. Compute the fitness of all old and new members of the population.
4. Delete candidates that are less fit than the others.

Candidates are usually called **genes** in the context of GA. While very simple, the combination of mutating existing genes, but also combining two genes into so-called **children**, this proves very powerful in practice. In particular, we call the **genes** the members of the population. Moreover, following the above 4 steps is called a **generation** of the GA. For more details, cf. [Wi24]. The most important part of a GA is the **fitness function**. For a give gene, it outputs a positive real number. In out case and some multivariate polynomial $p(x_0, \dots x_{N-1}$ for some $N \in \mathbb{N}$

$$f(c, w) := c + 1/w$$

for a some cycle length c and some width w. This clearly puts emphasis on a long cycle length c, but also a small number of terms w for the corresponding polynomial p. A gene in this GA becomes p, s where s is an initial state. Note that the state is not part of the fitness function.

For this algorithm, we have used a **population size** of 100, and up to 100 *generations*. If the fitness of the fittest members of the population stays the same, the GA will abort. See `gaRun` for the corresponding implementation. In particular, a *gene* is defined as (p, s) for some polynomial p and some initial state s.

A GA can be aborted if a certain number of repetitions of the above loop is completed. Each of these loop is called a **generation**. Alternatively, the algorithm terminates if a certain optimization result is reached. The corresponding Python code is given in the method `gaRun`.

As noted in Sect. 4.7, the results on NLFSR and LFSR cannot be compared directly: Due to the constant term in NLFSR, there is one degree of freedom, so the cycle length of a NLFSR in N variables will be higher than the LFSR in N variables.

D.2 NLFSR—Affine Polynomials

Here we see the maximal cycle length for polynomials over $\mathbb{F}_{\text{lip}}\mathit{Int}\dot{s}$ for $D = 16$ as computed by the genetic algorithm that was described in the previous section. A discussion can be found in Sect. 4.7.

Affine multivariate polynomials, generated via a genetic algorithm

vars	D	Width	Cycle	Polynomial
4	4	3	15	$x_1 + 3x_0 + 3$
		3	15	$3x_1 + x_0 + 3$
		3	15	$3x_3 + x_0 + 3$
		3	15	$x_3 + 3x_0 + 3$
		3	15	$3x_3 + 3x_0 + 1$

Affine multivariate polynomials, generated via a genetic algorithm

vars	D	Width	Cycle	Polynomial
		3	15	$x_1 + x_0 + 1$
		3	15	$x_3 + x_0 + 1$
		3	15	$3x_1 + 3x_0 + 1$
		3	14	$3x_2 + x_1 + 3x_0$
		3	14	$3x_3 + x_2 + 3x_0$
4	8	3	30	$5x_3 + 5x_0 + 1$
		3	30	$x_1 + x_0 + 1$
		3	30	$x_3 + 5x_0 + 1$
		3	30	$3x_3 + x_0 + 3$
		3	30	$x_3 + 3x_0 + 3$
		3	30	$3x_3 + 7x_0 + 5$
		3	30	$5x_3 + 5x_0 + 5$
		3	30	$3x_1 + 5x_0 + 3$
		3	30	$5x_3 + 7x_0 + 7$
		3	30	$x_3 + 7x_0 + 3$
4	16	3	60	$15x_1 + 15x_0 + 1$
		3	60	$11x_3 + x_0 + 11$
		3	60	$11x_1 + 15x_0 + 5$
		3	60	$7x_1 + 13x_0 + 7$
		3	60	$15x_1 + 7x_0 + 1$
		3	60	$7x_1 + 15x_0 + 1$
		3	60	$5x_1 + 9x_0 + 5$
		3	60	$13x_1 + x_0 + 5$
		3	60	$7x_1 + 7x_0 + 13$
		3	60	$3x_1 + x_0 + 3$
5	4	3	31	$3x_3 + x_0 + 3$
		3	31	$3x_3 + 3x_0 + 1$
		3	31	$3x_2 + 3x_0 + 1$
		3	31	$3x_2 + x_0 + 3$
		3	31	$x_2 + 3x_0 + 3$
		3	31	$x_2 + x_0 + 1$
		3	31	$x_3 + 3x_0 + 3$
		3	31	$x_3 + x_0 + 1$
		5	31	$x_4 + 3x_3 + 3x_2 + x_0 + 3$
		5	31	$3x_3 + 3x_2 + x_1 + x_0 + 3$
5	8	3	62	$x_3 + 7x_0 + 7$
		3	62	$x_3 + x_0 + 5$
		3	62	$5x_2 + 7x_0 + 3$
		3	62	$7x_2 + x_0 + 3$

Affine multivariate polynomials, generated via a genetic algorithm

vars	D	Width	Cycle	Polynomial
		3	62	$x_3 + 5x_0 + 5$
		3	62	$5x_3 + 7x_0 + 7$
		3	62	$x_3 + x_0 + 1$
		3	62	$3x_2 + 5x_0 + 3$
		3	62	$x_2 + 5x_0 + 1$
		3	62	$5x_2 + 5x_0 + 5$
5	16	3	124	$11x_3 + 7x_0 + 1$
		3	124	$15x_2 + 15x_0 + 5$
		3	124	$15x_2 + 3x_0 + 5$
		3	124	$5x_3 + 11x_0 + 3$
		3	124	$13x_3 + 13x_0 + 1$
		3	124	$7x_2 + 3x_0 + 1$
		3	124	$7x_2 + 3x_0 + 9$
		3	124	$7x_3 + 9x_0 + 7$
		3	124	$13x_3 + 13x_0 + 5$
		3	124	$15x_2 + 15x_0 + 1$
6	4	3	63	$x_1 + x_0 + 1$
		3	63	$x_5 + 3x_0 + 3$
		3	63	$3x_5 + x_0 + 3$
		3	63	$3x_1 + 3x_0 + 1$
		3	63	$3x_5 + 3x_0 + 1$
		3	63	$3x_1 + x_0 + 3$
		3	63	$x_5 + x_0 + 1$
		5	63	$3x_5 + 3x_3 + 3x_2 + 3x_0 + 3$
		5	63	$x_5 + x_2 + x_1 + 3x_0 + 1$
		5	63	$x_5 + x_4 + x_1 + 3x_0 + 1$
6	8	3	126	$7x_1 + 3x_0 + 1$
		3	126	$5x_5 + 3x_0 + 7$
		3	126	$3x_1 + 7x_0 + 1$
		3	126	$x_1 + 3x_0 + 7$
		3	126	$7x_5 + 7x_0 + 1$
		3	126	$x_1 + 5x_0 + 5$
		3	126	$3x_1 + 3x_0 + 5$
		3	126	$3x_1 + 7x_0 + 5$
		3	126	$x_5 + 5x_0 + 5$
		3	126	$7x_1 + 5x_0 + 3$
6	16	3	252	$5x_1 + 3x_0 + 11$
		3	252	$5x_1 + x_0 + 1$
		3	252	$13x_1 + 11x_0 + 7$

Affine multivariate polynomials, generated via a genetic algorithm

vars	D	Width	Cycle	Polynomial
		3	252	$13x_5 + 7x_0 + 3$
		3	252	$5x_1 + 15x_0 + 3$
		3	252	$9x_5 + 7x_0 + 3$
		3	252	$9x_1 + 15x_0 + 11$
		3	252	$15x_1 + 7x_0 + 1$
		3	252	$7x_1 + 7x_0 + 9$
		3	252	$15x_1 + 13x_0 + 3$
7	4	3	127	$3x_1 + 3x_0 + 1$
		3	127	$x_4 + 3x_0 + 3$
		3	127	$x_1 + 3x_0 + 3$
		3	127	$3x_6 + 3x_0 + 1$
		3	127	$3x_4 + 3x_0 + 1$
		3	127	$x_1 + x_0 + 1$
		3	127	$x_3 + 3x_0 + 3$
		3	127	$3x_3 + x_0 + 3$
		3	127	$3x_1 + x_0 + 3$
		3	127	$3x_6 + x_0 + 3$
7	8	3	254	$3x_4 + 5x_0 + 7$
		3	254	$5x_3 + 3x_0 + 3$
		3	254	$7x_3 + 7x_0 + 5$
		3	254	$3x_4 + 3x_0 + 1$
		3	254	$x_4 + x_0 + 5$
		3	254	$7x_6 + 3x_0 + 1$
		3	254	$7x_6 + 3x_0 + 5$
		3	254	$7x_4 + x_0 + 7$
		3	254	$x_6 + 5x_0 + 1$
		3	254	$x_1 + 5x_0 + 1$
7	16	3	508	$7x_6 + 3x_0 + 5$
		3	508	$9x_6 + 5x_0 + 1$
		3	508	$3x_6 + 15x_0 + 5$
		3	508	$11x_4 + x_0 + 15$
		3	508	$9x_6 + 3x_0 + 15$
		3	508	$5x_4 + 15x_0 + 15$
		3	508	$3x_1 + 13x_0 + 15$
		3	508	$5x_1 + 5x_0 + 9$
		3	508	$7x_3 + 3x_0 + 13$
		3	508	$3x_3 + 11x_0 + 5$
8	4	5	255	$x_7 + 3x_3 + 3x_2 + x_0 + 3$
		5	255	$3x_7 + 3x_5 + x_3 + 3x_0 + 1$

Affine multivariate polynomials, generated via a genetic algorithm

vars	D	Width	Cycle	Polynomial
		5	255	$x_7 + x_5 + 3x_3 + x_0 + 1$
		5	255	$x_7 + x_2 + x_1 + 3x_0 + 1$
		5	255	$x_6 + x_5 + 3x_1 + x_0 + 1$
		5	255	$3x_7 + x_5 + x_3 + x_0 + 1$
		5	255	$x_5 + 3x_3 + x_1 + x_0 + 1$
		5	255	$x_7 + 3x_5 + 3x_3 + x_0 + 3$
		5	255	$3x_7 + x_5 + 3x_3 + x_0 + 3$
		5	255	$3x_6 + x_5 + 3x_1 + 3x_0 + 1$
8	8	5	510	$x_7 + 7x_3 + x_2 + 5x_0 + 1$
		5	510	$7x_7 + 5x_5 + 7x_3 + 3x_0 + 5$
		5	510	$5x_4 + 3x_3 + 3x_2 + x_0 + 7$
		5	510	$x_7 + 7x_3 + 3x_2 + 3x_0 + 5$
		5	510	$3x_4 + x_3 + 3x_2 + 7x_0 + 5$
		5	510	$5x_7 + 7x_3 + 3x_2 + 3x_0 + 5$
		5	510	$7x_5 + 3x_3 + 5x_1 + 7x_0 + 5$
		5	510	$5x_4 + 7x_3 + 3x_2 + x_0 + 7$
		5	510	$7x_7 + 3x_5 + 7x_3 + 3x_0 + 7$
		5	510	$3x_6 + 3x_3 + 3x_2 + 3x_0 + 3$
8	16	5	1,020	$15x_5 + x_3 + 11x_1 + 11x_0 + 9$
		5	1,020	$3x_5 + 7x_3 + 3x_1 + 13x_0 + 9$
		5	1,020	$7x_5 + 11x_3 + 9x_1 + 9x_0 + 3$
		5	1,020	$3x_7 + 7x_3 + 3x_2 + x_0 + 5$
		5	1,020	$13x_7 + 13x_2 + 9x_1 + 3x_0 + 5$
		5	1,020	$9x_5 + 11x_3 + 15x_2 + 11x_0 + 1$
		5	1,020	$11x_5 + 7x_3 + 3x_1 + 13x_0 + 1$
		5	1,020	$3x_6 + 13x_5 + 9x_3 + x_0 + 13$
		5	1,020	$3x_6 + 7x_5 + 7x_3 + 3x_0 + 11$
		5	1,020	$13x_7 + 13x_2 + 7x_1 + 3x_0 + 11$
9	4	3	511	$x_5 + 3x_0 + 3$
		3	511	$x_4 + x_0 + 1$
		3	511	$3x_4 + 3x_0 + 1$
		3	511	$x_4 + 3x_0 + 3$
		3	511	$x_5 + x_0 + 1$
		3	511	$3x_5 + 3x_0 + 1$
		5	511	$x_8 + 3x_5 + x_4 + 3x_0 + 3$
		5	511	$x_8 + x_5 + 3x_4 + 3x_0 + 3$
		5	511	$3x_8 + 3x_4 + 3x_1 + x_0 + 1$
		5	511	$3x_7 + x_5 + 3x_1 + 3x_0 + 1$
9	8	3	1,022	$7x_4 + 3x_0 + 5$

Affine multivariate polynomials, generated via a genetic algorithm

vars	D	Width	Cycle	Polynomial
		3	1,022	$3x_5 + 5x_0 + 7$
		3	1,022	$3x_4 + 7x_0 + 5$
		3	1,022	$7x_5 + 5x_0 + 7$
		3	1,022	$3x_4 + 3x_0 + 5$
		3	1,022	$5x_4 + 7x_0 + 3$
		3	1,022	$7x_4 + 7x_0 + 5$
		3	1,022	$7x_4 + 5x_0 + 3$
		3	1,022	$5x_4 + 3x_0 + 7$
		3	1,022	$5x_4 + x_0 + 5$
9	16	3	2,044	$13x_4 + 13x_0 + 9$
		3	2,044	$13x_4 + 13x_0 + 1$
		3	2,044	$5x_4 + 15x_0 + 7$
		3	2,044	$9x_5 + 13x_0 + 9$
		3	2,044	$13x_4 + 15x_0 + 7$
		3	2,044	$5x_4 + x_0 + 5$
		3	2,044	$13x_4 + 5x_0 + 1$
		3	2,044	$5x_4 + 3x_0 + 7$
		3	2,044	$15x_5 + 13x_0 + 15$
		3	2,044	$13x_4 + 13x_0 + 5$
10	4	3	1,023	$x_3 + x_0 + 1$
		3	1,023	$x_7 + 3x_0 + 3$
		3	1,023	$x_3 + 3x_0 + 3$
		3	1,023	$3x_3 + 3x_0 + 1$
		3	1,023	$3x_3 + x_0 + 3$
		5	1,023	$x_7 + x_3 + x_1 + x_0 + 3$
		5	1,023	$3x_9 + 3x_6 + x_1 + x_0 + 3$
		5	1,023	$x_9 + 3x_4 + x_2 + x_0 + 1$
		5	1,023	$3x_5 + 3x_3 + 3x_2 + x_0 + 1$
		5	1,023	$x_8 + x_7 + 3x_5 + x_0 + 1$
10	8	3	2,046	$3x_7 + 7x_0 + 1$
		3	2,046	$3x_7 + x_0 + 3$
		3	2,046	$x_3 + x_0 + 5$
		3	2,046	$x_3 + x_0 + 1$
		3	2,046	$3x_7 + x_0 + 7$
		3	2,046	$7x_7 + 7x_0 + 1$
		3	2,046	$7x_3 + x_0 + 3$
		3	2,046	$5x_7 + x_0 + 1$
		3	2,046	$5x_7 + x_0 + 5$

Affine multivariate polynomials, generated via a genetic algorithm

vars	D	Width	Cycle	Polynomial
		3	2,046	$7x_3 + x_0 + 7$
10	16	3	4,092	$13x_7 + 3x_0 + 7$
		3	4,092	$15x_7 + 15x_0 + 9$
		3	4,092	$9x_3 + 9x_0 + 1$
		3	4,092	$9x_7 + 11x_0 + 7$
		3	4,092	$5x_3 + 11x_0 + 15$
		3	4,092	$x_3 + 13x_0 + 1$
		3	4,092	$15x_7 + 15x_0 + 5$
		3	4,092	$9x_7 + 11x_0 + 11$
		3	4,092	$13x_7 + 11x_0 + 7$
		3	4,092	$9x_7 + 3x_0 + 7$
11	4	3	2,047	$x_9 + 3x_0 + 3$
		3	2,047	$3x_9 + 3x_0 + 1$
		3	2,047	$x_9 + x_0 + 1$
		5	2,047	$3x_8 + 3x_4 + x_1 + x_0 + 3$
		5	2,047	$x_{10} + 3x_6 + 3x_5 + x_0 + 3$
		5	2,047	$3x_8 + x_4 + x_1 + x_0 + 1$
		5	2,047	$x_9 + 3x_4 + x_1 + 3x_0 + 3$
		5	2,047	$3x_7 + x_6 + x_4 + 3x_0 + 3$
		5	2,047	$3x_9 + 3x_8 + x_1 + 3x_0 + 1$
		5	2,047	$x_9 + x_4 + 3x_2 + 3x_0 + 3$
11	8	3	4,094	$x_9 + 5x_0 + 1$
		3	4,094	$7x_9 + 3x_0 + 1$
		3	4,094	$5x_2 + x_0 + 1$
		3	4,094	$x_2 + 7x_0 + 3$
		3	4,094	$x_9 + 3x_0 + 7$
		3	4,094	$5x_2 + 3x_0 + 7$
		3	4,094	$x_2 + 3x_0 + 3$
		3	4,094	$5x_2 + 3x_0 + 3$
		3	4,094	$x_2 + x_0 + 1$
		3	4,094	$x_9 + x_0 + 1$
11	16	3	8,188	$x_9 + 5x_0 + 13$
		3	8,188	$15x_2 + 9x_0 + 3$
		3	8,188	$5x_9 + 5x_0 + 5$
		3	8,188	$5x_2 + 9x_0 + 5$
		3	8,188	$15x_2 + 5x_0 + 3$
		3	8,188	$x_9 + 5x_0 + 1$
		3	8,188	$5x_2 + 5x_0 + 5$
		3	8,188	$15x_2 + 5x_0 + 7$

Affine multivariate polynomials, generated via a genetic algorithm

vars	D	Width	Cycle	Polynomial
		3	8,188	$x_9 + 5x_0 + 5$
		3	8,188	$13x_2 + 9x_0 + 5$
12	4	5	4,095	$3x_{10} + x_2 + 3x_1 + 3x_0 + 1$
		5	4,095	$x_{11} + 3x_{10} + 3x_2 + 3x_0 + 1$
		5	4,095	$x_{11} + 3x_{10} + x_2 + x_0 + 1$
		7	4,095	$3x_{11} + 3x_{10} + 3x_4 + x_2 + 3x_1 + x_0 + 1$
		7	4,095	$3x_{10} + 3x_9 + 3x_7 + x_5 + 3x_2 + x_0 + 1$
		7	4,095	$3x_{11} + x_{10} + 3x_7 + x_2 + 3x_1 + 3x_0 + 1$
		7	4,095	$3x_{10} + x_9 + x_7 + x_5 + 3x_2 + x_0 + 1$
		7	4,095	$3x_{10} + 3x_9 + x_7 + x_5 + 3x_2 + 3x_0 + 1$
		7	4,095	$x_{10} + 3x_9 + 3x_8 + 3x_7 + 3x_3 + x_0 + 1$
		7	4,095	$x_{11} + 3x_{10} + x_6 + x_3 + x_2 + 3x_0 + 1$
12	8	5	8,190	$3x_{10} + 3x_2 + 7x_1 + 5x_0 + 5$
		5	8,190	$3x_{10} + 3x_2 + 7x_1 + 5x_0 + 1$
		7	8,190	$x_9 + 3x_8 + 5x_3 + x_2 + 7x_1 + 5x_0 + 5$
		7	8,190	$3x_9 + x_5 + 7x_4 + x_3 + 7x_2 + x_0 + 7$
		7	8,190	$3x_9 + x_5 + 5x_4 + 3x_2 + x_1 + 3x_0 + 7$
		7	8,190	$x_9 + 5x_7 + x_6 + x_3 + 3x_1 + x_0 + 7$
		7	8,190	$x_9 + 5x_7 + x_6 + x_3 + 3x_1 + 5x_0 + 7$
		7	8,190	$7x_9 + 3x_5 + 3x_4 + 3x_2 + 5x_1 + 3x_0 + 3$
		7	8,190	$7x_{10} + 7x_5 + 3x_4 + x_2 + 3x_1 + 3x_0 + 3$
		7	8,190	$5x_9 + x_5 + x_4 + 3x_2 + 7x_1 + 5x_0 + 5$
12	16	5	16,380	$11x_{10} + 7x_2 + x_1 + x_0 + 15$
		5	16,380	$11x_{10} + 7x_2 + 5x_1 + 15x_0 + 13$
		5	16,380	$11x_{10} + 7x_2 + 13x_1 + 15x_0 + 13$
		5	16,380	$11x_{10} + 7x_2 + x_1 + 15x_0 + 13$
		7	16,380	$5x_{11} + 3x_{10} + 7x_7 + 11x_2 + 9x_1 + 7x_0 + 9$
		7	16,380	$x_{11} + 11x_{10} + 3x_5 + 5x_2 + 5x_1 + 7x_0 + 15$
		7	16,380	$7x_{11} + 11x_{10} + 3x_9 + 13x_4 + 7x_2 + x_0 + 13$
		7	16,380	$7x_{11} + 11x_{10} + 15x_9 + 13x_4 + 7x_2 + x_0 + 13$
		7	16,380	$11x_{11} + 9x_9 + x_5 + 9x_3 + 5x_1 + 15x_0 + 9$
		7	16,380	$7x_{11} + 11x_{10} + 9x_7 + 7x_2 + 5x_1 + 15x_0 + 13$
13	4	5	8,191	$3x_{12} + x_{10} + 3x_6 + 3x_0 + 1$
		5	8,191	$x_{10} + x_8 + 3x_5 + x_0 + 1$
		5	8,191	$x_{10} + x_9 + x_2 + 3x_0 + 1$
		5	8,191	$3x_{11} + 3x_8 + 3x_6 + x_0 + 1$
		5	8,191	$3x_{12} + 3x_2 + x_1 + x_0 + 3$
		5	8,191	$3x_6 + x_5 + 3x_2 + 3x_0 + 1$
		5	8,191	$x_6 + 3x_5 + 3x_2 + x_0 + 3$

Affine multivariate polynomials, generated via a genetic algorithm

vars	D	Width	Cycle	Polynomial
		5	8,191	$3x_{12} + x_{10} + x_9 + 3x_0 + 3$
		5	8,191	$3x_6 + 3x_5 + 3x_2 + x_0 + 1$
		5	8,191	$x_{11} + 3x_6 + 3x_2 + x_0 + 3$
13	8	5	16,382	$5x_{11} + 7x_{10} + x_5 + 3x_0 + 3$
		5	16,382	$7x_{10} + 3x_6 + x_5 + 7x_0 + 5$
		5	16,382	$x_{11} + 3x_6 + x_2 + 7x_0 + 7$
		5	16,382	$x_9 + x_6 + 7x_1 + 3x_0 + 7$
		5	16,382	$7x_{10} + 3x_4 + 5x_1 + 7x_0 + 5$
		5	16,382	$7x_{10} + 7x_9 + x_2 + 7x_0 + 5$
		5	16,382	$7x_{12} + x_4 + x_3 + 7x_0 + 3$
		5	16,382	$5x_{11} + 7x_{10} + x_5 + 7x_0 + 3$
		5	16,382	$5x_{11} + 7x_{10} + x_5 + 3x_0 + 7$
		5	16,382	$5x_{11} + 3x_9 + x_5 + 3x_0 + 3$
13	16	5	32,764	$5x_7 + x_6 + 7x_1 + 5x_0 + 13$
		5	32,764	$13x_6 + 11x_4 + 9x_1 + 5x_0 + 1$
		5	32,764	$7x_{12} + 11x_2 + x_1 + 11x_0 + 5$
		5	32,764	$11x_{10} + 9x_7 + x_6 + 5x_0 + 13$
		5	32,764	$15x_{10} + 11x_6 + 11x_4 + 5x_0 + 1$
		5	32,764	$13x_{10} + 13x_3 + x_1 + x_0 + 11$
		5	32,764	$5x_{12} + 3x_{10} + 7x_9 + x_0 + 11$
		5	32,764	$11x_{10} + 13x_7 + 13x_6 + x_0 + 9$
		5	32,764	$13x_5 + 11x_2 + x_1 + 15x_0 + 15$
		5	32,764	$15x_{10} + 9x_9 + x_2 + 5x_0 + 9$
14	4	5	16,383	$x_{11} + 3x_7 + 3x_1 + 3x_0 + 1$
		5	16,383	$3x_{11} + 3x_6 + x_1 + x_0 + 3$
		5	16,383	$3x_{11} + 3x_6 + 3x_1 + x_0 + 1$
		5	16,383	$3x_{12} + x_{11} + x_1 + x_0 + 1$
		5	16,383	$3x_{13} + 3x_8 + x_3 + 3x_0 + 1$
		5	16,383	$x_8 + 3x_6 + x_1 + x_0 + 1$
		5	16,383	$x_{11} + 3x_6 + x_1 + x_0 + 1$
		5	16,383	$3x_{13} + x_7 + 3x_3 + 3x_0 + 1$
		5	16,383	$x_{12} + x_{11} + 3x_1 + x_0 + 1$
		5	16,383	$x_5 + x_3 + 3x_1 + x_0 + 1$
14	8	5	32,766	$5x_{11} + x_6 + 3x_1 + x_0 + 1$
		5	32,766	$3x_{12} + 5x_{11} + 5x_5 + 7x_0 + 7$
		5	32,766	$3x_{10} + 3x_3 + 3x_1 + 3x_0 + 7$
		5	32,766	$5x_9 + x_8 + 3x_3 + 3x_0 + 7$
		5	32,766	$5x_9 + 5x_8 + 3x_3 + 3x_0 + 7$
		5	32,766	$5x_9 + x_8 + 3x_3 + 7x_0 + 7$

Affine multivariate polynomials, generated via a genetic algorithm

vars	D	Width	Cycle	Polynomial
		5	32,766	$x_9 + x_8 + 3x_3 + 3x_0 + 3$
		5	32,766	$3x_{13} + 7x_{10} + x_6 + x_0 + 7$
		5	32,766	$5x_{11} + x_6 + 5x_1 + x_0 + 3$
		5	32,766	$5x_9 + x_8 + 3x_3 + 3x_0 + 3$
14	16	5	65,532	$5x_5 + 7x_3 + x_1 + 7x_0 + 15$
		5	65,532	$5x_5 + 7x_3 + 11x_1 + 7x_0 + 1$
		5	65,532	$5x_5 + 7x_3 + 5x_1 + 7x_0 + 3$
		5	65,532	$5x_5 + 7x_3 + x_1 + 13x_0 + 1$
		5	65,532	$15x_5 + 5x_3 + x_1 + 7x_0 + 15$
		5	65,532	$5x_5 + 7x_3 + 5x_1 + 15x_0 + 3$
		5	65,532	$5x_5 + 7x_3 + 15x_1 + 13x_0 + 15$
		5	65,532	$5x_5 + 7x_3 + 5x_1 + 7x_0 + 7$
		5	65,532	$5x_5 + 7x_3 + 11x_1 + 13x_0 + 15$
		5	65,532	$5x_5 + 9x_3 + 11x_1 + 7x_0 + 15$
15	4	3	32,767	$x_8 + 3x_0 + 3$
		5	32,767	$x_{12} + x_8 + x_6 + 3x_0 + 1$
		5	32,767	$x_9 + x_6 + 3x_2 + 3x_0 + 3$
		5	32,767	$x_8 + 3x_5 + x_2 + 3x_0 + 3$
		5	32,767	$x_{12} + x_6 + x_1 + 3x_0 + 1$
		5	32,767	$x_{12} + x_6 + x_5 + 3x_0 + 1$
		5	32,767	$3x_5 + 3x_3 + x_2 + 3x_0 + 1$
		5	32,767	$x_{11} + x_8 + 3x_3 + 3x_0 + 3$
		5	32,767	$x_{12} + x_9 + x_1 + 3x_0 + 1$
		5	32,767	$x_{11} + x_8 + 3x_2 + 3x_0 + 3$
15	8	3	65,534	$3x_1 + 7x_0 + 1$
		3	65,534	$5x_4 + 5x_0 + 1$
		3	65,534	$5x_{14} + 3x_0 + 3$
		3	65,534	$3x_1 + 3x_0 + 1$
		3	65,534	$5x_{14} + 7x_0 + 3$
		3	65,534	$x_{11} + 5x_0 + 1$
		3	65,534	$3x_1 + 5x_0 + 3$
		3	65,534	$5x_4 + 3x_0 + 3$
		3	65,534	$3x_4 + 7x_0 + 1$
		3	65,534	$x_{11} + 3x_0 + 7$
15	16	3	131,068	$11x_1 + 13x_0 + 7$
		3	131,068	$11x_8 + 3x_0 + 9$
		3	131,068	$11x_1 + 13x_0 + 15$
		3	131,068	$11x_1 + 9x_0 + 7$
		3	131,068	$3x_1 + 9x_0 + 3$

Affine multivariate polynomials, generated via a genetic algorithm

vars	D	Width	Cycle	Polynomial
		3	131,068	$11x_8 + 13x_0 + 7$
		3	131,068	$11x_1 + x_0 + 7$
		3	131,068	$11x_8 + 3x_0 + 13$
		3	131,068	$11x_1 + 13x_0 + 11$
		3	131,068	$3x_{11} + 15x_0 + 13$
16	4	5	65,535	$3x_{15} + x_{13} + x_4 + 3x_0 + 3$
		5	65,535	$3x_{15} + x_4 + x_1 + 3x_0 + 3$
		7	65,535	$x_{15} + 3x_{13} + x_5 + x_3 + x_2 + 3x_0 + 1$
		7	65,535	$3x_{15} + x_{13} + 3x_{12} + x_4 + x_1 + 3x_0 + 3$
		7	65,535	$x_{14} + 3x_9 + x_6 + 3x_3 + x_1 + 3x_0 + 3$
		7	65,535	$3x_{13} + x_{12} + x_5 + 3x_2 + x_1 + 3x_0 + 3$
		7	65,535	$3x_{15} + 3x_{12} + x_5 + x_2 + x_1 + 3x_0 + 3$
		7	65,535	$3x_{15} + 3x_{14} + x_{13} + x_4 + x_1 + 3x_0 + 3$
		7	65,535	$x_{13} + 3x_{10} + 3x_9 + x_5 + x_1 + 3x_0 + 3$
		7	65,535	$x_{13} + 3x_9 + x_5 + 3x_2 + x_1 + 3x_0 + 3$
16	8	5	131,070	$5x_{15} + 3x_7 + 3x_2 + 3x_0 + 5$
		5	131,070	$5x_{15} + 7x_7 + 3x_2 + 3x_0 + 5$
		5	131,070	$5x_{15} + 7x_7 + 3x_2 + 3x_0 + 1$
		5	131,070	$7x_{14} + 7x_{11} + 7x_7 + 5x_0 + 1$
		5	131,070	$x_{11} + x_9 + 7x_7 + 5x_0 + 1$
		5	131,070	$5x_{15} + 3x_7 + 3x_2 + 5x_0 + 3$
		5	131,070	$7x_9 + 3x_7 + 3x_2 + 3x_0 + 7$
		5	131,070	$3x_{10} + 7x_7 + 3x_4 + 3x_0 + 3$
		5	131,070	$5x_{15} + 7x_7 + 3x_2 + 5x_0 + 3$
		5	131,070	$7x_9 + 7x_7 + 3x_2 + 3x_0 + 3$
16	16	5	262,140	$7x_{12} + 9x_3 + 7x_1 + 3x_0 + 13$
		5	262,140	$7x_{12} + 9x_3 + 7x_1 + 11x_0 + 13$
		5	262,140	$7x_{12} + 13x_3 + 7x_1 + 3x_0 + 13$
		5	262,140	$7x_{12} + 9x_3 + 7x_1 + 15x_0 + 13$
		5	262,140	$15x_{12} + 9x_3 + 15x_1 + 3x_0 + 9$
		5	262,140	$13x_{12} + 13x_9 + 11x_6 + 7x_0 + 3$
		5	262,140	$7x_{12} + 5x_7 + 7x_1 + 3x_0 + 13$
		5	262,140	$7x_{12} + x_{10} + 13x_3 + 3x_0 + 15$
		5	262,140	$7x_{12} + 9x_3 + 7x_1 + 7x_0 + 13$
		5	262,140	$7x_{12} + 13x_9 + 13x_6 + 3x_0 + 3$
17	4	3	131,071	$3x_{12} + 3x_0 + 1$
		3	131,071	$3x_{11} + 3x_0 + 1$
		3	131,071	$3x_{11} + x_0 + 3$
		3	131,071	$3x_{12} + x_0 + 3$

Affine multivariate polynomials, generated via a genetic algorithm

vars	D	Width	Cycle	Polynomial
		5	131,071	$3x_{11} + 3x_7 + x_4 + 3x_0 + 1$
		5	131,071	$3x_{12} + x_8 + 3x_2 + 3x_0 + 1$
		5	131,071	$3x_{12} + x_9 + 3x_2 + 3x_0 + 1$
		5	131,071	$3x_{15} + 3x_{12} + x_2 + x_0 + 3$
		5	131,071	$3x_{15} + 3x_{11} + 3x_8 + x_0 + 1$
		5	131,071	$x_{13} + 3x_{12} + 3x_2 + 3x_0 + 1$
17	8	3	262,142	$7x_{14} + 5x_0 + 7$
		3	262,142	$7x_{14} + x_0 + 3$
		3	262,142	$3x_5 + 7x_0 + 5$
		3	262,142	$7x_{14} + 5x_0 + 3$
		3	262,142	$x_3 + 5x_0 + 1$
		3	262,142	$7x_3 + 5x_0 + 3$
		3	262,142	$7x_{14} + 7x_0 + 1$
		3	262,142	$5x_{11} + 5x_0 + 1$
		3	262,142	$5x_{11} + 7x_0 + 3$
		3	262,142	$3x_5 + 5x_0 + 3$
17	16	3	524,284	$5x_3 + 5x_0 + 9$
		3	524,284	$5x_{11} + 9x_0 + 13$
		3	524,284	$11x_{12} + 5x_0 + 11$
		3	524,284	$15x_5 + x_0 + 7$
		3	524,284	$5x_3 + 5x_0 + 13$
		3	524,284	$13x_3 + 5x_0 + 13$
		3	524,284	$13x_5 + x_0 + 9$
		3	524,284	$11x_{12} + x_0 + 15$
		3	524,284	$11x_{12} + x_0 + 11$
		3	524,284	$11x_{12} + 5x_0 + 15$

D.3 NLFSR—HigherDegree Polynomials

1/2 chance auf quadratisch Was genau passiert hier?
Heading: “multivariate” auf jeder Seite

High degree multivariate polynomials, generated via a genetic algorithm

vars	Width	Cycle		Polynomial
4	4	3	15	$3x_0x_0x_1 + 3x_0 + 1$
		3	15	$x_1x_1x_3 + 3x_0x_1x_3 + 3$
		3	15	$x_0x_2x_3 + 3x_2 + 3$
		3	15	$x_0x_3x_3 + 3x_1 + 3$
		3	15	$3x_1 + 3x_0 + 1$
		3	15	$3x_0x_0x_1 + 3x_0 + 1$

High degree multivariate polynomials, generated via a genetic algorithm

vars	Width	Cycle	Polynomial	
		3	15	$3x_1 + 3x_0 + 1$
		3	15	$3x_0x_0x_1 + 3x_0 + 1$
		3	15	$3x_1 + 3x_0 + 1$
		3	15	$x_1x_1x_3 + 3x_0x_1x_3 + 3$
4	8	3	30	$3x_1x_2x_2 + 7x_0 + 5$
		3	30	$3x_1 + 5x_0 + 7$
		3	30	$x_0x_1x_3 + 5x_0x_0x_3 + 1$
		3	30	$7x_3 + 5x_0 + 7$
		3	30	$x_0x_1x_3 + 3x_3 + 7$
		3	30	$7x_3 + 5x_0 + 7$
		3	30	$7x_3 + x_0 + 3$
		3	30	$x_0x_3x_3 + 7x_1 + 7$
		3	30	$x_0x_1x_3 + 3x_3 + 3$
		3	30	$3x_3 + 3x_0 + 1$
4	16	3	60	$15x_3 + 3x_0 + 5$
		3	60	$11x_1 + 9x_0 + 11$
		3	60	$x_1 + 3x_0 + 11$
		3	60	$9x_0x_2x_2 + x_3 + 13$
		3	60	$7x_1x_1x_3 + 9x_0x_1x_3 + 11$
		3	60	$11x_0x_3x_3 + 13x_1 + 11$
		3	60	$15x_0x_1x_3 + 7x_1 + 1$
		3	60	$x_3 + 7x_0 + 7$
		3	60	$13x_3 + x_0 + 9$
		3	60	$15x_3 + 3x_0 + 9$
5	4	3	31	$x_1x_2 + 3x_0x_1 + 3$
		3	31	$3x_1x_2x_2x_3 + 3x_0x_2 + 1$
		3	31	$3x_0x_1x_3 + x_2 + 3$
		3	31	$3x_2 + x_0 + 3$
		3	31	$3x_0x_1x_2x_2 + x_2x_3 + 3$
		3	31	$3x_3 + 3x_0 + 1$
		3	31	$3x_2x_3x_3x_3 + 3x_0x_2x_3x_3 + 1$
		3	31	$3x_2x_3 + 3x_0x_1 + 1$
		3	31	$3x_3x_4 + x_0x_4 + 3$
		3	31	$3x_2x_3 + 3x_0x_1 + 1$
5	8	3	62	$3x_1x_2x_4 + 7x_0 + 5$
		3	62	$x_2x_3x_3 + 5x_0x_3x_3 + 5$
		3	62	$7x_1x_1x_4x_4 + 5x_0x_1x_2x_3 + 7$
		3	62	$5x_0x_3 + 7x_0x_0 + 3$
		3	62	$7x_0x_1x_2 + 7x_1 + 5$

High degree multivariate polynomials, generated via a genetic algorithm

vars	Width	Cycle	Polynomial	
		3	62	$3x_2 + 5x_0 + 3$
		3	62	$3x_3 + x_0 + 7$
		3	62	$3x_2x_2x_3 + 3x_0 + 1$
		3	62	$7x_0x_1x_2 + 7x_1 + 1$
		3	62	$x_3 + 5x_0 + 5$
5	16	3	124	$9x_0x_2x_3x_4 + 11x_1x_2 + 3$
		3	124	$5x_0x_2x_3 + 9x_4 + 9$
		3	124	$x_2 + 7x_0 + 3$
		3	124	$9x_0x_2x_3x_4 + 11x_1x_2 + 3$
		3	124	$13x_2 + 3x_0 + 11$
		3	124	$x_2 + 7x_0 + 3$
		3	124	$5x_0x_2x_3 + 9x_4 + 13$
		3	124	$x_2 + 7x_0 + 15$
		3	124	$x_2 + 7x_0 + 3$
		3	124	$9x_0x_2x_3x_4 + 11x_1x_2 + 3$
6	4	3	63	$3x_1x_3x_5 + 3x_0x_1x_3 + 1$
		3	63	$x_4x_4 + 3x_0x_1 + 3$
		3	63	$x_5 + x_0 + 1$
		3	63	$x_0x_2x_4x_4x_5 + 3x_2 + 3$
		3	63	$x_5 + x_0 + 1$
		3	63	$x_4x_4 + 3x_0x_1 + 3$
		3	63	$3x_1x_3x_5 + 3x_0x_1x_3 + 1$
		3	63	$x_5 + x_0 + 1$
		3	63	$x_0x_2x_4x_4x_5 + 3x_2 + 3$
		3	63	$3x_1x_3x_5 + 3x_0x_1x_3 + 1$
6	8	3	126	$3x_5 + 3x_0 + 5$
		3	126	$x_5 + 7x_0 + 3$
		3	126	$5x_1x_4 + 7x_0x_3 + 3$
		3	126	$3x_5 + 3x_0 + 5$
		3	126	$x_5 + 7x_0 + 3$
		3	126	$7x_5 + x_0 + 3$
		3	126	$x_5 + 7x_0 + 3$
		3	126	$7x_5 + x_0 + 3$
		3	126	$x_5 + 7x_0 + 3$
		3	126	$7x_5 + x_0 + 3$
6	16	3	252	$7x_0x_2x_3 + 11x_5 + 9$
		3	252	$3x_1 + 11x_0 + 13$
		3	252	$15x_0x_1x_4 + 7x_5 + 1$
		3	252	$3x_1 + 13x_0 + 11$

High degree multivariate polynomials, generated via a genetic algorithm

vars	Width	Cycle	Polynomial	
		3	252	$7x_0x_2x_3 + 11x_5 + 9$
		3	252	$13x_0x_1x_2x_3x_3 + 15x_2x_3x_4 + 15$
		3	252	$5x_0x_1x_3x_3x_4 + 3x_4 + 11$
		3	252	$7x_0x_2x_3 + 11x_5 + 9$
		3	252	$3x_1 + 13x_0 + 15$
		3	252	$7x_0x_2x_3 + 11x_5 + 9$
7	4	3	127	$x_0x_1x_1x_5x_6 + x_2 + 1$
		3	127	$x_0x_1x_3x_3 + 3x_1x_4 + 3$
		3	127	$x_3x_3x_4 + x_0 + 1$
		3	127	$3x_0x_2x_3x_4x_6 + 3x_6 + 1$
		3	127	$x_0x_1x_1x_5x_6 + x_2 + 1$
		3	127	$x_3x_3x_4 + x_0 + 1$
		3	127	$x_0x_1x_1x_5x_6 + x_2 + 1$
		3	127	$3x_0x_2x_3x_4x_6 + 3x_6 + 1$
		3	127	$x_3x_3x_4 + x_0 + 1$
		3	127	$x_0x_1x_1x_5x_6 + x_2 + 1$
7	8	3	254	$7x_0x_3x_3x_6x_6 + x_3 + 7$
		3	254	$5x_1x_1x_3x_4x_5x_5 + 5x_0x_2 + 1$
		3	254	$7x_3 + 7x_0 + 5$
		3	254	$5x_0x_2x_3x_4x_5x_6 + 3x_2x_2x_6x_6 + 3$
		3	254	$5x_3x_5 + 3x_0x_5 + 7$
		3	254	$5x_1x_1x_3x_3x_6 + x_0 + 1$
		3	254	$5x_4 + 5x_0 + 5$
		3	254	$5x_0x_6x_6 + 5x_3 + 1$
		3	254	$5x_1x_1x_3x_4x_5x_5 + 5x_0x_2 + 1$
		3	254	$7x_3 + 7x_0 + 5$
7	16	3	508	$9x_6 + 3x_0 + 11$
		3	508	$11x_0x_2x_5 + 15x_6 + 9$
		3	508	$13x_4 + 3x_0 + 15$
		3	508	$9x_6 + 3x_0 + 11$
		3	508	$11x_0x_2x_5 + 15x_6 + 9$
		3	508	$9x_6 + 15x_0 + 11$
		3	508	$13x_4 + 3x_0 + 15$
		3	508	$13x_6 + 7x_0 + 3$
		3	508	$11x_0x_2x_5 + 15x_6 + 5$
		3	508	$9x_6 + 3x_0 + 11$
8	4	5	255	$3x_1x_2x_2x_7 + 3x_0x_3x_7 + 3x_3x_5 + x_3 + 1$
		5	255	$x_0x_3x_3 + 3x_5 + x_3 + x_2 + 1$
		5	255	$x_1x_2x_3x_5x_7 + 3x_5 + 3x_3 + 3x_0 + 1$

High degree multivariate polynomials, generated via a genetic algorithm

vars	Width	Cycle	Polynomial	
		5	255	$x_0x_3x_3 + 3x_5 + x_3 + x_2 + 1$
		5	255	$3x_1x_2x_2x_7 + 3x_0x_3x_7 + 3x_3x_5 + x_3 + 1$
		5	255	$3x_0x_1x_2x_3x_3 + x_0x_4 + 3x_0x_1 + x_3 + 3$
		5	255	$x_0x_5x_6x_7 + x_1x_7 + 3x_3 + x_1 + 1$
		5	255	$x_0x_3x_3 + 3x_5 + x_3 + x_2 + 1$
		5	255	$3x_2x_6x_7 + 3x_2x_2x_4 + x_0x_3x_3 + 3x_5 + 1$
		5	255	$3x_1x_2x_2x_7 + 3x_0x_3x_7 + 3x_3x_5 + x_3 + 1$
8	8	3	510	$3x_2x_4 + 5x_0x_3 + 3$
		3	510	$3x_2x_4 + 5x_0x_3 + 7$
		3	510	$3x_2x_4 + 5x_0x_3 + 3$
		3	510	$3x_1x_5x_5x_5 + 5x_0x_6 + 3$
		5	510	$x_5 + 3x_3 + x_2 + 3x_0 + 7$
		5	510	$x_1x_2 + 3x_0x_1 + 5x_6 + x_5 + 5$
		5	510	$7x_5 + 3x_3 + x_2 + 5x_0 + 3$
		5	510	$x_5 + 3x_3 + x_2 + 3x_0 + 7$
		5	510	$3x_0x_2x_4x_5x_7 + x_2x_6x_6x_7 + 5x_1x_3x_7 + 7x_4x_7 + 7$
		5	510	$x_5 + 3x_3 + x_2 + 7x_0 + 7$
8		5	1,020	$7x_1x_5x_7 + 13x_3 + 5x_1 + x_0 + 9$
		5	1,020	$7x_1x_5x_7 + 13x_3 + 15x_1 + x_0 + 15$
		5	1,020	$3x_0x_0x_1x_1 + 15x_6x_7 + x_1x_6 + 11x_0x_2 + 5$
		5	1,020	$15x_1x_7x_7 + 7x_5 + 13x_3 + 7x_0 + 5$
		5	1,020	$x_0x_0x_4x_6x_6 + 13x_3 + 7x_2 + 11x_0 + 15$
		5	1,020	$7x_1x_5x_7 + 13x_3 + 5x_1 + 9x_0 + 13$
		5	1,020	$15x_6x_7 + 9x_4x_5 + 15x_1x_4 + 11x_0x_2 + 5$
		5	1,020	$15x_6x_7 + 11x_0x_2 + x_3 + 15x_1 + 9$
		5	1,020	$7x_1x_5x_7 + 13x_3 + x_1 + x_0 + 1$
		5	1,020	$7x_1x_5x_7 + 5x_3 + 5x_1 + 11x_0 + 15$
9	4	3	511	$3x_4 + x_0 + 3$
		3	511	$3x_5x_8 + 3x_0x_1 + 1$
		3	511	$3x_0x_2x_4x_5 + 3x_7x_8 + 1$
		3	511	$x_0x_1x_8 + x_5 + 1$
		3	511	$3x_5x_8 + 3x_0x_1 + 1$
		3	511	$3x_4 + x_0 + 3$
		3	511	$3x_5x_8 + 3x_0x_1 + 1$
		3	511	$3x_0x_2x_4x_5 + 3x_7x_8 + 1$
		3	511	$3x_4 + x_0 + 3$
		3	511	$x_0x_1x_8 + x_5 + 1$
9	8	3	1,022	$5x_1x_8 + 7x_5 + 3x_0$
		3	1,022	$7x_5x_7x_7 + 3x_0 + 5$

High degree multivariate polynomials, generated via a genetic algorithm

vars	Width	Cycle	Polynomial	
		5	1,022	$x_1x_1x_8 + x_1x_8 + 5x_1x_4 + x_0 + 7$
		5	1,022	$7x_5x_7x_7 + x_7 + 5x_2 + 3x_0 + 3$
		5	1,022	$5x_1x_1x_3x_4x_6x_7x_7 + 3x_1x_3x_4x_5x_5x_7 + 3x_0x_8 + 5x_2 + 7$
		5	1,022	$3x_1x_3x_4x_5x_5x_7 + 3x_1x_1x_4 + 5x_0x_3 + 5x_2 + 3$
		5	1,022	$5x_1x_1x_3x_4x_6x_7x_7 + 3x_1x_3x_4x_5x_5x_7 + 3x_0x_8 + 5x_2 + 3$
		5	1,022	$7x_5x_7x_7 + x_7 + 5x_2 + 3x_0 + 7$
		5	1,022	$5x_2x_3x_4 + 3x_6x_7 + 5x_0x_4 + 7x_8 + 3$
		5	1,022	$3x_8 + x_7 + 5x_2 + 3x_0 + 3$
9	16	3	2,044	$15x_5 + 7x_0 + 9$
		3	2,044	$5x_1x_5x_7 + 11x_0 + 7$
		3	2,044	$5x_1x_5x_7 + 7x_0 + 7$
		3	2,044	$5x_1x_5x_7 + 11x_0 + 11$
		3	2,044	$5x_1x_5x_7 + 11x_0 + 7$
		3	2,044	$15x_5 + 7x_0 + 9$
		3	2,044	$5x_1x_5x_7 + 11x_0 + 7$
		3	2,044	$5x_1x_5x_7 + 7x_0 + 7$
		3	2,044	$15x_5 + 7x_0 + 9$
		3	2,044	$5x_1x_5x_7 + 7x_0 + 7$
10	4	3	1,023	$x_0x_2x_2x_2x_5x_5x_7 + 3x_8 + 3$
		5	1,023	$x_0x_2x_2x_2x_5x_5x_7 + 3x_0x_1x_7 + x_0x_1x_3 + 3x_8 + 3$
		5	1,023	$x_3x_4x_9 + 3x_6 + x_1 + 3x_0 + 3$
		5	1,023	$x_2x_2x_6x_6x_9 + 3x_0x_5x_5x_8 + x_1x_6 + 3x_2 + 3$
		5	1,023	$x_0x_2x_2x_2x_5x_5x_7 + x_2x_5 + 3x_0x_0 + 3x_8 + 3$
		5	1,023	$x_0x_2x_2x_2x_5x_5x_7 + 3x_8x_8 + 3x_1x_3 + x_2 + 3$
		5	1,023	$x_1x_2x_4x_4x_6x_6x_9 + 3x_4x_6 + x_0x_8 + 3x_6 + 3$
		5	1,023	$x_0x_2x_2x_2x_5x_5x_7 + x_2x_5 + 3x_0x_0 + 3x_8 + 3$
		5	1,023	$x_0x_2x_2x_2x_5x_5x_7 + 3x_8x_8 + 3x_1x_3 + x_2 + 3$
		5	1,023	$x_3x_4x_9 + 3x_6 + x_1 + 3x_0 + 3$
10	8	3	2,046	$5x_0x_1x_1 + x_3 + 1$
		3	2,046	$5x_0x_1x_1 + x_3 + 5$
		5	2,046	$7x_0x_1x_5x_9x_9x_9 + x_2x_2x_8 + 3x_4x_9 + 5x_1 + 7$
		5	2,046	$x_0x_9x_9 + 3x_4x_7 + 5x_1x_6 + 5x_9 + 5$
		5	2,046	$7x_6x_6x_6x_7x_7x_9 + 3x_1x_4x_4x_4x_6 + x_0x_9x_9 + 5x_2x_5 + 7$
		5	2,046	$x_0x_9x_9 + 3x_4x_7 + 5x_1x_6 + 5x_9 + 5$
		5	2,046	$7x_6x_6x_6x_7x_7x_9 + 3x_1x_4x_4x_4x_6 + x_0x_9x_9 + 5x_2x_5 + 3$
		5	2,046	$x_0x_9x_9 + 3x_4x_7 + 5x_1x_6 + 5x_9 + 5$
		5	2,046	$3x_6x_7 + x_1x_2 + 7x_5 + 5x_0 + 7$
		5	2,046	$x_0x_9x_9 + 3x_4x_7 + 7x_1x_6 + 5x_9 + 3$
10	16	3	4,092	$13x_0x_4x_9 + 11x_1 + 11$

High degree multivariate polynomials, generated via a genetic algorithm

vars	Width	Cycle	Polynomial	
		3	4,092	$13x_0x_4x_9 + 7x_1 + 15$
		3	4,092	$13x_0x_4x_9 + 11x_1 + 15$
		5	4,092	$15x_8x_9 + x_0x_4 + 13x_7 + 7x_1 + 15$
		5	4,092	$11x_1x_3x_4x_6 + 15x_3x_9 + x_1x_2 + 9x_0x_6 + 3$
		5	4,092	$11x_8x_8 + 5x_2x_5 + 3x_6 + 13x_0 + 11$
		5	4,092	$11x_1x_5x_7x_8 + x_0x_4 + 9x_9 + 7x_1 + 7$
		5	4,092	$11x_1x_3x_4x_6 + 15x_3x_9 + x_1x_2 + 9x_0x_6 + 15$
		5	4,092	$11x_1x_3x_4x_6 + 15x_3x_9 + x_1x_2 + 9x_0x_6 + 3$
		5	4,092	$5x_1x_3 + 15x_0x_6 + 13x_9 + 13x_3 + 9$
11	4	3	2,047	$3x_0x_3x_{10} + x_1 + 3$
		5	2,047	$x_2x_6x_7 + 3x_3 + x_1 + 3x_0 + 3$
		5	2,047	$3x_0x_7x_{10} + 3x_0x_3x_{10} + 3x_0x_1x_9 + 3x_5 + 3$
		5	2,047	$x_1x_3x_9 + x_8 + 3x_6 + x_0 + 1$
		5	2,047	$x_1x_3x_9 + x_{10} + x_8 + 3x_0 + 1$
		5	2,047	$x_2x_6x_7 + 3x_3 + x_1 + 3x_0 + 3$
		5	2,047	$3x_8 + 3x_6 + x_3 + x_0 + 3$
		5	2,047	$x_1x_3x_9 + x_{10} + x_8 + 3x_0 + 1$
		5	2,047	$3x_8 + 3x_6 + x_3 + x_0 + 3$
		5	2,047	$3x_2x_5x_9 + 3x_0x_7x_{10} + 3x_{10} + 3x_5 + 3$
11	8	3	4,094	$3x_4x_5x_6 + 7x_0 + 1$
		3	4,094	$3x_2x_4x_4x_8 + x_0x_3 + 7$
		3	4,094	$7x_8x_8x_9 + x_0 + 3$
		3	4,094	$x_1x_3x_5x_6 + x_0x_4x_8x_8 + 1$
		3	4,094	$3x_4x_5x_6 + 7x_0 + 1$
		3	4,094	$7x_8x_8x_9 + x_0 + 3$
		3	4,094	$3x_4x_5x_6 + 7x_0 + 1$
		3	4,094	$7x_0x_2x_{10} + 3x_9 + 1$
		3	4,094	$3x_4x_5x_6 + 7x_0 + 1$
		5	4,094	$3x_4x_5x_6 + 7x_7 + 5x_6 + 7x_0 + 1$
11	16	3	8,188	$13x_4x_4x_4x_6x_{10} + 15x_0x_5x_{10} + 7$
		3	8,188	$9x_0x_1x_3x_5x_6x_6x_7 + 3x_1 + 11$
		3	8,188	$x_6x_8 + 7x_0x_{10} + 3$
		3	8,188	$9x_0x_1x_3x_5x_6x_6x_7 + 3x_1 + 3$
		3	8,188	$13x_4x_4x_4x_6x_{10} + 15x_0x_5x_{10} + 7$
		3	8,188	$x_6x_8 + 7x_0x_{10} + 7$
		3	8,188	$13x_4x_4x_4x_6x_{10} + 15x_0x_5x_{10} + 7$
		3	8,188	$x_6x_8 + 7x_0x_{10} + 11$
		3	8,188	$x_6x_8 + 7x_0x_{10} + 3$

High degree multivariate polynomials, generated via a genetic algorithm

vars	Width	Cycle	Polynomial	
		3	8,188	$13x_4x_4x_4x_6x_{10} + 15x_0x_5x_{10} + 7$
12	4	5	4,095	$3x_1x_2x_{11} + x_0x_5x_{10} + 3x_9x_{11} + x_4x_{10} + 3$
		5	4,095	$x_0x_0x_1x_3x_{11} + x_2x_{10} + x_0x_6 + 3x_7 + 1$
		5	4,095	$3x_1x_2x_{11} + x_0x_5x_{10} + 3x_9x_{11} + x_4x_{10} + 3$
		5	4,095	$x_0x_0x_1x_3x_{11} + x_2x_{10} + x_0x_6 + 3x_7 + 1$
		5	4,095	$3x_9x_{11} + 3x_0x_6 + x_8 + 3x_7 + 1$
		5	4,095	$x_0x_0x_1x_3x_{11} + x_2x_{10} + x_0x_6 + 3x_1 + 1$
		5	4,095	$3x_9x_{11} + 3x_0x_6 + x_8 + 3x_7 + 1$
		5	4,095	$3x_9x_{11} + 3x_0x_6 + 3x_{11} + 3x_7 + 3$
		5	4,095	$3x_9x_{11} + 3x_0x_6 + x_8 + 3x_7 + 1$
		5	4,095	$3x_1x_2x_{11} + x_0x_5x_{10} + 3x_9x_{11} + x_4x_{10} + 3$
12	8	3	8,190	$7x_0x_6x_8 + 3x_{11} + 5$
		5	8,190	$5x_1x_3x_5 + 7x_0x_4x_5 + 7x_8x_9 + x_1x_2 + 3$
		5	8,190	$5x_2x_4x_9 + x_2x_{11} + 5x_1x_3 + 3x_0 + 1$
		5	8,190	$5x_1x_3x_5 + 7x_0x_4x_5 + 7x_8x_9 + 7x_1x_2 + 1$
		5	8,190	$5x_1x_8x_{10} + x_3 + x_2 + 7x_0 + 1$
		5	8,190	$5x_1x_3x_5 + 7x_0x_4x_5 + 7x_8x_9 + 7x_1x_2 + 1$
		5	8,190	$x_7x_8x_8x_9 + 7x_1x_2x_9x_{11} + 5x_1x_8x_{10} + 7x_0 + 7$
		5	8,190	$x_0x_1x_3x_4x_8x_9x_9x_{10} + x_7x_8x_8x_9 + 3x_{11} + 3x_7 + 3$
		5	8,190	$5x_1x_3x_5 + 7x_0x_4x_5 + 7x_8x_9 + 7x_1x_2 + 1$
		5	8,190	$5x_2x_4x_9 + x_2x_{11} + 5x_1x_3 + 3x_0 + 1$
12	16	3	16,380	$9x_0x_5x_7x_8x_9 + x_3 + 9$
		5	16,380	$9x_0x_5x_7x_8x_9 + 5x_8x_9 + 5x_1x_6 + x_3 + 11$
		5	16,380	$9x_0x_5x_7x_8x_9 + 5x_8x_9 + 5x_1x_6 + x_3 + 7$
		5	16,380	$9x_0x_5x_7x_8x_9 + 5x_8x_9 + 5x_1x_6 + 9x_3 + 3$
		5	16,380	$9x_0x_5x_7x_8x_9 + 5x_8x_9 + 5x_1x_6 + x_3 + 11$
		5	16,380	$9x_0x_5x_7x_8x_9 + 5x_8x_9 + 5x_1x_6 + 9x_3 + 3$
		5	16,380	$9x_0x_5x_7x_8x_9 + 5x_8x_9 + 5x_1x_6 + x_3 + 3$
		5	16,380	$9x_0x_5x_7x_8x_9 + 5x_8x_9 + 5x_1x_6 + 9x_3 + 11$
		5	16,380	$9x_0x_5x_7x_8x_9 + 5x_8x_9 + 5x_1x_6 + 9x_3 + 3$
		5	16,380	$9x_0x_5x_7x_8x_9 + 5x_8x_9 + 5x_1x_6 + x_3 + 11$
13	4	5	8,191	$x_3x_5x_6x_7x_7x_8x_8x_{10} + 3x_0x_{12} + 3x_{11} + 3x_2 + 1$
		5	8,191	$x_4x_4x_7x_7x_{12} + 3x_0x_3x_{12} + 3x_8 + 3x_2 + 1$
		5	8,191	$3x_0x_5x_5x_6x_{10}x_{11} + x_4x_4x_7x_7x_{12} + 3x_1x_2x_4x_6 + 3x_8 + 1$
		5	8,191	$x_4x_4x_7x_7x_{12} + 3x_0x_3x_{12} + 3x_8 + 3x_2 + 1$
		5	8,191	$3x_0x_5x_5x_6x_{10}x_{11} + x_4x_4x_7x_7x_{12} + 3x_1x_2x_4x_6 + 3x_8 + 1$
		5	8,191	$x_1x_1x_3x_4x_4x_6 + 3x_5x_6x_8 + 3x_0x_{12} + 3x_7 + 1$
		5	8,191	$3x_0x_5x_5x_6x_{10}x_{11} + x_4x_4x_7x_7x_{12} + 3x_1x_2x_4x_6 + 3x_8 + 1$
		5	8,191	$x_4x_4x_7x_7x_{12} + 3x_0x_3x_{12} + 3x_8 + 3x_2 + 1$

High degree multivariate polynomials, generated via a genetic algorithm

vars	Width	Cycle	Polynomial	
		5	8,191	$3x_0x_5x_5x_6x_{10}x_{11} + x_4x_4x_7x_7x_{12} + 3x_1x_2x_4x_6 + 3x_8 + 1$
		5	8,191	$x_2x_2x_6x_9 + 3x_0x_{12} + 3x_3 + 3x_1 + 1$
13	8	3	16,382	$3x_6x_7x_8x_9x_9x_{11} + 5x_0x_1x_1x_6 + 7$
		3	16,382	$x_0x_{10}x_{11} + x_6 + 1$
		3	16,382	$3x_6x_7x_8x_9x_9x_{11} + 5x_0x_1x_1x_6 + 7$
		3	16,382	$x_0x_{10}x_{11} + x_6 + 1$
		3	16,382	$3x_6x_7x_8x_9x_9x_{11} + 5x_0x_1x_1x_6 + 7$
		3	16,382	$x_0x_{10}x_{11} + x_6 + 1$
		3	16,382	$3x_6x_7x_8x_9x_9x_{11} + 5x_0x_1x_1x_6 + 7$
		3	16,382	$x_0x_{10}x_{11} + x_6 + 1$
		3	16,382	$3x_6x_7x_8x_9x_9x_{11} + 5x_0x_1x_1x_6 + 7$
		3	16,382	$x_0x_{10}x_{11} + x_6 + 1$
13	16	3	32,764	$x_0x_3x_9x_{10} + 7x_5x_8 + 7$
		5	32,764	$5x_1x_2x_9 + 15x_4x_7 + 11x_0x_1 + x_1 + 3$
		5	32,764	$x_0x_3x_9x_{10} + 15x_1x_1x_{12} + 7x_5x_8 + x_3 + 7$
		5	32,764	$3x_2x_4x_9x_9x_9 + 5x_1x_7x_8x_{12} + 11x_0x_1 + 7x_6 + 5$
		5	32,764	$15x_0x_1x_2x_2 + 3x_1x_9 + 7x_{11} + 13x_8 + 1$
		5	32,764	$9x_1x_6x_8x_9x_{11} + 3x_0x_5x_7x_9 + 11x_2x_5x_{10} + 13x_6x_{11} + 11$
		5	32,764	$x_2x_6x_6x_9 + 13x_0x_{12} + 15x_{12} + 11x_8 + 11$
		5	32,764	$3x_2x_4x_9x_9x_9 + 5x_1x_7x_8x_{12} + 11x_0x_1 + 7x_6 + 5$
		5	32,764	$9x_1x_6x_8x_9x_{11} + 3x_0x_5x_7x_9 + 11x_2x_5x_{10} + 13x_{10}x_{11} + 15$
		5	32,764	$9x_1x_6x_8x_9x_{11} + 3x_0x_5x_7x_9 + 11x_2x_5x_{10} + 13x_6x_{11} + 11$
14	4	3	16,383	$x_0x_1x_5x_8x_{10}x_{12}x_{13} + x_7 + 1$
		3	16,383	$x_0x_3x_7x_9 + 3x_9x_{13} + 3$
		3	16,383	$x_0x_1x_5x_8x_{10}x_{12}x_{13} + x_7 + 1$
		3	16,383	$x_0x_3x_7x_9 + 3x_9x_{13} + 3$
		3	16,383	$x_0x_1x_5x_8x_{10}x_{12}x_{13} + x_7 + 1$
		3	16,383	$x_0x_3x_7x_9 + 3x_9x_{13} + 3$
		3	16,383	$x_0x_1x_5x_8x_{10}x_{12}x_{13} + x_7 + 1$
		3	16,383	$x_0x_3x_7x_9 + 3x_9x_{13} + 3$
		3	16,383	$x_0x_1x_5x_8x_{10}x_{12}x_{13} + x_7 + 1$
		3	16,383	$x_0x_3x_7x_9 + 3x_9x_{13} + 3$
14	8	3	32,766	$5x_5x_6 + 3x_0x_9 + 3$
		3	32,766	$5x_5x_6 + 3x_0x_9 + 7$
		3	32,766	$5x_5x_6 + 3x_0x_9 + 3$
		3	32,766	$5x_5x_6 + 3x_0x_9 + 7$
		3	32,766	$5x_5x_6 + 3x_0x_9 + 3$
		5	32,766	$5x_2x_7 + 5x_0x_9 + 7x_{12} + 3x_6 + 3$
		5	32,766	$5x_4x_5x_8x_8x_9x_{10}x_{11} + 5x_5x_6 + 3x_0x_9 + x_2 + 1$

High degree multivariate polynomials, generated via a genetic algorithm

vars	Width	Cycle	Polynomial	
		5	32,766	$3x_0x_5x_8x_{13} + x_3x_5x_7 + 7x_1x_5 + 5x_4 + 3$
		5	32,766	$7x_{12} + 3x_2 + 3x_1 + 3x_0 + 7$
		5	32,766	$5x_5x_6 + 3x_0x_9 + 5x_7 + 5x_1 + 5$
14	16	3	65,532	$11x_0x_5x_9 + 5x_6 + 3$
		5	65,532	$15x_1x_3x_8x_{13} + 3x_0x_5x_6 + x_3x_{11} + 9x_{10} + 11$
		5	65,532	$5x_0x_5x_7x_7x_{13} + 5x_5x_6x_6x_{11} + x_1x_6x_8x_{11} + 15x_{13} + 9$
		5	65,532	$x_2x_9x_9x_{13} + 13x_2x_3 + 9x_{10} + 11x_0 + 13$
		5	65,532	$x_2x_9x_9x_{13} + 15x_1x_3x_8x_{13} + x_1 + 11x_0 + 11$
		5	65,532	$15x_1x_3x_8x_{13} + 3x_0x_5x_6 + x_3x_{11} + 9x_{10} + 11$
		5	65,532	$15x_1x_3x_8x_{13} + 11x_0x_5x_9 + 9x_2x_7 + 5x_6 + 3$
		5	65,532	$15x_1x_3x_8x_{13} + 11x_0x_5x_9 + 9x_2x_7 + 5x_6 + 11$
		5	65,532	$15x_1x_3x_8x_{13} + 11x_0x_5x_9 + 9x_2x_7 + 5x_6 + 3$
		5	65,532	$15x_1x_3x_8x_{13} + 3x_0x_5x_6 + x_3x_{11} + 9x_{10} + 11$
15	4	3	32,767	$3x_0x_7x_{11} + x_7 + 3$
		3	32,767	$x_7 + x_0 + 1$
		3	32,767	$3x_0x_7x_{11} + x_7 + 3$
		3	32,767	$x_7 + x_0 + 1$
		3	32,767	$3x_0x_7x_{11} + x_7 + 3$
		3	32,767	$x_7 + x_0 + 1$
		3	32,767	$3x_0x_7x_{11} + x_7 + 3$
		3	32,767	$x_7 + x_0 + 1$
		3	32,767	$3x_0x_7x_{11} + x_7 + 3$
		3	32,767	$x_7 + x_0 + 1$
15	8	3	65,534	$x_7 + 3x_0 + 7$
		3	65,534	$3x_0x_1x_4x_4x_9x_{13} + 7x_4x_{12} + 5$
		3	65,534	$3x_{14} + 7x_0 + 1$
		3	65,534	$5x_1x_6x_{10}x_{12}x_{13} + 5x_0 + 5$
		3	65,534	$3x_0x_1x_4x_4x_9x_{13} + 7x_4x_{12} + 5$
		3	65,534	$x_7 + 3x_0 + 7$
		3	65,534	$3x_0x_1x_4x_4x_9x_{13} + 7x_4x_{12} + 1$
		3	65,534	$5x_1x_6x_{10}x_{12}x_{13} + 5x_0 + 1$
		3	65,534	$3x_{14} + 3x_0 + 1$
		3	65,534	$5x_1x_6x_{10}x_{12}x_{13} + 5x_0 + 1$
15	16	3	131,068	$13x_4x_{10} + 13x_0x_9 + 13$
		3	131,068	$5x_{11} + 5x_0 + 1$
		3	131,068	$5x_{11} + 5x_0 + 5$
		3	131,068	$13x_4x_{10} + 13x_0x_9 + 13$
		3	131,068	$13x_4x_{10} + 13x_0x_9 + 5$
		3	131,068	$13x_4x_{10} + 13x_0x_9 + 9$

High degree multivariate polynomials, generated via a genetic algorithm

vars	Width	Cycle	Polynomial	
		3	131,068	$5x_{11} + 5x_0 + 1$
		3	131,068	$13x_4x_{10} + 13x_0x_9 + 1$
		3	131,068	$5x_{11} + 5x_0 + 5$
		3	131,068	$13x_4x_{10} + 15x_0x_9 + 11$
16	4	3	65,535	$3x_0x_2x_6x_7x_{12} + 3x_6 + 1$
		3	65,535	$x_0x_2x_8x_9 + 3x_6x_{13} + 3$
		3	65,535	$3x_0x_2x_6x_7x_{12} + 3x_6 + 1$
		3	65,535	$x_0x_2x_8x_9 + 3x_6x_{13} + 3$
		3	65,535	$3x_0x_2x_6x_7x_{12} + 3x_6 + 1$
		3	65,535	$x_0x_2x_8x_9 + 3x_6x_{13} + 3$
		3	65,535	$3x_0x_2x_6x_7x_{12} + 3x_6 + 1$
		3	65,535	$x_0x_2x_8x_9 + 3x_6x_{13} + 3$
		3	65,535	$3x_0x_2x_6x_7x_{12} + 3x_6 + 1$
		3	65,535	$x_0x_2x_8x_9 + 3x_6x_{13} + 3$
16	8	5	131,070	$7x_2x_4x_5x_{13}x_{15} + 3x_6x_9x_9x_{14} + 3x_4x_5 + x_0 + 5$
		5	131,070	$3x_1x_6x_{13}x_{14} + 3x_4x_5 + 7x_2 + x_0 + 5$
		5	131,070	$3x_1x_6x_{13}x_{14} + 3x_4x_5 + 5x_2 + x_0 + 7$
		5	131,070	$3x_1x_6x_{13}x_{14} + 3x_4x_5 + 7x_2 + x_0 + 5$
		5	131,070	$7x_2x_4x_5x_{13}x_{15} + 3x_6x_9x_9x_{14} + 3x_4x_5 + x_0 + 5$
		5	131,070	$3x_1x_6x_{13}x_{14} + 3x_4x_5 + 7x_2 + x_0 + 1$
		5	131,070	$3x_1x_6x_{13}x_{14} + 3x_4x_5 + 7x_2 + x_0 + 5$
		5	131,070	$7x_2x_4x_5x_{13}x_{15} + 3x_6x_9x_9x_{14} + 3x_4x_5 + x_0 + 5$
		5	131,070	$3x_1x_6x_{13}x_{14} + 3x_4x_5 + 7x_2 + x_0 + 5$
		5	131,070	$7x_2x_4x_5x_{13}x_{15} + 3x_6x_9x_9x_{14} + 3x_4x_5 + x_0 + 5$
16	16	3	262,140	$15x_2x_3x_5x_6x_{14} + 5x_0x_3x_4x_6x_{11} + 3$
		3	262,140	$15x_2x_3x_5x_6x_{14} + 5x_0x_3x_4x_6x_{11} + 7$
		3	262,140	$15x_2x_3x_5x_6x_{14} + 5x_0x_3x_4x_6x_{11} + 3$
		3	262,140	$15x_2x_3x_5x_6x_{14} + 5x_0x_3x_4x_6x_{11} + 7$
		3	262,140	$15x_2x_3x_5x_6x_{14} + 5x_0x_3x_4x_6x_{11} + 3$
		3	262,140	$15x_2x_3x_5x_6x_{14} + 5x_0x_3x_4x_6x_{11} + 7$
		3	262,140	$15x_2x_3x_5x_6x_{14} + 5x_0x_3x_4x_6x_{11} + 3$
		3	262,140	$15x_2x_3x_5x_6x_{14} + 5x_0x_3x_4x_6x_{11} + 7$
		5	262,140	$13x_5x_7x_{12} + 3x_0x_9x_{15} + x_2x_{13} + 15x_1x_5 + 15$
		5	262,140	$3x_0x_9x_{15} + 13x_7x_{14} + x_2x_{13} + 13x_3 + 1$
17	4	3	131,071	$3x_0x_7x_{10}x_{16} + 3x_{10}x_{15} + 1$
		5	131,071	$3x_1x_4x_{10}x_{10} + x_0x_{11} + 3x_{13} + 3x_7 + 1$
		5	131,071	$x_1x_5x_7x_8x_{12}x_{14} + x_0x_{15} + x_{14} + 3x_5 + 1$
		5	131,071	$x_1x_5x_7x_8x_{12}x_{14} + x_0x_{15} + 3x_9 + x_3 + 1$
		5	131,071	$3x_2x_{14} + x_0x_{11} + x_{12} + 3x_5 + 3$

High degree multivariate polynomials, generated via a genetic algorithm

vars	Width	Cycle	Polynomial	
		5	131,071	$x_1x_5x_7x_8x_{12}x_{14} + x_0x_{15} + 3x_9 + x_3 + 1$
		5	131,071	$3x_2x_{14} + x_0x_{11} + x_4 + 3x_1 + 3$
		5	131,071	$3x_0x_3x_{11} + 3x_7x_{10} + 3x_2x_{14} + x_2 + 1$
		5	131,071	$x_1x_5x_7x_8x_{12}x_{14} + x_0x_{15} + x_{14} + 3x_5 + 1$
		5	131,071	$x_1x_7x_{15} + 3x_1x_7 + x_0x_{15} + x_3 + 1$
17	8	3	262,142	$5x_{14} + 5x_0 + 5$
		3	262,142	$3x_0x_{12}x_{14} + 5x_{11} + 7$
		3	262,142	$5x_{14} + 5x_0 + 5$
		3	262,142	$3x_0x_{12}x_{14} + 5x_{11} + 7$
		3	262,142	$5x_{14} + 5x_0 + 5$
		3	262,142	$3x_0x_{12}x_{14} + 5x_{11} + 7$
		3	262,142	$3x_0x_{12}x_{14} + 5x_{11} + 3$
		3	262,142	$3x_0x_{12}x_{14} + 5x_{11} + 7$
		3	262,142	$3x_0x_{12}x_{14} + 5x_{11} + 3$
		3	262,142	$3x_0x_{12}x_{14} + 5x_{11} + 7$
17	16	3	524,284	$15x_0x_2x_4x_{13}x_{14} + 3x_{15} + 9$
		3	524,284	$x_{14} + x_0 + 9$
		3	524,284	$x_{14} + x_0 + 5$
		3	524,284	$x_{14} + x_0 + 9$
		3	524,284	$15x_0x_2x_4x_{13}x_{14} + 3x_{15} + 5$
		3	524,284	$15x_0x_2x_4x_{13}x_{14} + 3x_{15} + 9$
		3	524,284	$15x_0x_2x_4x_{13}x_{14} + 3x_{15} + 5$
		3	524,284	$x_{14} + 9x_0 + 9$
		3	524,284	$15x_0x_2x_4x_{13}x_{14} + 3x_{15} + 5$
		3	524,284	$15x_0x_2x_4x_{13}x_{14} + 3x_{15} + 9$

E How to write dịlƎ↶$Field$

We outline how to write the different structures in LaTeX.

We start with the definition of dịlƎ↶$Field$.

```
\newcommand{\FlipField}{\rotatebox[origin=c]{180}{Flip}${\curvearrowleft}{Field}$}
```

We also have dịlƎ$Int\dot{s}$.

```
\newcommand{\FlipInts}{\rotatebox[origin=c]{180}{Flip}\emph{In$\dot{t}$s}}
```

Plus dịlƎ$\breve{P}olys$.

```
\newcommand{\FlipPolys}{\rotatebox[origin=c]{180}{Flip}\emph{\u{P}olys}}
```

We also have **uədO P̆roblem**.

```
\textbf{\rotatebox[origin=c]{180}{Open}~\u{P}roblem}}
```

For historical reasons, we give the now deprecated dịlƎ$\overset{\curvearrowleft}{Field}$ and the corresponding command.

```
\newcommand{\FlipField}{\rotatebox[origin=c]{180}{Flip}$\overset{\curvearrowleft}{Field}$}
```

F Change Log

Version 1.15, 2026-09-19. 126 pages

- Added Sect. 4.4 SubstSolver.
- Added Sect. 4.10 with several lemmata.
- Added Sect. 5 on security reductions.
- Added Sect. 6.4.1 on Inner⊙dị⅃↶Fields and Outer♉dị⅃↶Fields.
- Moved the section Generalized LU Decomposition (App. B.2), Bespoke LU Decomposition (App. A.2), and Empirics on Solving Affine Equations $Ax = b$ (App. A.3) to the appendix.
- Small rewrite, correction of typos in Sect. 4 Mathematical Algorithms.

Version 1.10, 2026-08-10. 107 pages. (ePrint version number 20260810:052617)

- Finished the section on Bespoke LU Decomposition Sect. A.2 and clarified the sections above.
- Added a section on Trimmed LU, see Sect. 4.5.
- Added section to symbolic computation, see Sect. 7.3.
- Fixed some smaller errors.

Version 1.07, 2026-07-03. 105 pages. (ePrint version number 20260703:185234)

- Adding the App. A.3 and App. B.3 to deeper investigate solving affine equations.
- Clarified the treatment of Quaternions in Sect. 6.5.

Version 1.05, 2026-06-26. 94 pages. (ePrint version number 20260626:115650)

- Added ePrint paper number and URL.
- Tightened up Sect. 2.3 to remove ambiguity.
- Added Sect. 3.4.6 on the Hidden Subgroup Problem.
- Added Sect. 6.5 on complex numbers and Quaternions.
- Added App. E on how to write dị⅃↶*Field*s.
- Correcting minor spelling mistakes.

Version 0.99, 2026-05-28. 92 pages. (ePrint version number 20260528:210421)

- Initial version