Certain cryptographic systems based on an algebraic structure

Maximilian Duda, mxdu0407@gmail.com,
Marta Hanc, marta.hanc@student.ukw.edu.pl
Sebastian Kowalski, kspterna@student.ukw.edu.pl
Łukasz Matysiak, lukmat@ukw.edu.pl$^{(*)}$,
Martin Waldoch, martin.waldoch@student.ukw.edu.pl
Kazimierz Wielki University
Bydgoszcz, Poland

January 26, 2022

Abstract

In this paper we consider certaine cryptographic systems based on an Dedekind structure and Galois structure. We supplemented the created cryptosystems based on the Dedekind structure with programs written in C ++. Also we discuss about the inner structure of Galois in cryptography. It is widely recognized that such a structure is based only on finite fields. Our results reveals something more internal. The final section is a supplement information about square-free and radical factorizations in monoids consisting in searching for a minimal list of counterexamples. As an open problem, we leave creating a program that would generate such a list and how to use such a list to create a cryptosystem.

1 Introduction

In this paper we present developed cryptographic programs written in C ++ (see Sections 2 and 3). These cryptographic systems are based on the

Keywords: monoid, cryptography, Dedekind domain, Galois extension, factorization.
2010 Mathematics Subject Classification: Primary 94A60, Secondary 11R32.
$^{(*)}$ Corresponding author

Dedekind structure known in mathematics [2], [3]. This is the motivation coming from the reviewer of the paper [1], and from paper [2].

Dedekind domains are one of the most important rings in algebra. It has many valuable properties and has many uses. In such ring every nonzero proper ideal factors into primes and every non-zero fractional ideal of that ring is invertible.

In Section 2 we introduce a cryptographic programs, where the key is analog to the fractional ideal. In Section 3 we have a cryptographic programs, where an alphabet is analog to the fractional ideal.

In section 4 we have complementary of [1] and [2]. We presented the application of polynomial composites and monoid domains in cryptology in the form of certain cryptosystems. In this paper we present a program about this.

In section 5 we have a cryptosystem based on a certain Galois extension. Let's recall a Galois extension is an algebraic field extension $K \subset L$ that is normal and separably. We introduce an example of such cryptosystem using $\mathbb{Q} \subset \mathbb{Q}(\sqrt{2}, \sqrt{3})$ which is a Galois extension. This cryptosystem can be freely modified while maintaining the idea of its operation. The motivation here was the creation of a cryptosystem based on the Galois theory. They do exist, of course, but there is only talk of finite fields. To our knowledge, there are no cryptosystems that go deeper into this science so far.

By a monoid we mean a commutative cancellative monoid. In section 6 we have a minimal list of possible counterexamples to find in monoids. In [4] Section 4 we have 24 square-free and radical factorizations and all dependencies in general monoids and in particular monoids (GCD-, pre-Schreier-, SR-, ACCP-, atomic, factorial monoids).

Recall that a monoid is called GCD-monoid, if for any two elements there exists a greatest common divisor. A monoid H is called a pre-Schreier monoid, if any element $a \in H$ is primal, i.e. for any b, $c \in H$ such that $a \mid bc$ there exist a_1, $a_2 \in H$ such that $a = a_1 a_2$, $a_1 \mid b$ and $a_2 \mid c$. A monoid H is called SR-monoid, if every square-free element is radical (This definition comes from [4]). A monoid H is called ACCP-monoid any ascending sequence principal ideals of H stabilizes, i.e. for all sequence of principal ideals $I_1 \subset I_2 \subset \ldots$ there exists $n \in \mathbb{N}$ such that $I_n = I_{n+1} = \ldots$. A monoid H is called atomic, if every non-invertible element $a \in H$ be a finite product of irreducibles (atoms). A monoid H is factorial, if each non-invertible element can be written as a product of irreducible elements and this representation is unique.

Recall that every factorial monoid is ACCP-monoid, and ACCP-monoid is atomic. And recall that every factorial monoid is GCD-monoid is pre-Schreier. Since every pre-Schreier is AP-monoid (in such monoid an irreducible element (atom) is prime), then every atomic and AP-monoid is factorial.

In section 6 of this paper we consider a minimal list of counterexamples that we can look for. Some of them are at [4] Section 7.

2 The first cryptosystem with a Dedekind structure

Let us recall from [2] Section 2 the first cryptosystem mentioned.

Let $A = \{a_0, a_1, \dots, a_n\}$ be an alphabet such that $|A|$ be a prime number. Let $x \in \{2, 3, \dots, |A|\}$ be the value of one of the letters of the alphabet, $k \geqslant 2$ be an key. Then

$$y = xk \pmod{|A|},$$

where y be the value of one of the letters of the alphabet be an encrypted letter.

Now, assume we have encrypted letter y. Then we get a decrypted letter x by a formula

$$x = (y + (k - d) \cdot |A|) \cdot k^{-1},$$

where d be the remainder of dividing y by k.

The proof of the above formula is in [2], Section 2.

Below we present the algorithm in C++, it has a predefined 28-character alphabet. We encourage to modify the algorithm.

```
#include <iostream>

using namespace std;

char pub_alphabet[29], message[100]={}, test=message[100];

int x[29], p=29,noc,k;

int calc_number_of_charactes()
```

```
{
noc=0;
for(int i=0; i<100; i++)
{
if(message[i]!=test)
noc++;
}
return noc;
}

int search_ch(char a)
{
char test_key='A';
for(int i=0; i<p; i++)
{
if(a==test_key)
return i+2;
else
{
test_key++;
}
}
}

int encrypt(int m)
{
return (m*k)%p;
}


int main()
{
pub_alphabet[0]='A';
x[0]=2;

for(int i=1; i<=28; i++)
{
x[i]=2+i;
pub_alphabet[i]='A'+i;
}
pub_alphabet[28]=' ';
do
{
cout << "Enter a key(must be equal or greater than 2)" <<
endl;
cin >> k;
}while(k<2);

cout << "Write a message to encrypt(max 100 characters)" <<
endl;
cin >> message;

calc_number_of_charactes();

for(int i=0; i<noc; i++)
{
message[i]=toupper(message[i]);
cout << encrypt(search_ch(message[i])) << " ";
}

return 0;
}
```

3 The second cryptosystem with a Dedekind structure

Let us recall from [2], Section 3 the second cryptosystem mentioned.

Let A be a set of characters. Assume $|A|$ is equal to any prime number.

Secretly establish a second alphabet A' such that $A' \subset A$ with a prime length.

Let $m_1m_2m_3\ldots m_n$ be a message, we want to encrypt.

A secret short alphabet A' divides a large public alphabet into zones. We skip the extra characters such that 0, 1. So we have a clean alphabet from 2. Let's move one over, so we have 1. Suppose $p = |A|$, $q = |A'|$. We have $\lceil \frac{p}{q} \rceil$ zones. Zero zone, includes the alphabet from 1 to q. The first zone, i.e. the alphabet from $q+1$ to $2q$ and so on. The last zone $(\lceil \frac{p}{q} \rceil - 1)$ includes the alphabet from $\lceil \frac{p}{q} \rceil q$ to p.

Let's extend the message values with random numbers informing us about a given zone of a given letter (this information denote by z_i):

$$z_1 m_1 z_2 m_2 \dots z_n m_n$$

Denote by k the key. Multiply each value of the message (not the information about the zone) by k and use the modulo q.

Hence ciphertext is:

$$z_1 d_1 z_2 d_2 \dots z_n d_n,$$

where $d_1 d_2 \dots d_n$ be a encrypted message.

Now let's decode the message.

$$z_1 d_1 z_2 d_3 \dots z_n d_n$$

by dividing it into blocks (each block contains a zone and a message).

Let's apply the formula:

$$m_i = \frac{d_i + (z_i + t_i \cdot k)|A|}{k},$$

where m_i is the decoded letter, d_i encrypted letter, z is a number satisfies a congruence $|A|^{-1} z_i \equiv d_i \pmod{k}$, k be the key, t be a zone.

Below we present an algorithm in C ++. This program is limited to 100 characters. We leave it open as to how this program can be improved and also encourage to modify the algorithm.

```
#include <iostream>

using namespace std;

char pub_alphabet[29], priv_alphabet[3]={'A','B','C'},
message[100]={}, test=message[100];

int x[29], p=29, q=3,noc,k;

int calc_number_of_charactes()
{
noc=0;
for(int i=0; i<100; i++)
{
```

```
if(message[i]!=test)
noc++;
}
return noc;
}

int zones(int x)
{
int y=0;
while(x>=q)
{
y++;
x-=q;
}
return y;
}

int search_ch(char a)
{
char test_key='A';
for(int i=0; i<p; i++)
{
if(a==test_key)
return i;
else
{
test_key++;
}
}
}

int encrypt(int m)
{
return (m*k)%q;
}

bool key_check()
{
if(k%3!=0)
return true;
```

```
else
return false;
}

int main()
{
pub_alphabet[0]='A';
x[0]=2;

for(int i=1; i<=28; i++)
{
x[i]=2+i;
pub_alphabet[i]='A'+i;

}
pub_alphabet[28]=' ';

do
{
cout << "Enter a key (must not be divisible by 3)" << endl;
cin >> k;
}while(key_check()==false);

cout << "Write a message to encrypt(max 100 characters)" <<
endl;
cin >> message;

calc_number_of_charactes();

for(int i=0; i<noc; i++)
{
message[i]=toupper(message[i]);
cout <<  zones(search_ch(message[i])) <<
encrypt(search_ch(message[i])) << " ";
}

return 0;
}
```

4 Cryptosystem based on monoid domains.

Let us recall from [2], Section 5:

Recall that if F be a field and M be a submonoid of $\mathbb{Q}_+$ then we can construct a monoid domain:

$$F[M] = F[X;M] = \{a_0X^{m_0} + \cdots + a_nX^{m_n} \mid a_i \in F, m_i \in M\}.$$

Any alphabet of characters creates a finite set. Most ciphers are based on finite sets. But we can have the idea of using the infinite alphabet $\mathbb{A}$, although in reality they can be cyclical sets with an index that would mean a given cycle. For example, A_0 - 0, B_0 - 1, ... , Z_0 - 25, A_1 - 0, B_1 - 1, ... , where A_i=A, ..., Z_i=Z for $i = 0, 1, \ldots$. We see that this is isomorphic to a monoid $\mathbb{N}_0$ non-negative integers by a formula

$$f\colon \mathbb{A} \to \mathbb{N}, f(m_i) = i.$$

Then we can use a monoid domain by a map

$$\varphi\colon \mathbb{A} \to F[\mathbb{A}], \varphi(m_0, m_1, \ldots, m_n) = a_0X^{m_0} + \ldots a_nX^{m_n}.$$

We want to encrypt the message $m_0m_1m_2 \ldots m_n$ (the letters transform to numbers by a function φ). We establish the secret key X. Let F be a field. We determine any coefficients from this field: a_0, a_1, ..., a_n. Then the message $m_0m_1m_2 \ldots m_n$ be transformed into a polynomial of the form:

$$a_0X^{m_0} + a_1X^{m_1} + \cdots + a_nX^{m_n}.$$

We compute for $i = 0, 1, \ldots, n$: $d_i = a_iX^{m_i} \pmod{|\mathbb{A}|}$ ($|\mathbb{A}|$ must be prime) and then we have a decrypt message $d_0d_1 \ldots d_n$.

To decrypt it we need to use a formula (for $i = 0, 1, \ldots, n$):

$$m_i = \log_X \frac{d_i}{a_i} \pmod{|\mathbb{A}|}.$$

The following program is a modification of the presented mathematical algorithm. The user enters the key, the program creates coefficients which are successive powers of 2 modulo 10.

```
#include <iostream>
#include <math.h>

using namespace std;

char message[100]={}, test=message[100];

int p=29,noc,k;

struct polynomial
{
char a;
int x;
};

int calc_number_of_charactes()
{
noc=0;
for(int i=0; i<100; i++)
{
if(message[i]!=test)
noc++;
}
return noc;
}

int search_ch(char a)
{
char test_key='A';
for(int i=0; i<p; i++)
{
if(a==test_key)
return i;
else
{
test_key++;
}
}
}

int encrypt(int m, int x)
```

```
{
int en;
en=x*pow(k,m);
return en%p;
}

int main()
{
polynomial table[29];
table[0].a='A';

for(int i=1; i<=28; i++)
{
table[i].x=2;
table[i].x=pow(table[i].x,i);
table[i].x%=10;
table[i].a='A'+i;
}
table[28].a=' ';

cout << "Enter a key" << endl;
cin >> k;

cout << "Write a message to encrypt(max 100 characters)" << endl;
cin >> message;

calc_number_of_charactes();

for(int i=0; i<noc; i++)
{
message[i]=toupper(message[i]);
cout << encrypt(search_ch(message[i]),
table[search_ch(message[i])].x) << " ";
}
cout << endl;
int de;

return 0;
}
```

5 A cryptosystem based on a certain Galois extension

Let L be a Galois extension field of $\mathbb{Q}$. Recall, if $K \subset L$ is a Galois extension, then $Aut_K L$ is called the Galois group of $K \subset L$ and denoted by $G(L \mid K)$. It is well known how to form a Galois group of such an extension. We show a simple example.

Let $L = \mathbb{Q}(\sqrt{2}, \sqrt{3})$. Of course L is a Galois extension field of $\mathbb{Q}$. Then $Gal(L \mid \mathbb{Q}) = \{\sigma_1, \sigma_2, \sigma_3, \sigma_4\}$, where
$\sigma_1(a + b\sqrt{2} + c\sqrt{3} + d\sqrt{6}) = a + b\sqrt{2} + c\sqrt{3} + d\sqrt{6} = id$,
$\sigma_2(a + b\sqrt{2} + c\sqrt{3} + d\sqrt{6}) = a + b\sqrt{2} - c\sqrt{3} - d\sqrt{6}$,
$\sigma_3(a + b\sqrt{2} + c\sqrt{3} + d\sqrt{6}) = a - b\sqrt{2} + c\sqrt{3} - d\sqrt{6}$,
$\sigma_4(a + b\sqrt{2} + c\sqrt{3} + d\sqrt{6}) = a - b\sqrt{2} - c\sqrt{3} + d\sqrt{6}$,
with a, b, c, $d \in \mathbb{Q}$.

Automorphisms $\sigma_1, \sigma_2, \sigma_3, \sigma_4$ will be conveniently written as:
$\sigma_1(a, b, c, d) = (a, b, c, d) = id$,
$\sigma_2(a, b, c, d) = (a, b, -c, -d)$,
$\sigma_3(a, b, c, d) = (a, -b, c, -d)$,
$\sigma_4(a, b, c, d) = (a, -b, -c, d)$.

Consider an english alphabet which has 26 letters. Let $m_1m_2m_3m_4$ be a 4-letter block of a certain message.

In Galois group we ignore id because it doesn't change anything. Let's encode the block with the automorphism σ_2:

$$\sigma_2(m_1, m_2, m_3, m_4) = (m_1, m_2, -m_3, -m_4).$$

Then we switch $-m_3$ into the letter that has the opposite possition in the alphabet, for example D is the 4th letter of the alphabet, so $-D$ we switch into W because W is the 4th to last letter of the alphabet. Similarly we do with $-m_4$.

If a letter such m_1 doesn't change as a result we check what place in the alphabet does it take up and we divide by 2, and switch with the letter that has that position. But, if the remainder of dividing this letter's position by 2 equals 1, then we round down and instead of placing a capital letter, we place a small one. Similarly m_2. The table below may help with the operation.

Place	Letter	Negative letter	The letter it turns into if it doesn't change
1	A	Z	z
2	B	Y	A
3	C	X	a
4	D	W	B
5	E	V	b
6	F	U	C
7	G	T	c
8	H	S	D
9	I	R	d
10	J	Q	E
11	K	P	e
12	L	O	F
13	M	N	f
14	N	M	G
15	O	L	g
16	P	K	H
17	Q	J	h
18	R	I	I
19	S	H	i
20	T	G	J
21	U	F	j
22	V	E	K
23	W	D	k
24	X	C	L
25	Y	B	l
26	Z	A	M

We encode very similarly with the other automorphisms.

Using the table above and the Galois group, we can decode the encoded message without any problems.

In a very similar way, we can create an analogous cryptosystem, where the key will be an extension field of rational numbers, and if the addressee knows the Galois theory, he will easily calculate the Galois group of such extension and perform the appropriate steps.

From [5] Theorem 2.2. we know that every finite group is a Galois group of certain extension field of $\mathbb{Q}$. This means that instead of extension field of $\mathbb{Q}$, we can pass a finite group as a key, which further increases the security of our cryptosystem.

6 Minimal list of counterexamples in monoids

In [4] Section 4 author consider 24 square-free and radical factorizations and all dependencies in general monoids and in particular monoids. In this section we consider a minimal list of possible counterexamples that we can look for in a commutative cancellative monoids. Some of them are at [4] Section 7.

The statement that there are (in general) no other implications than the ones stated in [4] is equivalent to the existence of the following counterexamples.

1. Any monoid satisfying: $4s \wedge \neg 0s$, $5s \wedge \neg 0s$, $5.1s \wedge \neg 0s$, $3s \wedge \neg 1s$, $5.1s \wedge \neg 4s$, $4s \wedge \neg 4.1s$, $2s \wedge \neg 4.2s$, $5.3s \wedge \neg 4.2s$, $1s \wedge \neg 5s$, $3s \wedge \neg 5s$, $4s \wedge \neg 5s$, $5.1s \wedge \neg 5s$, $2s \wedge \neg 5.3s$, $4s \wedge \neg 5.3s$, $4.1s \wedge \neg 5.3s$, $1s \wedge \neg 6s$, $4s \wedge \neg 6s$, $5s \wedge \neg 6s$, $5.1s \wedge \neg 6s$, $5r \wedge \neg 0r$, $0r \wedge \neg 1r$, $3r \wedge \neg 1r$, $5.1r \wedge \neg 4r$, $5.2r \wedge \neg 4.1r$, $1r \wedge \neg 4.2r$, $3r \wedge \neg 4.2r$, $5.3r \wedge \neg 4.2r$, $1r \wedge \neg 5.3r$, $3r \wedge \neg 5.3r$, $4r \wedge \neg 5.3r$, $1r \wedge \neg 6r$, $4r \wedge \neg 6r$.

2. Non-factorial GCD-monoids satisfying: $4/5s \wedge \neg 0s/1s/2s/3s$, $6s \wedge \neg 4.1s$ $/5.1s$, $5.3s \wedge \neg 4.2s/5.2s$, $4.1s/5.1s \wedge \neg 6s$.

3. Pre-Schreier non-GCD-monoids satisfying: $4s/5s \wedge \neg 0$, $4.1s/5.1s \wedge \neg 4s/5s$, $4.2s/5.2s \wedge \neg 4.1s/5.1s$, $5.3s \wedge \neg 4.2s/5.2s$, $3s \wedge \neg 5.3s$, $0s \wedge \neg 6s$, $4.1s/5.1s \wedge \neg 6s$.

4. SR-non-pre-Schreier monoids satisfying : $5s \wedge \neg 0s$, $0s \wedge \neg 1s$, $3s \wedge \neg 1s$, $5.1s \wedge \neg 4s$, $5.2s \wedge \neg 4.1s$, $1s \wedge \neg 4.2s$, $3s \wedge \neg 4.2s$, $5.3s \wedge \neg 4.2s$, $1s \wedge \neg 5.3s$, $3s \wedge \neg 5.3s$, $4s \wedge \neg 5.3s$, $1s \wedge \neg 6s$, $4s \wedge \neg 6s$.

5. Non-factorial ACCP-monoids satisfying: $4s \wedge \neg 0s$, $4.1s \wedge \neg 0s$, $5.2s \wedge \neg 0s$, $6s \wedge \neg 0s$, $0s \wedge \neg 1s$, $5.1s \wedge \neg 4s$, $4s \wedge \neg 4.1s$, $5.2s \wedge \neg 4.1s$, $2s \wedge \neg 4.2s$, $5.3s \wedge \neg 4.2s$, $6s \wedge \neg 4.2s$, $1s \wedge \neg 5s$, $4s \wedge \neg 5s$, $4.1s \wedge \neg 5s$, $5.3s \wedge \neg 5s$, $6s \wedge \neg 5s$, $2s/3s \wedge \neg 5.3s$, $4s \wedge \neg 5.3s$, $4.1s \wedge \neg 5.3s$, $1s \wedge \neg 6s$, $4s \wedge \neg 6s$, $4.1s \wedge \neg 6s$, $5.2s \wedge \neg 6s$, $4r \wedge \neg 0r$, $5.3r \wedge \neg 0r$, $0r \wedge \neg 1r$, $4.1r \wedge \neg 4r$, $4.2r \wedge \neg 4.1r$, $1r \wedge \neg 4.2r$, $5.3r \wedge \neg 4.2r$, $1r \wedge \neg 5.3r$, $4r \wedge \neg 5.3r$.

6. An atomic non-ACCP monoid satisfying exactly the same conditions as in 1.

   We strongly encourage to write a program that generates the minimum set of counterexamples for any set of implications. And we leave the question about how to use such a list to create a cryptosystem.

References

[1] Matysiak, Ł, *Generalized RSA cipher and Diffie-Hellman protocol*, J. Appl. Math.& Informatics Vol.39 (2021), No. 1 - 2, pp. 93 – 103

[2] Matysiak, Ł, *A structure of Dedekind in the cryptosystem*, `https://lukmat.ukw.edu.pl/files/A-structure-of-Dedekind-in-the-cryptosystem.pdf`, (2021).

[3] Jankowska, M., Matysiak, Ł, *A polynomial composites and monoid domains as algebraic structures and their applications*, Global Journal of Science Frontier Research: F Mathematics and Decision Sciences, **21** (3), (2021).

[4] Matysiak, Ł, *On square-free and radical factorizations and existence of some divisors and relationships with the Jacobian conjecture*, (2021).

[5] Matysiak, Ł, *A finite group is a Galois group*, (2022).