Discreet Discrete Mathematics

Secret Communication Using Latin Squares and Quasigroups

Christoffer Olsson
Abstract

This thesis describes methods of secret communication based on latin squares and their close relative, quasigroups. Different types of cryptosystems are described, including ciphers, public-key cryptosystems, and cryptographic hash functions. There is also a chapter devoted to different secret sharing schemes based on latin squares. The primary objective is to present previously described cryptosystems and secret sharing schemes in a more accessible manner, but this text also defines two new ciphers based on isotopic latin squares and reconstructs a lost proof related to row-latin squares.

Diskret diskret matematik
Hemlig kommunikation med latinska kvadrater och kvasigrupper

Sammanfattning

1 Introduction and basic notation

The internet has drastically changed the way people communicate and one can argue that society today communicates more than it has ever done before, but with the increase in telecommunication comes another, less desirable consequence: the possibility of someone eavesdropping. Given the large amount of information that is transferred using the internet, of which some (banking, shopping, private communications etc.) is certainly considered sensitive, it is clear that ways to protect ones privacy are very important. Cryptography gives the possibility to do just that, protect ones privacy, and it does so in a variety of ways.

Perhaps the most classical scenario in which cryptography is used is in direct communication between two (or more) parties that want to minimize the risk of someone eavesdropping. The parties involved can be everything from two friends having a conversation to a web server and a client visiting it. To guarantee privacy the parties can use ciphers or public-key cryptosystems to obscure their messages.

Another area of importance in modern cryptography is the question of authentication or, in other words, how to verify that a message claiming to be from one person is not actually sent by an outsider. If the outsider has malicious intent it might be trying to get access to private information that it shouldn’t normally have access to. On a similar note, cryptography also concerns itself with data integrity, i.e. how to verify that a set of data, for example a computer file, has not been tampered with by an outsider. Both of the problems above can be overcome using cryptographic hash functions.

A problem closely related to cryptography arise in situations where certain information or actions require the cooperation of several parties to be accessed, for example when several business executives need to agree to a certain transaction, or when several military leaders need to agree to activate a nuclear weapon. Secret sharing schemes can be implemented to successfully and securely coordinate such scenarios.

No cryptosystem is perfect and there is always the possibility of loopholes that can be exploited to reveal information that is supposed to be hidden. Such an exploit is called a cryptographic attack and can be used by an outsider to eavesdrop on what is supposed to be private communication. In addition to describing new cryptosystems, modern cryptography also concerns itself with preemptively describing vulnerabilities in existing cryptosystem in order to minimize the effectiveness of such attacks.

This text is a survey of cryptographic methods and secret sharing schemes that use latin squares or quasigroups in some way. It will focus mostly on the practical details of the proposed schemes, and only mention security aspects and possible cryptographic attacks in passing. For answers to more general question regarding cryptography and secret sharing schemes one can consult [22].

1.1 Latin squares

As will be seen in later chapters, latin squares and related concepts can be useful in many aspects of secret communication. First of all some notation needs to be introduced.
Definition 1.1. A latin square of order \( n \) is an \( n \times n \) array of symbols from the set \( \mathbb{N} \) of size \( n \) such that each symbol occurs exactly once in each row and each column.

See figure 1.1 for an example of a latin square.

Latin squares have been proposed as good candidates for use in cryptography due to the large number of different squares for a fairly small order; there are, for example, approximately \( 10^{37} \) different latin squares of order 10 [21].

The symbol set for a latin square can be taken to be any set with a cardinality that corresponds to the order of the square, but in practice it is often taken to be the set \( \{1, 2, ..., n\} \) (see figure 1.1) or \( \mathbb{Z}_n = \{0, 1, ..., n-1\} \) (see figure 1.2) for a square of order \( n \).

For ease of notation a latin square can be seen as a set of ordered triples \( \{(i, j; k)\} \), such that \( k \in \mathbb{N} \) is the symbol in row \( i \) and column \( j \).

1.2 Quasigroups

There is a close relationship between latin squares and the algebraic structures called quasigroups, and in the context of cryptography both of them can be useful.

Definition 1.2. A quasigroup \((Q, \ast)\) is a set, \( Q \), together with a binary operation, \( \ast \), such that both of the equations \( x \ast a = b \) and \( a \ast y = b \) have unique solutions for all \( a, b \in Q \). The order of the quasigroup is the cardinality of the set \( Q \).

For a finite set \( Q \) one can describe the structure of the quasigroup \((Q, \ast)\) using a multiplication table. The uniqueness of solutions of the equations mentioned above guarantees

\[
\begin{array}{ccc}
* & a & b & c \\
\hline 
a & b & a & c \\
b & c & b & a \\
c & a & c & b \\
\end{array}
\]

Figure 1.3: A multiplication table of a quasigroup of order 3.
that each element occurs exactly once in each row and each column, thus the multiplication
table of a quasigroup is a latin square (see figure 1.3).

Quasigroups have been proposed as good candidates for use in cryptography due to their
lack of associativity [8].

1.3 Isotopisms, isomorphisms and conjugates

An isotopism is a mapping that permutes the rows, columns and symbols of a latin square.
Two latin squares are said to be isotopic if one can be mapped into the other using an
isotopism.

Definition 1.3. Let \( L \) be a latin square and let \( \alpha, \beta \) and \( \gamma \) be permutations of the rows, the
columns and \( N \) (the set of symbols of the latin square), respectively. Then \( \theta = (\alpha, \beta, \gamma) \) is an
isotopism, and \( L \) is isotopic to \( \theta(L) \).

Example 1.1. The two latin squares in figure 1.4 are isotopic since \( \theta \) with
\[
\alpha, \beta, \gamma = \begin{pmatrix}
1 & 2 & 3 \\
2 & 3 & 1 \\
3 & 1 & 2
\end{pmatrix}
\]
maps figure 1.4a into 1.4b.

![Two isotopic latin squares](image)

Figure 1.4: Two isotopic latin squares

Note that the relation of being isotopic defined on the set of all latin squares is an
equivalence relation. The equivalence classes formed in this way are called isotopy classes.

An isotopism from one latin square to itself is said to be an autotopism. If \( \theta \) is an
autotopism of \( L \) one can say that \( L \) admits \( \theta \).

A special case of isotopism is when \( \alpha = \beta = \gamma \).

Definition 1.4. An isomorphism is an isotopism where \( \theta = (\alpha, \alpha, \alpha) \)

An isomorphism from one latin square to itself is called an automorphism.

Definition 1.5. Let \( \pi \) be a permutation of the set \( \{r, c, s\} \) where the elements correspond
to row, column and symbol of a latin square, respectively. Then \( \pi \) gives rise to a function
that, applied to a latin square, generates a new latin square. The new latin square is called a
conjugate of \( L \).

Since the set \( \{r, c, s\} \) gives rise to six permutations there are six conjugates (including
the square itself) of every latin square.

Example 1.2. Let \( L = \{(i, j; k)\} \) be the latin square from figure 1.1 and let
\[
\pi = \begin{pmatrix}
r & c & s \\
c & r & s
\end{pmatrix}
\]
then \( \pi \) gives rise to the latin square \( L' = \{(j, i; k)\} \), which can be seen in figure 1.5. Using
notation related to matrices, \( L' \) can be seen as the transpose of \( L \).
If one takes the union of a latin square’s six conjugates and their corresponding isotopy classes one obtains what is called the latin square’s main class.

The notions above can also be applied to quasigroups. An isotopism from \((Q,\ast)\) to \((Q,\circ)\) can be defined according to

\[
\alpha(a) \circ \beta(b) = \gamma(a \ast b)
\]

where \(\alpha\), \(\beta\) and \(\gamma\) are permutations. In the same way one can define an isomorphism from \((Q,\ast)\) to \((Q,\circ)\) as

\[
\alpha(a) \circ \alpha(b) = \alpha(a \ast b)
\]

1.4 Remarks and an outline of the thesis

As will probably be obvious to the reader, the cryptosystems discussed in this text will vary greatly in security and practicality. Anyone wanting to implement any of them should take this into consideration. In some cases a remark from the author will point out problems that might have been found.

This text is by no means exhaustive. There are many more proposed cryptosystems in the literature, especially related to quasigroups. Those have been omitted here due to other articles questioning their security or since they are based on theory that go beyond the scope of this text.

This thesis is organized as follows: In chapter 2, two simple ciphers based on latin squares are described. Chapter 3 discusses the application of quasigroups in stream ciphers. Chapter 4 describes the discrete logarithm problem, row-latin squares and some cryptosystems based on these. Chapter 5 gives examples of the use of quasigroups in the construction of hash functions. Finally, chapter 6 discusses secret sharing schemes and ways of constructing them by using latin squares.

Chapter 2-4 address problems of communication between two or more parties, as well as introduce some concepts important for later chapters. Chapter 5 addresses problems of authentication and data integrity. Chapter 6 addresses problems of secret sharing.

1.5 Contributions of this thesis

The main contributions of this thesis is the collection of diverse topics regarding latin squares and quasigroups in cryptography in one place as well as a more accessible presentation of these. Furthermore, this text provides some new examples to illustrate the different cryptosystems and secret sharing schemes. In addition to this, two new ciphers are described and a proof that appears to have been lost is reconstructed from a proof sketch.
2 Simple ciphers based on latin squares

This chapter will focus on some simple ciphers that, even though they might lack for security in the face of modern computers, demonstrates the usefulness of latin squares in cryptography. Before the definition of the ciphers, a general description of the cryptographic process is given.

2.1 Cryptography

Cryptography is concerned with methods that allow two parties, often called Alice and Bob, to communicate over an insecure channel without allowing for a third party, referred to as an adversary, to know what is being said.

In the most general case, where Alice intends to send a message to Bob, she chooses a message, called plaintext, and encrypts it using the cryptographic method of choice, see figure 2.1. The result is called the ciphertext and is sent to Bob over an appropriate channel. When Bob receives the message he decrypts it, using the specified cryptographic method, and acquires the original plaintext. A pair of algorithms used for encryption and decryption is commonly called a cipher.

For a cryptographic method to be efficient it should be easy to encrypt the message, but very difficult or impossible to decrypt it without certain information, that should, preferably, only be known to Bob and perhaps Alice. Such hidden information that is necessary for decryption is commonly called a key. Thus, if the adversary intercepts the ciphertext when it is sent to Bob, they should not be able to acquire the original plaintext without access to the key.

![Figure 2.1: Message transmission](image-url)
2.2 Two new ciphers based on isotopic latin squares

The notion of isotopisms gives rise to two simple and closely related ciphers in the following way.

2.2.1 Cipher with an isotopism as key

Let the message consist of a latin square \( P \) of order \( n \) with symbols taken to be the elements in the set \( N \). Let the secret key be an isotopism \( \theta \). Define the encryption procedure to be

\[ \theta(P) = C \]

where \( C \) is the ciphertext which is to be sent over a possibly insecure channel. Since \( \theta \) consists of three permutations it is invertible, and thus the decryption procedure becomes

\[ \theta^{-1}(C) = P \]

The square \( P \) could be a message in its own right, with rules for interpreting the latin square decided on beforehand. It could also be a latin square to be used in any of the other cryptosystems defined in this text, for example the e-transformation defined in chapter 3. In the later case this cipher allows for secure communication of cryptographic keys.

Example 2.1. Assume that Alice wants to send a message, \( M \), to Bob. Alice and Bob have a shared secret key in the form of \( \theta = (\alpha, \beta, \gamma) \) with

\[ \alpha = \begin{pmatrix} 1 & 2 & 3 & 4 \\ 2 & 4 & 1 & 3 \end{pmatrix}, \beta, \gamma = \begin{pmatrix} 1 & 2 & 3 & 4 \\ 4 & 3 & 2 & 1 \end{pmatrix} \]

Let \( M \) be the latin square in figure 1.1. Alice encrypts \( M \) by taking \( C = \theta(M) \), the resulting latin square can be seen in figure 2.2. She sends this to Bob. When Bob wants to decipher \( C \) he takes \( \theta^{-1}(C) = M \).

![Figure 2.2: A latin square of order 4, \( \theta(M) \) where \( M \) is the square in figure 1.1](image)

An alternative approach Choosing a latin square \( P \) as plaintext might be limiting when compared to a cryptosystem that allows for encryption of an arbitrary string of characters. An alternative approach is to attach an ordered list of positions in the latin square to the ciphertext. The message would then be made up of the symbols found in those positions in the original square \( P \). By using this approach, one could forgo the need of inverting \( \theta \). Instead one could randomize a latin square, \( L \), and then send this square as ciphertext together with an ordered list of positions in \( \theta(L) \). In this case the plaintext could be any combination of symbols in the latin square, i.e. in the set \( N \).

One problem with attaching a list of positions to the ciphertext is that two positions referring to the same row or column can’t correspond to the same symbol, by definition of a latin square. This fact might reveal partial information of the plaintext, which is clearly
a security issue. To circumvent the problem one could let each symbol in the plaintext alphabet correspond to several symbols in the latin square so that \( n = k |A| \), where \( k \) is a positive integer and \( A \) is the set of symbols that make up the possible plaintexts. Note that it is important that \( \theta \) doesn’t have too many fixed points, since this could reveal partial information of the plaintext.

If this alternative method is used, the ciphertext and the list of positions would be public knowledge, while the key, \( \theta \), would have to be kept secret.

**Example 2.2.** Assume that one wants to send the message \( M = (2, 3, 1, 2, 3) \) as an encrypted message using the cipher with isotopism as key and an attached list of positions.

Let \( P \) be the latin square in figure 2.3a and assume that it is to be encrypted and communicated to another party. By encrypting \( P \) as described above and attaching the list of positions \( ((1, 2)(3, 1)(2, 3)(3, 3)(2, 2)) \) one sees that \( M \) can be recovered from the transmitted ciphertext.

Let \( P' \) be latin square in figure 2.3b and that \( M \) is to be sent using \( P' \) and an ordered list of position. Furthermore, let each symbol in the message, i.e. 1, 2 and 3, correspond to two symbols in \( P' \) in the following way

\[
\begin{align*}
1 & \sim \{a, d\} \\
2 & \sim \{b, e\} \\
3 & \sim \{c, f\}
\end{align*}
\]

where \( \sim \) is to be read as corresponds to. Then the ordered list \( ((4, 2)(5, 5)(2, 6)(6, 3)(1, 6)) \) defines \( M \).

![Latin Square of Order 3](a) and Latin Square of Order 6](b)

**Figure 2.3**

### 2.2.2 Cipher with a latin square as key

Another approach is to take a latin square as the secret key, and to send an isotopism to be applied to the square as the ciphertext. In this case it would be wise to number the rows and columns arbitrarily (see figure 2.4) since the permutations corresponding to rows and columns doesn’t provide any extra security if an adversary knows their original ordering.

Finding an isotopism such that \( L' = \theta(L) \) for given \( L \) and \( L' \) can be difficult ([15] gives an algorithm with worst case complexity \( O(n^{\log_2 n}) \) for the closely related issue of determining if two latin squares are isotopic). Therefore this method works best when an ordered list of positions is attached to the ciphertext. This allows for taking a random isotopism \( \theta \) as the ciphertext and then attaching a list of positions in \( \theta(L) \), where \( L \) is the secret key. These positions could either refer to the natural ordering of rows and columns, or to the arbitrary
ordering applied to them. Of course, the communicating parties need to decide on what the positions refer to beforehand.

In this cipher the ciphertext $\theta$ and the list of positions would be public knowledge. The latin square and the arbitrary ordering of the rows and columns should be kept secret.

Example 2.3. Say that Alice and Bob have as shared key the latin square, $L$, in figure 2.5 with ordering of rows and columns as specified in the figure. If Bob wants to send the message $(1,3,4,3,2)$ to Alice, he first needs to decide on an isotopism. Assume that Bob randomly generates $\theta$ from example 2.1. He applies $\theta$ to $L$ and gets the latin square, $L'$, which can be seen in figure 2.6. He sends $\theta$ together with an ordered list of positions in $L'$:

$$
\begin{array}{cccc}
4 & 1 & 2 & 3 \\
3 & 1 & 2 & 3 & 4 \\
1 & 2 & 3 & 4 & 1 \\
4 & 4 & 1 & 2 & 3 \\
2 & 3 & 4 & 1 & 2 \\
\end{array}
$$

Figure 2.5: A latin square of order 4, taken to be the key in example 2.3. The numbers in the uppermost row correspond the the ordering of the columns, and the numbers in the leftmost column correspond to the ordering of the rows.

Alice can easily retrieve $\theta(L) = L'$ and find the given positions in this square which gives her the decrypted message.

2.3 A cipher based on mutually orthogonal Latin squares (MOLS)

The concept of orthogonal latin squares is a classic idea in mathematics, and, as will be shown, it can have applications in cryptology.

Definition 2.1. Two latin squares, $L_1$ and $L_2$, of order $n$ are said to be orthogonal if every ordered pair $(k_1,k_2)$, where $(i,j;k_1) \in L_1$ and $(i,j;k_2) \in L_2$, occurs exactly once for all

$$
((1,2)(1,4)(4,3)(3,3)(2,4)) \text{ (these positions refer to the natural ordering of the rows and columns) to Alice.}
$$

Alice can easily retrieve $\theta(L) = L'$ and find the given positions in this square which gives her the decrypted message.
positions \((i, j)\). A set of latin squares is called **mutually orthogonal** if all latin squares in the set are pairwise orthogonal to each other.

An example of orthogonal latin squares can be seen in figure 2.7.

In [20] Sarvate and Seberry proposes an encryption method using mutually orthogonal latin squares (MOLS). The main idea of the article is to encrypt the message \((i, j)\) by sending the \(m\)-tuple occurring in the position \((i, j)\) of \(m\) mutually orthogonal squares. Since the squares are MOLS each tuple will uniquely determine a position \((i, j)\). To construct the cipher one needs a set of MOLS and a way to choose and order \(m\) of them for encryption or decryption. This information will be kept secret.

**Example 2.4.** Take \(m = 2\) and use the set of MOLS in figure 2.7. Assume that the key chooses the first and second latin squares, in that order. If the message to be encrypted was \((3, 2)\) the transmitted ciphertext would be the entry in position \((3, 2)\) of each of the squares, i.e. \((3, 4)\). Given that they don’t know which latin squares were used for encryption, an eavesdropping adversary would not be able to tell which position the ordered pair corresponds to.

One could also take \(m = 3\) and use all three of the squares from table 2.7, in the same order as they appear there. In this case the message \((3, 2)\) would result in the ciphertext \((3, 4, 1)\).

**Remark.** Since the plaintext is one position in a latin square, the approach described above allows for \(n^2\) different messages, where \(n\) is the order of the squares, regardless of how large \(m\) is. One might wonder if this encryption method could not be made even better by taking the position as ciphertext and entries as plaintext. Then the ciphertext would specify a position in a latin square and the plaintext would be all of the symbols found in that position in the \(m\) different MOLS. Assuming an ordering of the MOLS has been specified in advance, the result would be an ordered \(m\)-tuple that can be seen as a string. Since each position specifies one of \(n\) symbols in each of the \(m\) squares, there are \(n^m\) possible plaintexts for arbitrary \(m\) instead of \(n^2\). In the same way the messages could be compressed since each position encrypts \(m\) symbols.

It would seem that this would make the system more difficult to break since an adversary wouldn’t now the length of the message without knowing \(m\), which should be private knowledge just like the key.
3 Stream ciphers

The simplest approach to ciphers is to encrypt all digits of the message using the same key, \( K \), as was done in chapter 2. Another approach is to generate a stream of digits, called a keystream, and to encrypt every digit in the message individually, using the corresponding digit of the keystream. The keystream depends on the chosen key, \( K \), to set up its initial state. This form of cipher is called a stream cipher.

In some cases the keystream depends on the previous letters in the plaintext or ciphertext message, in addition to \( K \). This kind of stream cipher is called asynchronous or self-synchronizing. When the keystream depends only on \( K \) and is independent of the message it is called synchronous.

Stream ciphers often work with binary digits, and in those cases encryption and decryption is usually taken to be addition modulo 2, which is the same as subtraction modulo 2. Using the common notion of letting 0 correspond to the Boolean value "false" and 1 to correspond to the Boolean value "true" one finds that addition mod 2 is equivalent to the exclusive-or operation.

Definition 3.1. The exclusive-or operator is an operator that takes as input two Boolean values and returns "true" if exactly one of the inputs is true and "false" otherwise. It is often abbreviated XOR.

The XOR operation is efficient to implement in hardware, which makes stream ciphers using the operation efficient, as well.

Since XOR is equivalent to both addition mod 2 and its inverse subtraction mod 2, one can use the operation for both encryption and decryption, given a certain keystream.

Example 3.1. Assume that the string \( M = (0, 0, 0, 1, 1, 0, 1, 1) \) is to be encrypted and that the following keystream has been generated by some method: \( K = (0, 1, 1, 0, 1, 0, 0, 1) \). Applying the XOR operation results in the following ciphertext

\[
C = M \ XOR \ K = (0, 1, 1, 0, 0, 1, 0)
\]

The message can then be decrypted by again applying the XOR operation.

\[
C \ XOR \ K = (0, 0, 0, 1, 1, 0, 1, 1) = M
\]

3.1 Quasigroup string transformations

If one sees the elements of a quasigroup \( (Q, *) \) as the letters of an alphabet it is possible to combine them into strings. Since this allows for writing messages in this alphabet it has obvious applications in cryptology.

When working with strings it is useful to define the concatenation operation.

Definition 3.2. The concatenation operation, denoted by \( || \), takes two strings \( \alpha = (a_1, a_2, \ldots, a_n) \) and \( \beta = (b_1, b_2, \ldots, b_k) \) as input and outputs one string

\[
\alpha || \beta = (a_1, a_2, \ldots, a_n, b_1, b_2, \ldots, b_k)
\]
Figure 3.1: A graphical representation of the e-transformation with leader $a$, $\alpha = (a_1, a_2, \ldots, a_n)$ and $e_{l,*}(\alpha) = (b_1, b_2, \ldots, b_n)$. The figure is taken from [12].

To allow for encryption of this kind of quasigroup strings, so called quasigroup string transformations have been introduced. Two such transformations are described here.

**Definition 3.3.** Let $(Q, \ast)$ be a quasigroup and $Q^+$ be the set of nonempty strings formed by the elements of $Q$. Then the function $e_{l,*}: Q^+ \to Q^+$ is defined for each $l \in Q$ and $\alpha = (m_1, m_2, \ldots, m_n) \in Q^+$ by

$$e_{l,*}(\alpha) = (c_1, c_2, \ldots, c_n)$$

$$c_1 = l \ast m_1$$

$$c_i = c_{i-1} \ast m_i \text{ for } 1 < i \leq n$$

The function $e_{l,*}$ is called an *e-transformation* of $Q^+$ with leader $l$ and operation $\ast$.

One can see the e-transformation as an asynchronous stream cipher with $(l, \ast)$ as the key. In this case the string $(c_1, c_2, \ldots, c_n)$ can be seen as the ciphertext. A graphical representation of the e-transformation can be seen in figure 3.1.

Since the equation $x \ast a = b$ has a unique solution in a quasigroup (by definition) one can define another operation $\star$ defined by

$$x \star b = a \text{ if and only if } x \ast a = b$$

This is also a quasigroup since its corresponding latin square is a conjugate (see definition 1.5) of the original quasigroup’s latin square. It can be used to define the inverse of the e-transformation, $d_{l,*}$.

**Definition 3.4.** Let $(Q, \ast)$ be a quasigroup and define a function $d_{l,*}: Q^+ \to Q^+$ for each $l \in Q$ which maps a string $(c_1, c_2, \ldots, c_n) \in Q^+$ into $(m_1, m_2, \ldots, m_n)$, where

$$m_1 = l \star c_1$$

$$m_i = c_{i-1} \star c_i \text{ for } 1 < i \leq n$$

Just like with the e-transformation, $l$ is called the leader and $\ast$ is called the operation.

The function $d_{l,*}$ can be used to decrypt a message encrypted by $e_{l,*}$.

**Example 3.2.** Assume that Alice wants to send the message $abc$ to Bob, and that the two of them have previously agreed to encrypt their messages using the quasigroup in figure 1.3 with $a$ as leader. To encrypt the plaintext $abc$ Alice should use $e_{a,*}$ and make the following calculations:

$$c_1 = a \ast a = b$$

$$c_2 = b \ast b = b$$

$$c_3 = b \ast c = a$$
Thus the ciphertext that she sends to Bob is $bba$.

When Bob receives the ciphertext and wants to decrypt it he can use $d_a$, and compute:

$$m_1 = a * b = a$$
$$m_2 = b * b = b$$
$$m_3 = b * a = c$$

which brings back the original message, $abc$.

Since it is possible to define many different operations and leaders on a set $Q$ one can extend the notion of an $e$-transformation in the following way

**Definition 3.5.** Given a set $Q$, let $*_1, *_2, \ldots, *_n$ be a set of operations on $Q$ and let $l_1, l_2, \ldots, l_n \in Q$ be called leaders (neither the operations nor the leaders need be distinct). Define $E_k = E_{l_1, l_2, \ldots, l_n} = e_{l_1, *_1} \circ e_{l_2, *_2} \circ \ldots \circ e_{l_n, *_n}$, where $\circ$ denotes function composition. $E_k$ is called an $E$-transformation of $Q$.

### 3.2 Edon80, a synchronous stream cipher

Edon80 is a synchronous stream cipher based on the E-transformation. It was first described in [12] and is designed to be efficiently implemented in hardware with significant speed asymmetry, i.e. much better performance when implemented in hardware than in software. Hardware encryption can be useful in, for example, protection and access control of multimedia content like DVDs and CDs, and the speed asymmetry makes it more difficult to simulate the cipher in software in attempts to break it.

Edon80 uses four quasigroups of order 4 and the e-transformation to generate a keystream which is then added to the binary message using the XOR operation. The quasigroups will consist of the elements 0, 1, 2 and 3. A list of quasigroups of order 4 that is suitable for use in an implementation of edon80 is given in [12]. Ideally, the quasigroups are kept secret but for security considerations they can be assumed to be public knowledge.

Other than the four quasigroups, the cipher depends on a key, which will be a vector of 80 bits, and an initialization vector (IV) of 64 bits. Both of these will be kept secret. The IV needs to be padded by the 16 bits: 1110010000011011, so that it, too, is 80 bits long. After this, both the key and the IV will be seen as the concatenation, or vector, of 40 2-bit variables. Since the numbers 0, 1, 2 and 3 (written in base 10) can be represented in base 2 by using 2 bits, both the key and the IV will be vectors of 40 symbols from the set $Q = \{0, 1, 2, 3\}$.

The keystream itself will be generated by applying 80 e-transformations to a periodic string $(0, 1, 2, 3, 0, 1, 2, 3, \ldots)$. The IV will be used to set up the leaders of the e-transformations and the key will be used to assign an operation, corresponding to one of the four quasigroups, to each of the 80 e-transformations. Note that one can see the 80 e-transformations of Edon80 as one E-transformation. In light of this, one can think of the key as deciding the sequence of operations, $*_i$, for the E-transformation, where all $*_i$ belongs to one of the four quasigroups.

Label the four quasigroups $(Q, *_1), (Q, *_2), (Q, *_3), (Q, *_4)$, and let the key, $K = (K_0, K_1, \ldots, K_{39})$, and the IV, $V = (v_0, v_1, \ldots, v_{39})$, be vectors of 40 2-bit variables, i.e. elements of the four quasigroups. The keystream is determined in three steps: **KeySetup** mode, **IVSetup** mode and **Keystream** mode.
The leaders will be chosen according to:

• where \(e\)-transformations on a string consisting of the key concatenated with the IV:

\[ K_0, K_1, ..., K_{39}, v_0, v_1, ..., v_{39} \]

Keystream cipher. A graphical representation of the IV setup of \(K_0, K_1, ..., K_{39}, v_0, v_1, ..., v_{39}\) can be every second symbol of the string, i.e. \(K_0, K_1, ..., K_{39}, v_0, v_1, ..., v_{39}\) mode can be found in figure 3.2

Figure 3.2: A graphical representation of IV setup mode where \(a_{i, j}\) is the \(j\)th symbol in the string resulting from the e-transformation using operation \(*_i\). The original string is in the first row and the string in the last row is used to initialize leaders. This is a modified version of a figure taken from [12].

In KeySetup mode the key will determine the sequence of operations according to

\[ *_i = \begin{cases} *_{K_i}, & 0 \leq i \leq 39 \\ *_{K_{40-i}}, & 40 \leq i \leq 79 \end{cases} \]

where \(*_i\) is the \(i\)th operation in the sequence.

In IV Setup mode the leaders of the operations will be determined by performing the 80 e-transformations on a string consisting of the key concatenated with the IV: (\(K_0, K_1, ..., K_{39}, v_0, v_1, ..., v_{39}\)), and with the same string in reverse (seen as a vector) as leaders: (\(v_{39}, ..., v_1, K_39, ..., K_1, K_0\)). The symbols of the resulting string will be taken as the leaders for the e-transformations generating the keystream. If the string is \((a_0, a_1, ..., a_{79})\), the leaders will be chosen according to: \(l_i = a_i\) for \(0 \leq i \leq 79\). A graphical representation of IV setup mode can be found in figure 3.2

In Keystream mode the 80 e-transformations will be performed on the periodic string \((0, 1, 2, 3, ...)\) using the sequence of operations determined in KeySetup mode and the leaders determined in IV Setup mode. If the resulting string is \((a_0, a_1, ...)\), the keystream will be every second symbol of the string, i.e. \((a_1, a_3, ..., a_{2k-1})\) for \(k = 1, 2, ...\). The principle of only choosing the odd numbered entries in the string is important to the security of the cipher. A graphical representation of the keystream mode can be found in figure 3.3.

To encrypt a message one XORs it with the keystream, and since XOR is equivalent to bot addition and subtraction mod 2, the decryption is performed in the same way. Because

<table>
<thead>
<tr>
<th>Operation</th>
<th>Leader</th>
<th>String</th>
</tr>
</thead>
<tbody>
<tr>
<td>*0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>*1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>*79</td>
<td>79</td>
<td>79</td>
</tr>
</tbody>
</table>

Figure 3.3: A graphical representation of keystream mode where \(a_{i, j}\) is the \(j\)th symbol in the string resulting from the e-transformation using operation \(*_i\). The original string is in the top row and the string in the last row is used to initialize the keystream. This is a modified version of a figure taken from [12].
of this, anyone that knows the key and the IV can produce the necessary keystream and decrypt the message.

See [12] for further considerations regarding implementation of Edon80 in hardware and a security analysis.
4 Public-key cryptography using row-latin squares

The ciphers considered in this text so far have all been designed to allow two or more prede-
termined parties to communicate using a single, agreed upon, key. Such ciphers are called
symmetric since the same key is used for both encryption and decryption. This chapter will
instead consider asymmetric ciphers, in which there are different keys for encryption and
decryption. This kind of cipher is often used in public-key cryptography, which is based on
two different keys, one that is publicly available and one that is only known to the owner
of the cryptosystem. Thus, like the subject of the previous two chapters, public-key cryp-
tography allows for secure communication between two or more parties, but it does so in a
slightly different way.

Many public-key systems derive their security from the difficulty of something called
the discrete logarithm problem.

Definition 4.1. A discrete logarithm is an integer k such that \( a^k = b \) where \( a \) and \( b \) are
elements of a group \( G \)

Thus a discrete logarithm is an analogue of ordinary logarithms in the context of group
theory. Depending on the group there might not be an integer \( k \) for all combinations of \( a \)
and \( b \), or there might be several. The discrete logarithm problem is the problem of finding
\( k \) given \( a \) and \( b \).

So far, no efficient general algorithm for solving the discrete logarithm problem has
been found and thus it can be very difficult if the right group is used. On the other hand it
is fairly easy to calculate \( b \) given \( a \) and \( k \). Many cryptographic methods based on groups
make use of this discrepancy in difficulty to make encryption (i.e. exponentiation) easy
while keeping decryption (i.e. taking the discrete logarithm) difficult, unless one has access
to certain knowledge making the decryption easier.

4.1 Public-key cryptography

Using the discrete logarithm problem one can devise methods of public-key cryptography.
Using such a method a person, Alice, finds two keys: one private, that she keeps to herself,
and one public that she makes publicly available. If some other person, Bob, wants to send
a message to Alice he can look up her public key and use it encrypt his message. Due to
the nature of the cryptographic method and the discrete logarithm problem the resulting
ciphertext will be difficult to decrypt unless one also has access to the private key, which
only Alice has.

The cryptographic schemes described in this chapter are based on earlier and more
widely known public-key cryptosystems. The original systems are described here, and their
analogues are described in a subsequent section.
4.1.1 The Diffie-Hellman key exchange

The Diffie-Hellman key exchange is a method for exchanging, or establishing, cryptographic keys in a secure way over an insecure channel. It is a form of public-key cryptography that depends on the discrete logarithm problem.

Working in a finite cyclic group \(G\) of order \(n\), Alice and Bob agrees on a generator of the group, \(g\). The generator is not required to be kept secret. Alice chooses a random positive integer \(a\) with \(1 \leq a < n\) and sends \(g^a\) to Bob. Similarly, Bob chooses a random positive integer \(b\) with \(1 \leq a < n\) and sends \(g^b\) to Alice. Now they both compute \((g^a)^b = (g^b)^a = g^{ab}\) and take this as their shared key. The only information an eavesdropping adversary could gain from this communication is \(g^a\) and \(g^b\), and therefore, assuming that the discrete logarithm problem is difficult in \(G\), the key \(g^{ab}\) remains secret. This means that the group \(g\) can be seen as the public key, and the integers \(a, b\) can be seen as private keys.

Note that if an adversary might be assumed to know \(g\), taking \(a, b\) such that either \(g^a = g\) or \(g^b = g\) would make the system insecure, since (assuming that \(g^a = g\)) the agreed-upon key would be \((g^a)^b = g^b\) which is sent over the insecure channel. Because of a similar argument it is prudent to take \(\gcd(a, n)\) and \(\gcd(b, n)\) since this guarantees that \(g^a \neq e\) and \(g^b \neq e\) by Lagrange’s theorem.

4.1.2 The ElGamal cryptosystem

The ElGamal system is public-key system based on the Diffie-Hellman key exchange. Like the no-key system it allows for encryption of arbitrary messages, not only cryptographic keys.

If Alice wants to set up a public-key system that allows anyone to send a secure message to her she can do so in the following way. She should choose a finite, cyclic group of order \(n\) and with generator \(g\). She should then calculate \(g^a\) for \(1 \leq a < n\) chosen randomly. She makes \(g^a\), as well as information about \(G, n\) and \(g\), public. This is her public key and \(a\) is her private key.

If Bob wants to send a secure message, \(m\), to Alice, he chooses a random \(b\) such that \(1 \leq b < n\). Using \(b\) and Alice’s public key he computes and sends the encrypted message

\[(g^b, m \cdot g^{ab})\]

When Alice receives the encrypted message from Bob she can use her private key to compute \((g^b)^a = g^{ab}\) and find \(g^{-ab}\). Then she can take

\[m \cdot g^{ab} \cdot g^{-ab} = m \cdot g^{ob - ab} = m \cdot g^0 = m\]

and thus retrieve the plaintext message.

4.1.3 The RSA cryptosystem

The RSA cryptosystem is an example of public-key cryptography that depends on the difficulty of the discrete logarithm problem and the problem of factoring a product of two large prime numbers. It was first described in [19].

If Alice wants to set up the RSA cryptosystem she needs to choose two prime numbers \(p\) and \(q\) and find their product \(n = pq\). She also needs to calculate \(\phi(n) = (p - 1)(q - 1)\)
(where $\phi(\cdot)$ is Euler’s totient function) and find an integer $d$ such that $1 < d < \phi(n)$ and $\gcd(\phi(n), d) = 1$. Finally she should compute the multiplicative inverse of $d$: $e = d^{-1} \pmod{\phi(n)}$. The public key will consist of the pair $(n, e)$ and the private key will be $d$.

If Bob wants to send a message, to Alice, he needs to transform the message into an integer $m$ such that $\gcd(m, n) = 1$. He then computes $c \equiv m^e \pmod{n}$ and sends this to Alice. Alice can then use her private key to find $cd \equiv m \pmod{n}$.

This is true since $ed \equiv 1 \pmod{\phi(n)}$ implies $ed = k \cdot \phi(n) + 1$ for some positive integer $k$ and therefore $m^{ed} = m^{k \phi(n) + 1} = m^{(m^{\phi(n)})^k} \equiv m \pmod{n}$ due to Euler’s theorem, since $\gcd(m, n) = 1$.

If $p$ and $q$ are large enough it will be difficult to factor $n$, and therefore it will also be difficult to find $\phi(n) = (p - 1)(q - 1)$. Without knowing $\phi(n)$ one can’t find $d$ such that $ed = 1 \pmod{\phi(n)}$. At the same time, the discrete logarithm problem makes it difficult to find $d$ such that $(m^e)^d \equiv m \pmod{n}$, even if one knows $e$, $n$ and even $m$.

### 4.2 Row-latin squares

Before describing the latin square-based analogues of the cryptosystems described in the previous section it is necessary to define a new type of latin square.

**Definition 4.2.** A row latin square of order $n$ is an $n \times n$ array of symbols from the set $N$ of size $n$ such that each symbol occurs exactly once in each row.

In other words a row-latin square allows the same symbol to occur several times in the same column, but not in the same row, see figure 4.1. There is also a corresponding definition of column-latin squares, but this text will focus on row-latin squares.

![Figure 4.1: A row-latin square of order 4](image)

One can see that each row in the row-latin square is a permutation of the set $N$. With this observation one can uniquely determine a row-latin square by giving the permutation of each row $(P_1, P_2, \ldots, P_n)$.

Let two row-latin squares be given by $A = (P_1, P_2, \ldots, P_n)$ and $B = (Q_1, Q_2, \ldots, Q_n)$. Define the product of two row-latin squares to be:

$$AB = (P_1Q_1, P_2Q_2, \ldots, P_nQ_n)$$

and use exponents to denote the product of a row-latin square by itself: $A^2 = AA$.

Using this definition a theorem from [18] states:

**Theorem 4.1.** The set of all row-latin squares (of order $n$) is a group of order $(n!)^n$. 
The group of row-latin squares of order $n$ will be called $RL_n$. The identity element of this group is the row-latin square, $E$, where all rows are the identity permutation. Note that since every row of a row-latin square can be seen as a permutation of the set $N$, a product $AB \in RL_n$ can be seen as $n$ disjoint products, $(a_1b_1, a_2b_2, \ldots, a_nb_n)$, in the symmetric group $S_n$ and, in group theoretic terms, $RL_n$ can be seen as the direct product of $S_n$ with itself $n$ times.

Theorem 4.1 makes it possible to construct encryption methods based on the discrete logarithm problem using row-latin squares. To this end the following theorems are useful:

**Theorem 4.2.** Let $A \in RL_n$ and $E$ be the identity element of $RL_n$, then $A^{n!} = E$.

*Proof.* Look at each row, $i$, of $A^{n!}$ as the product of disjoint products of permutations $P_i$ in the symmetric group of $N$. The order of this group is $n!$ and thus $a_{ij}^{n!}$ is the identity permutation for $1 \leq i \leq n$ by Lagrange’s theorem. Since every row is the identity permutation $A^{n!} = E$ by definition. □

Thus $n!$ is an upper bound on the order of an element in $RL_n$. It is actually possible to do even better and specify a least upper bound on the order of an element, as will be shown after the following definition and lemma.

**Definition 4.3.** Let the least common multiple of integers $a_1, a_2, \ldots, a_n$ be the smallest integer that is divisible by all of $a_1, a_2, \ldots, a_n$. Denote the least common multiple by

$$lcm(a_1, a_2, \ldots, a_n)$$

**Lemma 4.1.** Assume $R \in RL_n$ and let $c_{i,j}$ be the length of the $j$th cycle in the permutation making up the $i$th row of $R$. Then the order of $R$ is

$$O = lcm(lcm(c_{1,1}, c_{1,2}, \ldots, c_{1,k_1}), lcm(c_{2,1}, c_{2,2}, \ldots, c_{2,k_2}), \ldots, lcm(c_{n,1}, c_{n,2}, \ldots, c_{n,k_n}))$$

where $k_i$ is the number of cycles in the $i$th row of $R$.

*Proof.* Consider first the order, $o_i$, of a row $P_i$ seen as a permutation in the symmetric group $S_n$. Let $P_i$ consist of cycles of length $c_{i,1}, c_{i,2}, \ldots, c_{i,k_i}$. For a cycle of length $c_{i,j}$, the elements of that cycle will be in the ordering of the identity permutation for all permutations $P_i^m$ such that $c_{i,j}|m$, where $m$ is a positive integer. Since this is true for all of the cycles of $P_i$, the order $o_i$ is the least common multiple of the cycle lengths, i.e.

$$o_i = lcm(c_{i,1}, c_{i,2}, \ldots, c_{i,k_i}) \quad (4.1)$$

In much the same way, the order of a row-latin square $R$ depends on the order of its rows, seen as elements of $S_n$.

Let $o_1, o_2, \ldots, o_n$ be the order of the rows. The $i$th row is in the form of the identity permutation for all squares $R^l$ such that $o_i|l$, where $l$ is a positive integer. Hence the order of $R$ is $lcm(o_1, o_2, \ldots, o_n)$ which, together with equation 4.1, completes the proof. □

The following theorem gives the least upper bound on the order of an element in $RL_n$.

**Theorem 4.3.** Let $RL_n$ be the group of all row-latin squares of order $n \geq 2$. Let $p_i$ denote the $i$th prime number. Then the least upper bound on the order of any element in $RL_n$ is

$$M_n = p_1^{r_1} p_2^{r_2} \ldots p_k^{r_k}$$

where $p_i \leq n$ and $r_i$ is the largest integer such that $p_i^{r_i} \leq n$ for $1 \leq i \leq k$. 

Remark. A sketch of this proof is given in [17], with reference to a preprint of an article that is supposed to contain the proof in its entirety. Efforts to find this article in published form have failed, and therefore the proof is reconstructed here.

Proof. First it will be proved that $M_n$ is an upper bound on the order of an element in $RL_n$ and then it will be shown that one can always construct a row-latin square of order $M_n$, making it the least upper bound.

Take $R \in RL_n$ with order $O$ in the group. One can see that $R^m = E$ for all positive integers $m$ such that $O|m$. To prove that $M_n$ is an upper bound it is therefore enough to prove that the order of any square divides $M_n$.

Lemma 4.1 shows that the order of row-latin square depends on the length of the cycles of its rows. Consider $c_{x,y}$, the cycle length of the $y$th cycle of the $x$th row. It can either be the number 1, a prime number $p_i \leq n$, or it can be a product of (not necessarily distinct) primes $l = p_1 p_2 ... p_z$ with $p_j < n$, $1 \leq j \leq z$. By lemma 4.1, $O$ is the least common multiple of products of primes, $p_i$ smaller than or equal to $n$. Since $M_n$ is the product of all such primes (such that $p_i \leq n$), $M_n$ must be a multiple of $O$. This completes this part of the proof.

To construct a square of order $M_n$, take the $i$th row to consist of two cycles of length $p_i^r$ and $n - p_i^r$, respectively, for $1 \leq i \leq k$, where $k$ is the number of prime numbers smaller or equal to $n$. The rest of the rows can have arbitrary cycle structure. The order of the $i$th row will be at least $p_i^r$, by lemma 4.1, and thus the order $O$ of the square will be

$$O \geq \text{lcm}(p_1^{r_1}, p_2^{r_2}, ..., p_k^{r_k}) = p_1^{r_1} p_2^{r_2} ... p_k^{r_k} = M_n$$

It has already been shown that $M_n$ is a multiple of $O$ and thus $O$ can’t be larger than $M_n$. Hence $M_n$ is the least upper bound on the order of an element in $RL_n$.

The following corollary follows immediately from the the fact that $O|M_n$ for the order $O$ of any row-latin square in $RL_n$ (which is shown in the proof of theorem 4.3).

Corollary 4.1. $A^{M_n} = E$ for all $A \in RL_n$

By comparing theorem 4.3 with theorem 4.2 one finds the following.

Corollary 4.2. $M_n|n!$

Theorem 4.3 and its corollaries will be useful for adapting the cryptosystems in section 4.1 to the context of row-latin squares.

4.3 Encryption methods

Classical encryption schemes based on the discrete logarithm problem can be implemented by using the group $RL_n$. Several such schemes are proposed in [17] and are presented here, but in some cases with a slight modification from their original description. To increase security it is best to not take the order $n$ of the row-latin squares to be too small.

Note that for all of the schemes below, $M_n$ could be exchanged for $n!$ since $A^{M_n} = A^{n!} = E$.

4.3.1 An analogue of the Diffie-Hellman key exchange

Assume that Alice and Bob need to decide on a key (in the form of a row-latin square) for some encryption scheme, but that they are afraid that someone might eavesdrop on
their communication. Then they can use the following analogue of the Diffie-Hellman key exchange.

Alice and Bob agree on a row-latin square $L$ with order $M_n$ in $RL_n$. Then they each choose private integers $a$ and $b$ such that $1 \leq a, b < M_n$. Like before, it is wise to choose both $a$ and $b$ so that they are relatively prime to $M_n$ and so that neither $L^a = L$ nor $L^b = L$.

Now Alice calculates $L^a = A$ and sends this to Bob. In the same way Bob calculates $L^b = B$ and sends it to Alice. When they have received their respective row-latin square they can find $C = L^{ab}$, which will be their common key.

**Example 4.1.** Take $L \in RL_4$ to be the row-latin square from figure 4.1. For this square $M_n = 12$. Alice takes a random integer $a = 3$ and produces the square in figure 4.2a, which she sends to Bob. Bob takes $b = 2$ and finds the square in figure 4.2b, which he sends to Alice. Now they can both find the square $L^6$, which can be found in figure 4.2c and which will be their common key.

![Figure 4.2: Three row-latin squares of order 4. $L$ is the row-latin square from figure 4.1.](image)

**Remark.** It would seem that the difficulty of the discrete logarithm problem in $RL_n$ depends heavily on its difficulty in $S_n$. Assume that an adversary knows the original square $L$ and the encrypted square $L^a$. If the adversary can find the order of the rows, $o_1, o_2, \ldots, o_n$, in $S_n$ and integers $1 \leq k_i \leq o_i$ such that $l_i^{k_i} = l_i^a$ for every row $l_i$ in $L$ it would greatly reduce the number of values that $a$ could have, since $a$ would have to satisfy

$$a = k_i + m_i o_i, \quad 1 \leq i \leq n$$

for some integers $m_i$.

Lemma 4.1 gives that finding the orders of the rows is only a matter of finding the lengths of their cycles, which might be quite easy, considering that it is assumed that the adversary knows $L$.

A concept related to the Diffie-Hellman key exchange is what is called a no-key system. In this case, the original row-latin square should not only be kept secret, but will actually be the plaintext message. Like before $a$ and $b$ are seen as private keys.

If Alice wants to send a message to Bob she represents her message in the form of a row-latin square $M$ of order $M_n$ in $RL_n$. She then finds an integer $1 \leq a < M_n$ and computes $A_1 = M^a$, which she sends to Bob. Bob, in turn, computes $B = A^b = M^{ab}$ for an integer $a \leq b < M_n$ and sends this to Alice. Alice finds $d'$ such that $ad' \equiv 1 \pmod{M_n}$, and similarly Bob finds $b'$ such that $bb' \equiv 1 \pmod{M_n}$. Note that it is important that $gcd(a, M_n) = 1$ and $gcd(b, M_n) = 1$ so that the inverses, $a', b'$, exist. Alice computes $A_2 = B^{d'}$ and sends it to Bob, who can find

$$A_2^b = (M^{ad'})^{bb'} = M^{bb'} = M$$

since $ad' = k_1 M_n + 1$ and $bb' = k_2 M_n + 1$ for some positive integers $k_1, k_2$. 
Remark. A problem with this system is that, even though \(M^a \neq M\), for some rows one might have \(m^a_i = m\). This would mean that partial information of the message might be leaking out. This makes the system less secure.

4.3.2 An analogue of ElGamal encryption

An analogue of the ElGamal system in \(RL_n\) can be realized as follows. Alice chooses an element \(L \in RL_n\) and a secret integer \(1 \leq a < M\). Like with the Diffie-Hellman key exchange, the square \(L\) should have order \(Mn\). Alice makes \(n, L^a\) and \(L\) publicly available and keeps \(a\) as her private key.

If Bob wants to send Alice a message \(M\) he computes and sends \((L^b, ML^{ab})\) for \(1 \leq b < M\). Now Alice can find \((L^{ab})^{-1} = L^{-ab}\) and compute

\[
ML^{ab}L^{-ab} = M
\]

Like before, \(a, b\) should be relatively prime to \(M\) and such that \(L^a \neq L\) and \(L^b \neq L\).

4.3.3 An analogue of RSA

In order to set up an analogue of RSA in \(RL_n\), Alice needs to find a public key \(1 \leq e_A < M\) such that \(gcd(e_A, Mn) = 1\) and a private key \(d_A\) such that \(e_A d_A \equiv 1 \pmod{Mn}\). The order \(n\) will also be public knowledge.

If Bob wants to send the message \(M \in RL_n\) to Alice, he looks up Alice’s public key, \(e_A\), and calculates \(B = M^{e_A}\). When Bob sends \(B\) to Alice, she can recover the message by using her private key \(d_A\) to calculate \(B^{d_A} = M^{e_A d_A} = M^1 = M\). In similarity to the RSA system, this works since \(M^{e_A d_A} = M^{kMn+1}\) for some integer \(k\) and

\[
M^{kMn+1} = (M^M)^k M = EM = M
\]

by corollary 4.1.

Example 4.2. Assume that Alice has set up a cryptosystem in \(RL_4\) with \(M_4 = 12\). Also assume that Alice has taken her public key to be \(e_A = 7\). Note that \(gcd(7, 12) = 1\).

Say that Bob wants to send a message in the form of the row-latin square in figure 4.1. Call this square \(M\). Bob calculates \(B = M^7\) and produces the square in figure 4.3. Bob sends

\[
\begin{array}{cccc}
2 & 3 & 4 & 1 \\
3 & 1 & 2 & 4 \\
4 & 1 & 2 & 3 \\
1 & 4 & 2 & 3 \\
\end{array}
\]

Figure 4.3: A row-latin square of order 4, \(M^7\), where \(M\) is the square from figure 4.1.

this square to Alice who finds \(d_A = 7\), since \(7 \cdot 7 = 1 \pmod{12}\). Now she can take

\[
B^7 = (M^{e_A})^{d_A} = M^{e_A d_A} = M
\]

and has thus retrieved the original message.

A twist on this concept is to let Bob send \(B' = M^{d_B}\) (where \(d_B\) is his private key) to Alice. Given that \(e_B d_B \equiv 1 \pmod{M_n}\) Alice can then use Bob’s public key, \(e_B\) to find \(M\) in the same way as above. The advantage of this method is that it allows Alice to verify that it really was Bob who sent the message, since he is the only one who knows \(d_B\).
Remark. The original RSA system relies on the difficulty of factoring $n = pq$, which makes finding $\phi(n)$, and in turn $d_A$, difficult unless one knows $p, q$. This aspect of the scheme is not present in the adaption using row-latin squares since $M_n$ is as easy to calculate for anyone knowing the order $n$, which is public knowledge. Finding $d_A$ while knowing $e_A$ and $M_n$ is a fairly easy problem and therefore this cryptosystem should be considered highly insecure.
5 Hash functions and message authentication codes (MACs)

A hash function is a function that takes as input data of arbitrary size and outputs data of a fixed size, called a hash value or a hash. Since the hashes are of fixed size the set of possible inputs is usually larger than the set of outputs. A cryptographic hash function is such a function that is also a one-way function, i.e. it is easy to calculate the hash of a given input but difficult to find the preimage of a given hash.

A secure cryptographic hash function should exhibit the following properties:

- **Preimage resistance**: Given a hash value \( h \) it should be difficult to find an \( x \) such that \( f(x) = h \). Such an \( x \) is called a preimage.

- **Second preimage resistance**: Given a message \( x \) it should be difficult to find an \( x' \) such that \( x \neq x' \) and \( f(x) = f(x') \).

- **Collisions resistance**: It should be difficult to find \( x, x' \) such that \( x \neq x' \) and \( f(x) = f(x') \). If such a pair is found it is called a collision.

These properties are in part motivated by hash functions’ use as “fingerprints” that can guarantee data integrity. Assume that Alice has a publicly available file and that she wants to guarantee that this file has not been tampered with. By computing a hash value of the data in the file and storing it in a secure place she can easily check that the file has not been altered by again calculating the data’s hash and comparing it with the stored hash. Given that the hash function is secure it should be difficult to alter the data in such a way that it still produces the same hash.

A concept closely related to hash functions is that of a compression function.

**Definition 5.1.** A compression function is a function that mixes two inputs of fixed size and maps it to a fixed size output of the same size as one of the inputs.

For example, a compression function might take as input two strings of 128 bits and produce an output string of 128 bits. The two strings can also be seen as one input of 256 bits, which is then compressed to produce an output of 128 bits (hence the name "compression function"). The two inputs can be of different sizes, but in this text it will be assumed that they are not. A hash function differs from a compression function in that a hash function maps arbitrary sized inputs to fixed-size outputs.

It is possible to construct a hash function (with arbitrary input and fixed output) out of a compression function. This is done by repeatedly applying the compression function to the message according to some predetermined algorithm. Such a hash function is called an iterative hash function.

A common way to create an iterative hash function is to use the Merkle-Damgård construction, which relies on one-way compression functions. The Merkle-Damgård construction first pads the message to be hashed so that it is a multiple of the input/output size, \( s \), of the compression function and then divides the message into \( n \) message blocks, \( m_i \), of size \( s \). The construction depends on an initialization vector (IV) (also of size \( s \)) and a compression function, \( f \). It is defined recursively according to...
\[ h_1 = f(IV, m_1) \]
\[ h_i = f(h_{i-1}, m_i) \text{ for } 1 < i \leq n \]

The final hash, \( h_n \), will be the output of the iterative hash function and will always have size \( s \), even for input of arbitrary size. The hashes \( h_i \) for \( 1 \leq i < n \) are called intermediate hashes.

It can be proven [5] that the iterative hash function produced in this way will be collision resistant if the padding is done appropriately and if the underlying compression function is collision resistant.

A message authentication code (MAC) is a digital fingerprint in the context of message transmission. If two persons, Alice and Bob, use a keyed hash function (a hash function that takes a key as input in addition to the plaintext), they can send both the message and the hash value over an insecure channel. For anyone not knowing their key it is difficult to tamper with either the message or the hash without detection. Bob can easily verify that a given message hasn’t been changed and that it came from Alice, since only she could attach the hash corresponding to their secret key. In this way, MACs provides both authentication and data integrity.

### 5.1 A simple hash function based on quasigroups

Perhaps the simplest version of a cryptographic hash function using a quasigroup is to compute a single digit-hash by repeated use of the quasigroup’s operation on the digits of the message. Let the message to be hashed be \((m_1, m_2, \ldots, m_q)\), then the hash would be found by

\[ h = (((m_1 * m_2) * \ldots) * m_q) \]

Note that the hash will be only one digit long. Such a short hash is by no means ideal, and more secure hash functions are described below.

### 5.2 A MAC using quasigroups

In [6], Denés and Keedwell suggests the following way of calculating MACs. Instead of calculating a single digit-hash they suggest that one splits the message into subsets of digits. The digits in a subset need not necessarily be consecutive digits in the message.

Let \((m_1, m_2, \ldots, m_q)\) be the message and assume that the MAC is to consist of \( s \) digits. Also let \( q = st \), then one can divide the message into \( s \) subsets of \( t \) digits each. If \( \{m_{i1}, m_{i2}, \ldots, m_{it}\} \) is the \( i \)th subset, the \( i \)th digit of the MAC, \( h_i \) is computed like in the simple hash function above, that is

\[ h_i = (((m_{i1} * m_{i2}) * \ldots) * m_{it}) \]

In this case the quasigroup used for the calculations and the specifics of how to divide the message into subsets can be seen as the key of the MAC, and should be kept secret from anyone but the persons involved in the communication. Anyone wanting to check the integrity of the message can simply divide the message into the prescribed subsets and calculate the digits of the MAC. If these agree with the MAC appended to the message the data can be assumed to be unmodified.
In [6] it is also explained how to adapt the scheme above when working with binary messages. Let the quasigroup consist of the elements in \( \mathbb{Z}_{2^m} \) and let each subset of the message, \( S_i \), consist of \( tm \) digits. A string of \( m \) binary digits can represent any of the integers in the interval \([0, 2^m - 1]\), and therefore it can be seen as representing an element of the quasigroup. Thus each subset \( S_i \) can be seen as \( t \) elements of the quasigroup.

Now one can proceed as above and calculate a hash digit, \( h_i \), by using the \( t \) elements in subset \( S_i \). The biggest difference is that each subset will give rise to a hash that is \( m \) binary digits long so that the MAC has length \( l = sm \) where \( s \) is the number of subsets.

### Figure 5.1: A latin square of order \( n = 2^2 \) with numbers written in binary

<table>
<thead>
<tr>
<th></th>
<th>00</th>
<th>01</th>
<th>10</th>
<th>11</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>10</td>
<td>11</td>
<td>00</td>
<td>01</td>
</tr>
<tr>
<td>01</td>
<td>01</td>
<td>10</td>
<td>11</td>
<td>00</td>
</tr>
<tr>
<td>10</td>
<td>11</td>
<td>00</td>
<td>01</td>
<td>10</td>
</tr>
<tr>
<td>11</td>
<td>00</td>
<td>01</td>
<td>10</td>
<td>11</td>
</tr>
</tbody>
</table>

**Example 5.1.** Let \( n = 2^2 = 4 \) and use the quasigroup, \( (Q, \ast) \), from figure 5.1 which consists of elements from \( \mathbb{Z}_4 \) written in binary. Assume that the message to be signed is: \( m = 1101001101011001 \). Divide the message into four subsets consisting of \( tm = 2 \cdot 2 \) binary digits each.

\[
S_1 = 1101 \\
S_2 = 0011 \\
S_3 = 0101 \\
S_4 = 1001
\]

Each of these subsets will give rise to a hash of two digits by seeing every pair of digits as an element of \( (Q, \ast) \) and calculating

\[
h_1 = 11 \ast 01 = 01 \\
h_2 = 00 \ast 11 = 01 \\
h_3 = 01 \ast 01 = 10 \\
h_4 = 10 \ast 01 = 00
\]

The MAC is now finalized by concatenating the hashes \( h_1, h_2, h_3 \) and \( h_4 \) which gives

\[01011000\]

In the case where the message length \( q \) isn’t an exact multiple of \( t \) one can easily modify the scheme so that the last subset contains fewer elements, or so that the last message is padded in some way that is agreed upon by the parties in the communication.

The security of the MAC relies on the following theorem, the proof of which can be found in [6].

**Theorem 5.1.** The probability that a certain digit should occur in a certain position in the signature is the same for all digits and positions, i.e., the signatures are equi-probable.
Remark. Note that for this procedure to produce a MAC of the desired length, $s$, the message must be of suitable length, $n$. In particular, one can’t have $n < s$ and still produce a MAC of length $s$. If $n < 2s$, one or more of the digits in the MAC will be an unaltered digit from the message, since they would be the lone members of their respective subsets. Therefore it must be considered insecure to have $n < 2s$.

5.3 Edon-$\mathcal{R}$

In [13] the e-transformation (see chapter 3) is used to create an infinite family of hash functions, Edon-$\mathcal{R}$.

The idea is to apply the e-transformation repeatedly with the digits in the message as leaders. To this end, the following definition is given.

Definition 5.2 (From [10]). Let $e_{l,*}(m_1,m_2,\ldots,m_q) = (c_1,c_2,\ldots,c_q)$ be the e-transformation with leader $l$ and operation $*$, then the reverse string transformation is

$$R_l = e_{m_q,*}(e_{m_{q-1},*}(\ldots(e_{m_1,*}(m_1,m_2,\ldots,m_q)\ldots)$$

To ensure the security of the hash function it is important that the quasigroup is of a certain type.

Definition 5.3. A quasigroup $(Q,*)$ of order $n$ that is non-associative, non-commutative, doesn’t have neither a right nor a left identity, doesn’t contain proper sub-quasigroups and has no $k < 2n$ such that

$$x*(x*(\ldots(x*y)\ldots)) = y$$

$$((\ldots(y*x)\ldots)*x)*x = y$$

is called a shapeless quasigroup.

In [13] it is conjectured that $R_l$ is a one-way function in a shapeless quasigroup. The security of the hash function is based on this conjecture.

The proposed hash algorithm uses padding and iterative use of $R_l()$ to compute the hash of the message, thus Edon-$\mathcal{R}$ is an iterative hash function.

Algorithm 5.1. Edon-$\mathcal{R}$ (From [13])

Input: $M$ - the message to be hashed  
$(Q,*)$ - a shapeless quasigroup of order $2^w$, for some $w \geq 4$  
$N$ - a positive integer such that the length of the output is $wN$ bits.

Output: A hash of $wN$ bits.

1. Pad the message so that the size of padded message, $M'$, is $|M'| = k \cdot wN$ bits, where $k$ is a positive integer. In other words, $M'$ can be seen as a multiple of $N$ words, each of length $w$ bits. The $w$-bit words will be seen as elements in $(Q,*)$. 
2. Set up an initial vector $H_0$ according to
\[ H_0 = (0 \mod 2^w, 1 \mod 2^w, \ldots, 2N - 1 \mod 2^w) \]
where the numbers should be seen as elements of $Q$.

3. Compute the hash according to
\[ H_i = R_1 (H_{i-1} || M_i) \mod 2^{2wN}, \text{ for } 1 \leq i \leq k \]
where $M_i$ is a substring of $M'$ that is $wN$ bits long. Let $M_1$ be the first $wN$ bits of $M'$, $M_2$ the next $wN$ bits, etc.

4. Finalize the hash by taking
\[ \text{Output} = H_k \mod 2^{wN} \]
and thus producing a hash of size $wN$ bits.

Note that throughout the algorithm $R_1$ works on strings of size $3wN$ bits, i.e. numbers that can be represented by at most $3wN$ binary digits. Taking such a number $\mod 2^{2wN}$ will produce a number that can be represented by at most $2wN$ binary digits, i.e. a number that can be seen as strings of size $2wN$ bits.

The algorithm can output hashes of arbitrary length, which makes it very versatile and gives it the possibility to adapt to a society with increasing computer power at its disposal.

Since there is an infinite number of quasigroups of different sizes and a given implementation of Edon-$R_1$ uses only one, one can argue that Edon-$R_1$ is actually an infinite family of cryptographic hash functions.

Gligoroski and Knapskog have given examples of implementations of Edon-$R_1$ in [11]. The given implementations produces hashes of length 256, 384, 512 bits, respectively. This article and [13] both state that implementing the algorithm in hardware might lead to higher speeds due to the possibility of running some operations in parallel.

The security of Edon-$R_1$ has been analyzed in subsequent, independent papers. A list of discovered attacks can be found in [9].
6 Secret sharing schemes

A secret sharing scheme is concerned with splitting and distributing a secret among a group of participants so that the secret can only be recovered when an authorized group of participants combine their shares. It is usually assumed that there is a certified authority called a dealer that is responsible for distributing the shares.

In the cases of the business transaction and nuclear weapon activation described in chapter 1, the secret might be a password needed to start the process.

Definition 6.1. A \((t, n)\)-threshold scheme is a type of secret sharing scheme in which the there are \(n\) shares and any set of \(t \leq n\) or more shares can be used to recover the secret, but any set of fewer shares can not.

A second type of secret sharing scheme is the multilevel scheme in which some participants shares might contribute more to unlocking the secret than others. One example of a multilevel scheme could be a scheme where the secret can be unlocked by the shares of either two higher ranking executives or the shares of one higher ranking executive and two lower ranking executives.

Depending on the scheme it is possible that a share, or a collection of shares, reveals partial information of the secret. Since the secret is supposed to stay hidden unless an authorized group accesses it, it is preferable that no such partial information can leak. A secret sharing scheme in which no partial information of the secret can be revealed by any group of participants that is not an authorized group is called a perfect secret sharing scheme.

In many cases it is conceivable that the participants may lie about their own shares, perhaps to gain access to other shares. To prohibit this, one can use a verifiable secret sharing scheme in which the validity of the participants shares can be verified.

Example 6.1. Assume that the secret to be distributed is the (perhaps poorly chosen) password: "password", and that it should be distributed to eight participants. A possible secret sharing scheme would be to assign each of the participants one of the letters as well as a number that represents which position the letter holds in the password, e.g. \((p, 1)\), \((a, 2)\) etc.

This kind of construction would be a \((8, 8)\)-threshold scheme since all eight of the shares would be necessary to know the entire password. The scheme would not be verifiable since there is no way for the participants to check the validity of the other participants shares, and it would not be perfect since all the participant would know at least one of the letters in the password, i.e. partial information of the secret.

6.1 Partial latin squares and critical sets

Some latin square-based secret sharing schemes will be described later in this chapter, but first it is necessary to introduce some terminology regarding latin squares that have one or more empty cells.
Definition 6.2. A partial latin square or order \( n \) is an \( n \times n \) array of symbols from the set \( N \) of size \( n \) such that each symbol occurs at most once in each row and column.

Note that this means that a partial latin square may contain empty cells. Recall that one can see a latin square as a set of ordered triples. This notation can be applied to partial latin squares as well.

Definition 6.3. A latin square, \( L \), is said to be a completion of a partial latin square, \( P \), if for all \( (i, j; k) \in L \) either the entry \( (i, j) \) in \( P \) is undefined or \( (i, j; k) \in P \).

Some partial latin squares can be completed to latin squares by filling the empty cells appropriately, and some can not (see figure 6.1).

\[
\begin{array}{ccc}
2 & 3 & 1 \\
3 & 1 & 2 \\
\end{array}
\quad
\begin{array}{ccc}
1 & 3 & 2 \\
3 & 1 & 2 \\
\end{array}
\]

(a) Completable to a latin square  
(b) Not completable to a latin square

Figure 6.1: A pair of partial latin squares of order 3

Definition 6.4. A critical set in a latin square \( L \) of order \( n \) is a partial latin square \( C = \{(i, j; k)\} \) such that:
1. \( L \) is the only latin square of order \( n \) which has the element \( k \) in position \( (i, j) \) for each \( (i, j; k) \in C \).
2. No proper subset of \( C \) satisfies 1.

Thus a critical set has a unique completion to a latin square.

Definition 6.5. A minimal critical set in a latin square \( L \) is a critical set of minimum cardinality.

Example 6.2. Figure 6.2 shows a latin square and some of its minimal critical sets. They are minimal since no partial latin square with only one non-empty cell can have a unique completion to a latin square. An example of a critical set that is not minimal can be seen in figure 6.4b.

\[
\begin{array}{ccc}
1 & 2 & 3 \\
2 & 3 & 1 \\
3 & 1 & 2 \\
\end{array}
\quad
\begin{array}{c}
2 \\
1 \\
\end{array}
\quad
\begin{array}{c}
2 \\
3 \\
\end{array}
\quad
\begin{array}{c}
\end{array}
\]

Figure 6.2: A latin square of order 3 and some of its minimal critical sets

6.2 A secret sharing scheme using critical sets

This section is divided into two parts. First the scheme will be presented, and after that a flaw that has been found in it will be discussed.
6.2.1 The secret sharing scheme

Cooper, Donovan and Seberry [4] propose a secret sharing scheme using critical sets in latin squares. In this scheme the secret is a latin square, \( L \), of order \( n \). While \( L \) is kept secret the order \( n \) is made public.

The shares of the scheme are taken from a partial latin square defined as the union of a number of critical sets in \( L \):

\[
P = \bigcup_{i \in I} C_i \quad \text{where} \quad C_i \text{ is a critical set in } L
\]

The number of critical sets used depends on \( L \) and the number of participants in the scheme. The shares are taken to be the entries in \( P \) so that each of \((i, j; k) \in P\) is given to a participant. A group of participants whose shares make up a critical set can pool their shares and thus acquire a critical set which uniquely determines the latin square, \( L \). The security of the scheme requires that there are a large number of completions for every possible subset \( A \) of a critical set, otherwise a group of participants with less than the required amount might pool their shares and be able to guess the secret among a few possible completions.

**Example 6.3.** For a small example with three participants and using the latin square from figure 6.2 one can take the partial latin square to be \( P = \{(1, 2; 2), (2, 3; 1), (3, 1; 3)\} \). If one gives one of these shares to each of the participants one obtains a \( (2, 3) \)-threshold scheme since any two of the participants can pool their shares to acquire a critical set. Once they have a critical set they can find the secret, i.e. the original latin square.

The authors also show how using this approach makes it possible to design multilevel schemes as well as schemes for scenarios where one participant needs to be a part of several different secret sharing schemes at the same time.

To construct a multilevel scheme one takes \( P \) to be the union of critical sets of which not all have the same cardinality. In this way, some critical sets can be formed by the shares of a smaller group of participants. These shares can be thought of as more influential so that this construction is a multilevel scheme.

**Example 6.4.** For a multilevel scheme based on the latin square from figure 6.2 one can use the critical sets found in figure 6.3. If one distributes the shares \((1, 2; 2), (1, 3; 3), (3, 1; 3), (3, 3; 2)\) among four participants the number of shares needed to find a critical set and unlock the secret varies depending on which participants are pooling their shares.

\[
\begin{array}{ccc}
3 & & \\
2 & 3 & \\
3 & 2 &
\end{array} \quad \begin{array}{ccc}
2 & 3 & \\
2 & & \\
& &
\end{array} \quad \begin{array}{cc}
2 & \\
& 3
\end{array}
\]

**Figure 6.3:** Critical sets in the latin square from table 6.2 that can be used in a multilevel secret sharing scheme

When discussing scenarios where one participant needs to be part of several different schemes [4] gives the example of a medical administrator who needs access to patient data, organ bank data and hospital resources, all of which are restricted. In this example each of the data banks can be accessed through a different secret sharing scheme. To design such a scheme one needs one share \((i, j; k)\) which is common to all of the secret sharing schemes. To accomplish this it is possible to either choose a different secret, i.e. latin square, for
each of the schemes or to choose the same secret but distribute different critical sets to the participants of different schemes. In the latter case it is important that the secret can’t be unlocked even if participants from all schemes pool their shares.

**Example 6.5.** An example of critical sets to use when one participant, \( p_1 \), needs to be part of two different schemes at the same time can be found in figure 6.4a and 6.4b. In this case the share \((3,3;2)\) is needed to unlock each of the two secret sharing schemes and would be suitable to give to \( p_1 \). As can be seen in 6.4c the other participants’ shares does not uniquely define a latin square and thus these participants will not be able to unlock the secret even if they pool all of the shares across both schemes.

### 6.2.2 A flaw in the scheme

Grannell, Griggs and Street [14] as well as Donovan et al. [7] has pointed out a flaw in the proposed scheme. The main concern is that the scheme isn’t perfect and that if one knows that the scheme uses a critical set one might be able to access the secret with less than an authorized amount of shares.

Assume that \( t \) shares are needed to find the critical set, \( C \), and unlock the secret. Also assume that one knows this and has access to \( t - 1 \) shares. Call the partial latin square defined by the \( t - 1 \) shares \( P \). Look at all possible completions of \( P \). Since \( C \) is a critical set that defines a unique latin square only one of these completions will include \( C \), and the missing share must be unique to this latin square.

Look at all elements that occur in exactly one completion of \( P \). These elements are the ones that could potentially be the missing share, since appending them to \( P \) uniquely defines a latin square. If all of these elements occur in the same completion that square must be the secret.

The discussion above gives rise to the following definition.

**Definition 6.6.** A triple of a critical set, \((i,j;k) \in C\), is called bad if all triples which occur in exactly one completion of \( C \setminus (i,j;k) \) occur in the same one.

**Example 6.6.** An example (originally from [7]) of a critical set, \( C \), with a bad triple can be seen in figure 6.5. By finding all possible completions of the partial latin square \( C \setminus (2,3;4) \) one finds that \((2,3;4)\) is bad.

Donovan et al. [7] studies this flaw in latin squares and proves that for a certain infinite family of critical sets most of the elements exhibit a bad triple.
In [2], Chaudhry, Ghodosi and Seberry propose a perfect secret sharing scheme using Room squares (an object similar to latin squares) but also note that the same ideas can be adapted to a scheme using latin squares. This text focuses on the adapted scheme.

This scheme requires the set of symbols, \( N \), as well as the index for the rows and the columns, to be the elements in \( \mathbb{Z}_n \), where \( n \) is the order of the latin square. For example, the first entry in figure 1.2 is \((0,0;0)\), using the same notation as before.

The idea behind the scheme is to give all participants randomized shares that, counting in \( \mathbb{Z}_n \), add up to a critical set. This critical set, or the latin square that it can be completed to, is the secret of the scheme.

Given a critical set, \( C \), with \( |C| = c \), index the triples in \( C \), \((i_s, j_s; k_s)\) so that \( 1 \leq s \leq c \). Each participant’s share will consist of \( c \) triples. With \( p \) number of participants, choose the triples of the first \( p - 1 \) shares independently and at random from all possible values of \((\mathbb{Z}_n, \mathbb{Z}_n, \mathbb{Z}_n)\); one gets \((x_{r,s}, y_{r,s}; z_{r,s})\) where \( 1 \leq r \leq p - 1 \) is the index for the participants and \( 1 \leq s \leq c \) is the index for the triples. The last share is calculated in \( \mathbb{Z}_n \) according to:

\[
(x_{p,s}, y_{p,s}; z_{p,s}) \equiv (i_s, j_s; k_s) - \sum_{r=1}^{p-1} (x_{r,s}, y_{r,s}; z_{r,s}) \pmod{n}, \quad 1 \leq s \leq c
\]

To unlock the secret all of the participants need to pool their shares since adding them in \( \mathbb{Z}_n \) gives:

\[
\sum_{r=1}^{p} (x_{r,s}, y_{r,s}; z_{r,s}) \equiv (i_s, j_s; k_s) \pmod{n} \quad 1 \leq s \leq c
\]

Thus they can find all of the triples \((i_s, j_s; k_s) \in C\), \( 1 \leq s \leq c \) which constitute the critical set.

Since the critical set is revealed by taking the sum of all shares (as defined above), any collection of less than \( p \) shares will fail to reveal any partial information of the secret. If one has access to \( p - 1 \) shares, finding the critical set involves guessing \( c \) additional triples, which is as difficult as guessing the critical set without any knowledge other than the size of the latin square and its critical set. This shows that the secret sharing scheme is perfect.

**Remark.** Having access to only one share will reveal the size of the critical set, \( c \). Thus one can argue that the scheme is not perfect in the strictest sense. However, if one assumes that the size of the critical set is already publicly known, the scheme is indeed perfect.

While the scheme is perfect, it is not very flexible since either all of the participants are needed to unlock the secret or a different set of shares is needed for every authorized group of participants.
6.4 A secret sharing scheme using autotopisms

This section will make use of autotopisms, which were introduced in chapter 1, to define a secret sharing scheme, but first some new notation will be introduced.

6.4.1 Contours

Contours are related to autotopisms, and gives a way to generate a certain latin square using an autotopism of the square.

Definition 6.7. Let $L$ be a latin square. A contour is a partial latin square, $C$, of $L$ such that the entries of $C$ is exactly one element from each cycle of $L$ under the autotopism $\theta$.

Since $\theta$ can permute one element of a cycle into all of the other elements of the same cycle, one can generate a unique latin square, $L$, from $C$ and $\theta$. One can say that $(C, \theta)$ generates $L$.

![Figure 6.6: A latin square and one of its contours](image)

Example 6.7. The contour in figure 6.6b (with elements $(1, 2, 2), (1, 3, 3), (3, 3, 2)$) together with the autotopism $\theta$ where

$\alpha, \beta = \begin{pmatrix} 1 & 2 & 3 \\ 2 & 3 & 1 \end{pmatrix}$

and

$\gamma = \begin{pmatrix} 1 & 2 & 3 \\ 3 & 1 & 2 \end{pmatrix}$

generates the latin square in figure 6.6a since the cycles of $\theta$ are

$((1, 1, 1)(2, 2, 3)(3, 3, 2)), ((1, 2, 2)(2, 3, 1)(3, 1, 3)), ((1, 3, 3)(2, 1, 2)(3, 2, 1))$

and thus each cycle contains exactly one of the elements of the contour.

6.4.2 A verifiable scheme using an autotopism as secret

In this scheme, first described in [23], an autotopism is used as secret and a corresponding contour of a latin square is used for verification. The set of symbols used throughout the scheme is $\mathbb{Z}_n$. Due to the nature of the autotopism and the corresponding contour, $n$ will be required to be $n \equiv 2$ (mod 4).

Finding a suitable autotopism and corresponding contour can be difficult. Therefore, an autotopism of the form

$\zeta = (\tau, \tau, \tau)$, where $\tau = (0, 1, \ldots, n/2 - 1)(n/2, n/2 + 1, \ldots, n - 1)$

will be used as a starting point.
According to a theorem in [23] one can randomly generate a contour $D$ of order $n$ that corresponds to $\zeta$ by randomly taking integers $z_k \in \{0,n/2\}$ for $k \in \{0,1,\ldots,n/2-1\}$ and choosing
\[ d_{n/2-1-k,k} = d_{n-1-k,n/2+k} = z_k \]
\[ d_{n-1-k,k} = d_{n/2-1-k,n/2+k} = n/2 - z_k \]
where $d_{i,j}$ is the entry in row $i$ and column $j$ of $D$.

Let $L_{\text{prior}}$ be the latin square generated by $(D, \zeta)$. Replace each entry, $(i, j; d_{i,j})$, in $D$ with $\zeta'(i, j; d_{i,j})$ where $t \in \{0,1,\ldots,n/2-1\}$ is randomly chosen for each entry (note that $\zeta^0$ is the identity permutation). In other words, each entry in $D$ is replaced by another entry in the same cycle under $\zeta$. This results in a new, random contour $C_{\text{prior}}$ such that $(C_{\text{prior}}, \zeta)$ also generates $L_{\text{prior}}$.

After this one should take a random isotopism $\varphi$ and define
\[ C = \varphi(C_{\text{prior}}) \text{ and } \theta = \varphi \zeta \varphi^{-1} \]
Thus, $(C, \theta)$ generates the latin square $L = \varphi(L_{\text{prior}})$, since $L_{\text{prior}}$ admitting the autotopism $\zeta$ means that $L$ will admit the autotopism $\theta$. Note that $\theta$ will have the same cycle structure as $\zeta$ (see the definition of $\tau$ above) due to a theorem from [1].

The shares of the scheme will be isotopisms $\sigma_1, \sigma_2, \ldots, \sigma_p$ such that $\sigma_1 \sigma_2 \ldots \sigma_p = \theta$
where $p$ is the number of participants. To make this secure one should take the first $p-1$ isotopisms at random and the last one to be $\sigma_p = \sigma_{p-1}^{-1} \sigma_{p-1} \ldots \sigma_1 \theta$. This means that the secret can be recovered by pooling the shares and finding $\theta = \sigma_1 \sigma_2 \ldots \sigma_p$. This method of recovering the secret is similar to the one in the perfect secret sharing scheme described in section 6.3.

To make the scheme verifiable a new contour
\[ C_{\text{public}} = \xi(C), \text{ where } \xi = \sigma_p \sigma_{p-1} \ldots \sigma_1 \]
should be generated. This contour can be made public or can be known only to a dealer, if the scheme relies on one to verify the shares. As a security measure one should make sure that $\theta \neq \xi$. If $\theta = \xi$ the process needs to be restarted from the beginning.

When all of the shares of the scheme have been pooled, the contour $C_{\text{public}}$ can be used to verify that $\theta$ is the correct autotopism by calculating
\[ C' = \xi^{-1}(C_{\text{public}}) = \sigma_1^{-1} \sigma_2^{-1} \ldots \sigma_p^{-1}(C_{\text{public}}) \]
and then using an algorithm from [23] to make sure that $(C', \theta)$ generates a latin square. If all of the shares are valid one will have that $C' = C$ so that the resulting square is, indeed, a latin square. If some returned share is incorrect the probability that the result of the shares $\sigma_1 \sigma_2 \ldots \sigma_{\text{incorrect}} \ldots \sigma_p = \theta_{\text{incorrect}}$ will generate a latin square is relatively low, as is shown in [23].

**Example 6.8.** In this example $n = 6$ (thus $n = 2 \pmod{4}$). This means that
\[ \zeta(\tau, \tau, \tau) \text{ where } \tau = (0,1,2)(3,4,5) \]
Three integers are randomly generated: $z_0 = 0$, $z_1 = 1$, $z_2 = 1$. The resulting random contour $D$ corresponding to $\zeta$ can be seen in figure 6.7a. In order to find a contour $C_{\text{prior}}$ the following sequence of numbers is randomly generated
\[ (t_{0,2}, t_{0,5}, t_{1,1}, t_{1,4}, t_{2,0}, t_{2,3}, t_{3,2}, t_{3,5}, t_{4,1}, t_{4,4}, t_{5,0}, t_{5,3}, \ldots) = (0,0,2,0,2,2,1,2,0,0,1,1) \]
where the entries in $C_{prior}$ are given by $\zeta_{t(i,j)}$ for all entries $(i,j,d_{i,j})$ in $D$. The new contour can be seen in figure 6.7b.

A random isotopism $\varphi = (\alpha, \beta, \gamma)$ is generated, where:

$$\alpha = (12)(0345), \quad \beta = (043215), \quad \gamma = (035124)$$

The new contour, $C = \varphi(C_{prior})$, can be seen in figure 6.7c and $\theta = \varphi \zeta \varphi^{-1} = (\alpha', \beta', \gamma')$ is defined by

$$\alpha' = (045)(132), \quad \beta' = (023)(145), \quad \gamma' = (015)(243)$$

In order to keep this example simple it is assumed that this is a secret sharing scheme with only two shares. The first share is randomized and is given by:

$$\sigma_1 = (\alpha_1, \beta_1, \gamma_1) = (\langle 0 \rangle(15243), (1)(02534), (0132)(45))$$

The other share is calculated according to:

$$\sigma_2 = \sigma_1^{-1} \theta = (\langle 0235 \rangle(14), (0\langle 13425 \rangle, (0\langle 3 \rangle(14)(25))$$

Forgoing the process of making the scheme verifiable, the shares can now be distributed. It can be verified that $(C, \theta)$ generates a latin square.

### 6.5 A secret sharing scheme using cryptographic hash functions

This section will once again make use of the concept of cryptographic hash functions introduced in chapter 5, but this time hash functions with known weaknesses will be of interest. This is since a form of cryptographic attack will be used to define the secret sharing scheme.

#### 6.5.1 Nostradamus attack

Recall the notion of iterative hash functions and intermediate hashes from chapter 5. A *Nostradamus attack* or *herding attack* [16] uses collisions in an iterative hash function, $f$, to create a diamond structure (see figure 6.8) of intermediate hashes. The attacker might then "herd" many different possible messages into a given final hash through iterative application of $f$.

Each intermediate hash, $h_i$, leads to another hash in the structure by appending the corresponding message block, $m_i$, and taking $f(h_i|m)$. In this way an attacker might pretend to predict the result of an event by publishing a hash, $H$, in advance. When the event has happened, the attacker knows the results and can...
produce it in the form of a string $R$. Thus they want to find a way to produce the original
hash $H$ from $R$.

To do this the attacker need only find a string $S$ to append to $R$ so that $f(R||S)$ yields
an intermediate hash in the diamond structure. After finding an intermediate hash, $h_a$, the
attacker finds the messages, $m_1, m_2, ..., m_n$, that lead from $h_a$ to the hash $H$ in the diamond
structure, see figure 6.8. By appending these messages to $R||S$ the attacker can create a
string $R||S'$ with hash value $H$, where $S'$ is $S$ together with the appended messages.

To make the prediction more believable the string $S$ and appended messages should not
just be random noise, but rather contain information that might or might not be relevant to
the prediction.

This attack can be used to circumvent so called commitment schemes. In such a scheme
one party commits to a certain value or statement, while keeping it secret from other parties
in the scheme. The committed party might then have the possibility to reveal the value or
statement at a later time. In a secure commitment scheme one shouldn’t be able to change
the value or statement after one has committed to it. A Nostradamus attack clearly breaches
this property.

6.5.2 A secret sharing scheme based on the Nostradamus attack

In [3], Chum and Zhang propose a way to use latin squares and a cryptographic hash
function to construct a perfect secret sharing scheme. The scheme relies on the secret (in
this case a latin square) to be in the form of a hash. Because of this, it isn’t necessary for
the secret to be a latin square. Anything that can be stored as a hash can act as secret in this
scheme.

The scheme uses a diamond structure like the one used in the Nostradamus attack to-
gether with publicly available linking messages to recover the secret, and therefore doesn’t
depend on critical sets like many of the other schemes described in this chapter.

First one needs to decide the order of the latin square and which hash algorithm one will use. Note that, since the last row and column of a latin square are uniquely determined by the rest of the square (see figure 6.9) it is unnecessary to store these in the hash. The hash can thus be made shorter than it would have been if it contained the entire latin square.

\[
\begin{array}{cccccccc}
0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 \\
1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 \\
2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 0 \\
3 & 4 & 5 & 6 & 7 & 8 & 9 & 0 & 1 \\
4 & 5 & 6 & 7 & 8 & 9 & 0 & 1 & 2 \\
5 & 6 & 7 & 8 & 9 & 0 & 1 & 2 & 3 \\
6 & 7 & 8 & 9 & 0 & 1 & 2 & 3 & 4 \\
7 & 8 & 9 & 0 & 1 & 2 & 3 & 4 & 5 \\
8 & 9 & 0 & 1 & 2 & 3 & 4 & 5 & 6 \\
\end{array}
\]

**Figure 6.9:** A uniquely completable partial latin square of order 10

The shares of the scheme will be randomized strings of roughly the same size as the hashed secret. If an authorized group of participants pool their shares they can concatenate their strings into a message, \( M_{\text{priv}} = m_1 || m_2 || \ldots || m_k \), where \( m_1, m_2, \ldots, m_k \) are the pooled shares. When preparing the scheme, the dealer should have prepared a diamond structure based on the cryptographic hash function \( f \) consisting of all possible messages \( M_{\text{priv}} \) and corresponding strings \( M_{\text{pub}} \) linking them to the secret \( H \) (like in fig. 6.8). The strings \( M_{\text{pub}} \) as well as the hash function \( f \) should be public knowledge. Note that it is also important that the order in which participants should concatenate their shares is specified.

When a group of participants have pooled their shares and found \( M_{\text{priv}} \) they can concatenate the linking message \( M_{\text{pub}} \) and apply the hash function \( f \) to it. This will result in \( f(M_{\text{priv}} || M_{\text{pub}}) = H \), and thus the secret has been unlocked. Note that authorized groups can be chosen arbitrarily as long as a suitable linking message can be found.

If the shares are about as long as the hash of the secret the scheme is perfect, since even if an unauthorized group managed to pool \( t - 1 \) out of \( t \) necessary shares, guessing or finding the last share would be as difficult as guessing or finding the secret itself.

When setting up the diamond structure with randomized shares it might be difficult to ensure that the final hash \( H \) is the required latin square, or even a latin square at all. Because of this, [3] proposes that one generates many different latin squares as potential secrets and then finds a message that links \( H \) to one of these latin squares, which is chosen as the secret. The linking message is then made public. The search for a suitable linking message is simplified by having many potential latin squares to link to.

The authors also suggest a way to make the scheme verifiable by using an additional hash function, \( g \). In that case, the dealer would apply \( g \) to each share before handing them out, and then make both the function \( g \) and the resulting hashes public. If one wants to verify a given share, one can do so by applying \( g \) to it and comparing the result with the publicly available hash that the dealer calculated. If the two hashes match one can assume that the share is reliable.

If \( g \) is secure the public hashes will not reveal any information about individual partici-
pants’ shares.

**Example 6.9.** Consider a secret sharing scheme based on the hash function defined in section 5.1, i.e. where the hash of a string \((m_1, m_2, \ldots, m_q)\) is given by

\[
h = (((m_1 * m_2) * \ldots) * m_q)
\]

Call the hash function \(f\) and let the underlying quasigroup be given by fig. 6.10. Since the hash function produces hashes that are only one symbol long it is easy to find collisions, and the secret has to consist of only one symbol, i.e. it is a latin square of order 1. Both of these properties make the hash function suitable for this example, but unreliable for practical use.

Assumed that the secret is \(H = 1\). Note that no linking message is needed to ensure that \(H\) is a latin square of order 1. Further assume that the scheme has three participants, \(p_1, p_2\) and \(p_3\), and that they are given the strings \(s_1 = (1)\), \(s_2 = (2)\) and \(s_3 = (3)\) as shares, respectively. Also say that at least two participants are required to unlock the secret (i.e. that this is a \((2, 3)\)-threshold scheme) and that when concatenating strings, \(s_1\) will always go first and \(s_2\) will always go before \(s_3\). This means that the possible messages \(M_{priv}\) are

\[(1, 2), (1, 3), (2, 3), (1, 2, 3)\]

It is now easy for the dealer to calculate the linking messages \(M_{pub}\). For example, since \(f(s_1 || s_2) = 1 * 2 = 1\) and the secret is \(H = 1\) the message linking the string \(M_{priv} = (1, 2)\) to the secret is \(M_{pub} = 2\). By calculating the linking messages for the other possible messages in the same way one finds that the diamond structure for the scheme is the one that can be seen in fig. 6.11. If the dealer makes the linking messages as well as the hash function and quasigroup public any authorized group of participants will be able to unlock the secret.

![Figure 6.10: A multiplication table of a quasigroup of order 3.](image)

**Figure 6.10: A multiplication table of a quasigroup of order 3.**

![Figure 6.11: Diamond structure of the scheme in example 6.9](image)

**Figure 6.11: Diamond structure of the scheme in example 6.9**
7 Conclusions and further research

Many different cryptosystems and secret sharing schemes based on latin squares and quasi-groups have been treated in this thesis. Out of these, Edon80 and Edon-R are probably most apt for use in modern cryptography, perhaps together with one of the more sophisticated secret sharing schemes. The other schemes and cryptosystems are interesting in their own right but might not be of practical use due to security or implementational issues.

As was stated in the beginning of this text, this thesis has focused on only some of the many articles relating to this area of cryptography. The main area of interest is quasigroups, due to their non-associative nature. Thus, quasigroup-related cryptography is one interesting area of research that could very well lead to some strong and practical cryptosystems. On the same note, some further research on the strength of Edon80 and Edon-R would be good for determining their practical usefulness.

In chapter 2 of this thesis two new ciphers based on isotopisms were described. The ciphers seem simple enough to break with the aid of computers, but one area of study would be to describe the ciphers’ strengths and weaknesses.
References


